Low-Rank Adaptation for Critic Learning in Off-Policy Reinforcement Learning

Yuan Zhuang[1]* **Yuexin Bian**[2]* **Sihong He**[3] **Jie Feng**[2] **Qing Su**[1]
Songyang Han **Jonathan Petit**[4] **Shihao Ji**[1] **Yuanyuan Shi**[2] **Fei Miao**[5]†

[1]University of Connecticut [2]University of California San Diego
[3]University of Texas at Arlington [4]Qualcomm [5]Zhejiang University

yuan.2.zhuang@uconn.edu yubian@ucsd.edu sihong.he@uta.edu jif005@ucsd.edu
sq0925@gmail.com songyang418@gmail.com petit@qti.qualcomm.com
shihao.ji@uconn.edu yyshi@ucsd.edu feimiao@zju.edu.cn

Abstract

Scaling critic capacity is a promising direction for improving off-policy reinforcement learning (RL). However, recent work shows that larger critics are prone to overfitting and instability in replay-based bootstrapped training. In this paper, we propose using Low-Rank Adaptation (LoRA) as a structural regularizer for critic learning from scratch. Our approach freezes randomly initialized base matrices and optimizes only the corresponding low-rank adapters, thereby constraining the learned weight changes to low-rank residuals. We evaluate our method on state-based continuous-control tasks, using SAC with SimbaV2 and BRC on DMC-Hard, and FastTD3 with its SimbaV2 variant on IsaacLab. Empirically, LoRA reduces critic loss during training and improves overall policy performance, achieving the best or competitive results on most tasks. Extensive experiments demonstrate that our low-rank updates provide a simple and effective form of structural regularization for critic learning in off-policy RL. The code is available at https://github.com/paulzyzy/LoRA_RL.

1 Introduction

Off-policy reinforcement learning (RL) has become a leading paradigm for continuous control because it can reuse past experience through replay, substantially improving sample efficiency over on-policy methods [1, 2]. At the core of these methods is the *critic*, which estimates action values and provides the learning signal for policy improvement [2]. Since both policy evaluation and policy improvement depend on the quality of the learned value function, critic estimation plays a central role in the stability, scalability, and final performance of off-policy RL [3].

Motivated by this central role, a growing body of work has sought to improve off-policy RL primarily through better *critic design*. Early work focused on reducing estimation bias. Double Q-learning was introduced to mitigate the overestimation induced by the max operator by decoupling action selection from action evaluation [4], and this idea later became a standard ingredient in deep RL critics [5]. More recent works have revisited the critic architecture itself. CrossQ showed that carefully incorporating Batch Normalization can significantly improve sample efficiency in off-policy actor-critic learning [6]. SimBa proposed an RL backbone with normalization and residual design choices that enable more effective scaling of network capacity [7]. Building on this line, SimbaV2

*Equal contribution.

†Corresponding author: Fei Miao, Zhejiang University (email: feimiao@zju.edu.cn). This work was mostly done while the author was at University of Connecticut; part of the supplementary work was conducted at Zhejiang University.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

introduced hyperspherical normalization together with distributional value estimation to stabilize optimization in larger RL critics [8]. Another line of work revisited the critic learning objective. Distributional RL models the full return distribution rather than only its expectation, leading to more informative and stable value learning [9]. Building on this perspective, Farebrother et al. [3] showed that replacing value regression with classification can substantially improve the scalability of value learning.

More recently, several works have highlighted the promise of scaling in continuous-control RL [10, 11, 7, 12]. TD-MPC2 demonstrated that increasing model size can substantially improve robustness and performance in continuous control [11]. BRO further showed that larger critics can be highly effective when paired with strong regularization and optimistic exploration [12]. At the same time, recent studies suggested that simply scaling dense RL networks is often insufficient. Although larger critics can improve expressiveness and have contributed to the strong performance of recent RL methods [8], off-policy critic learning is particularly vulnerable to failure under scale because it relies on bootstrapped targets learned from non-stationary replay data. As a result, larger dense critics can overfit, become harder to optimize, and fail to realize the full benefits of increased capacity [13]. These findings indicate that successful critic scaling requires not only larger models, but also appropriate structural regularization.

In this work, we follow this line of thinking and propose an adaptive form of structured regularization for off-policy critics. Our key idea is to use Low-Rank Adaptation (LoRA) [14] not as a parameter-efficient fine-tuning method for pretrained models, but as a structural regularizer for critic learning from scratch. Specifically, we freeze the randomly initialized dense critic backbone and optimize only low-rank residual updates to its weights. This preserves a dense forward representation through the fixed random backbone, while restricting learning to a low-dimensional update subspace. In this sense, our method is closely related to sparsity-based regularization for RL [13]. While prior work regularizes the critic by statically setting a subset of weights to zero and inducing sparsity in the forward computation, our method keeps the critic network dense and regularizes learning by restricting weight updates to a low-dimensional subspace. We instantiate this idea in two commonly adopted architectures, SimbaV2 [8] and BRC [15]. For SimbaV2, where hyperspherical weight normalization makes direct LoRA integration nontrivial, we develop a compatible LoRA design that preserves the frozen base matrices while updating only the low-rank adapters, ensuring that the resulting effective weights remain on the hyperspherical geometry.

We summarize our main contributions as follows:

- **LoRA as structured sparse regularization for critic learning.** We propose leveraging LoRA as a structural regularizer for off-policy critic learning, where randomly initialized base matrices are frozen and only the corresponding low-rank adapters are optimized. By constraining critic updates to a low-dimensional subspace, our approach mitigates critic overfitting and improves performance on challenging RL tasks. To the best of our knowledge, this is the first systematic study of LoRA as a regularization mechanism for critic learning in off-policy RL.
- **Generality across algorithms and architectures.** Extensive experiments demonstrate that the proposed LoRA regularization principle generalizes across multiple off-policy RL settings. Specifically, we consider three representative settings: SAC with SimbaV2 [8] on DMC-Hard tasks, SAC with BRC [15] on DMC-Hard tasks, and FastTD3 with SimbaV2 [16] on IsaacLab robotics tasks. Across these algorithms, architectures, and benchmarks, LoRA achieves the best or competitive results on most tasks.

2 Related Work

Scaling Model Capacity in Reinforcement Learning. While larger models consistently improve supervised and self-supervised learning [17], scaling in RL has proven far more delicate [18, 19]. Naively widening or deepening value-function networks often degrades performance due to overfitting, plasticity loss, and optimization instability [20, 21, 22]. As a result, several recent works have explored architectures and training strategies that make scaling more effective in RL. SpectralNet [23] uses spectral normalization to stabilize large critics during training. SimBa [7] introduces a simplicity-biased residual architecture that enables effective parameter scaling across RL algorithms. BRO [12] combines strong ℓ_2 regularization, optimistic exploration, and high replay ratios to improve continuous-control performance at larger critic scales. SimbaV2 [8] further improves scalability

through hyperspherical normalization of features and weights together with distributional critics. FastTD3 [16] shows that parallel simulation and large-batch training can make scaled TD3-style agents competitive on challenging humanoid tasks. However, these advances do not fully resolve the overfitting and instability of critic learning under replay-based training. This has motivated a complementary line of work on regularization techniques for critic learning.

Regularization Techniques for Critic Learning. In supervised learning, overfitting is commonly mitigated through standard techniques such as weight decay [24] and dropout [25]. A related perspective comes from parameter-efficient fine-tuning, where constraining optimization to a sparse or low-dimensional subset of parameters can act as an implicit regularizer and often match or even outperform full fine-tuning [26, 14]. Critic learning in RL shares similarities with supervised learning, but is further complicated by distribution shift arising from replay data and bootstrapped targets. Prior work has therefore explored RL-specific approaches to controlling overfitting. DroQ improves critic stability at high update-to-data ratios through dropout-based Q-functions [27]. More recently, Ma et al. [13] show that static network sparsity induced by one-shot random pruning can mitigate optimization pathologies such as plasticity loss and gradient interference in large RL networks, thereby helping unlock the scaling potential of deep RL. Our work is most closely related to this line of research, but instead of imposing a fixed sparse structure, we improve critic robustness through adaptive low-rank updates that regularize how the critic can change during training. We further show that this design yields lower critic loss during training and consistently improves RL performance.

3 Preliminaries

We first review the off-policy RL algorithms and neural network architecture backbones used in our study. In particular, we consider Soft Actor-Critic (SAC) [1] and FastTD3 [16] as representative off-policy algorithms, together with recent state-of-art backbones including SimbaV2 [8] and BRC [15].

3.1 Off-Policy Actor-Critic Algorithms

Soft Actor-Critic. SAC [1] learns a stochastic policy $\pi_\theta(a \mid s)$ under the maximum-entropy objective

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t\geq 0}\gamma^t\big(r(s_t,a_t) + \alpha_{\text{ent}}\mathcal{H}(\pi(\cdot \mid s_t))\big)\right], \tag{1}$$

where α_{ent} is the entropy temperature. In scalar notation, the soft Bellman target is

$$y_{\text{SAC}} = r + \gamma^n m\left(Q^{\text{agg}}_{\bar{\phi}}(s',a') - \alpha_{\text{ent}}\log\pi_\theta(a' \mid s')\right), \qquad a' \sim \pi_\theta(\cdot \mid s'). \tag{2}$$

In Eqs. (2) and (6), n denotes the backup horizon and $r = \sum_{k=0}^{n-1}\gamma^k r_{t+k}$ denotes the accumulated discounted reward, where r_{t+k} is the immediate reward at step $t+k$. The next state is $s' = s_{t+n}$, and $m \in \{0,1\}$ is the bootstrap mask: $m = 1$ retains the bootstrap term and $m = 0$ omits it. The aggregation Q^{agg} depends on the critic design: SimbaV2 uses either a single critic or clipped double Q-learning depending on the environment setting, while BRC uses an ensemble average. The SAC actor minimizes

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\theta}\left[\alpha_{\text{ent}}\log\pi_\theta(a \mid s) - Q^{\text{agg}}_\phi(s,a)\right]. \tag{3}$$

Target critics are updated by Polyak averaging, where ϕ denotes the online critic parameters, $\bar{\phi}$ denotes the target critic parameters, and $\tau \in (0,1]$ is the target update rate.

$$\bar{\phi} \leftarrow (1-\tau)\bar{\phi} + \tau\phi. \tag{4}$$

FastTD3. FastTD3 [16] is a high-throughput variant of TD3 [2] that combines massively parallel simulation, large-batch updates, target-policy smoothing, and distributional critics. It uses a deterministic policy μ_θ and constructs the target action

$$\tilde{a}' = \text{clip}(\mu_{\bar{\theta}}(s') + \text{clip}(\epsilon, -c, c),\, a_{\min}, a_{\max}), \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I). \tag{5}$$

The scalar target is

$$y_{\text{TD3}} = r + \gamma^n m\, Q^{\text{agg}}_{\bar{\phi}}(s', \tilde{a}'). \tag{6}$$

The actor minimizes $-\mathbb{E}_{s\sim\mathcal{D}}[Q^{\text{agg}}_\phi(s, \mu_\theta(s))]$ with delayed policy updates.

3.2 Categorical Critic Learning

Categorical value learning has recently been widely adopted to improve critic learning in off-policy reinforcement learning [9, 3]. Both SimbaV2 and BRC build on this idea by using categorical value learning for their critics. Instead of directly outputting a scalar value, the critic predicts a categorical distribution over a fixed value support. The scalar Q-value can then be recovered as the expectation of this distribution, while the critic is trained using a cross-entropy loss between the predicted and target value distributions.

Let $\mathcal{Z} = \{z_i\}_{i=1}^N$ be a fixed discrete value support with N atoms, where $z_i = v_{\min} + \frac{i-1}{N-1}(v_{\max} - v_{\min})$. A categorical critic predicts a distribution over this support rather than a scalar value. Specifically, for a state-action pair (s, a), it outputs logits $f_\phi(s, a) \in \mathbb{R}^N$ and probabilities $p_\phi(s, a) = \mathrm{softmax}(f_\phi(s, a))$. The scalar value estimate is obtained as the expectation $Q_\phi(s, a) = \sum_{i=1}^N z_i\, p_{\phi,i}(s, a)$. The target distribution is constructed by applying the Bellman backup to the categorical distribution predicted by the target critic and projecting the shifted distribution back onto $\mathcal{Z}$ using the C51 projection [9]. Let $\hat{p}(s, a)$ denote the projected target distribution. The critic is trained with the cross-entropy loss $\mathcal{L}_Q(\phi) = -\mathbb{E}_{(s,a)\sim\mathcal{D}} \sum_{i=1}^N \hat{p}_i(s, a) \log p_{\phi,i}(s, a)$.

3.3 SimbaV2

SimbaV2 [8] stabilizes large off-policy critics by combining hyperspherical feature normalization, post-update weight projection, LERP residual blocks, and categorical value learning. Given a normalized observation-action input $\bar{x}_t$, SimbaV2 appends a constant coordinate and embeds

$$\tilde{x}_t = \ell_2\text{-Norm}([\bar{x}_t; c_{\text{shift}}]), \qquad h_t^0 = \ell_2\text{-Norm}(s_h^0 \odot W_h^0 \tilde{x}_t). \tag{7}$$

Each residual block applies an inverted bottleneck MLP followed by a learned linear interpolation:

$$\tilde{h}_t^\ell = \ell_2\text{-Norm}\big(W_{h,2}^\ell \left[\mathrm{ReLU}\big(s_h^\ell \odot W_{h,1}^\ell h_t^\ell\big)\right]\big), \qquad h_t^{\ell+1} = \ell_2\text{-Norm}\Big((\mathbf{1} - \beta^\ell) h_t^\ell + \beta^\ell \odot \tilde{h}_t^\ell\Big). \tag{8}$$

Here $\mathbf{1} \in \mathbb{R}^{d_h}$ denotes the all-ones vector, $\beta^\ell \in \mathbb{R}^{d_h}$ is a learnable interpolation vector, $W_{h,1}^\ell \in \mathbb{R}^{4d_h \times d_h}$, $W_{h,2}^\ell \in \mathbb{R}^{d_h \times 4d_h}$, and $s_h^\ell \in \mathbb{R}^{4d_h}$ is a learnable scaling vector. The critic head maps h_t^L to categorical logits. Each update is followed by a hyperspherical projection of the weight vectors:

$$W \leftarrow \ell_2\text{-Norm}(W - \eta \nabla_W \mathcal{L}). \tag{9}$$

3.4 BRC

BRC [15] combines a high-capacity BroNet [12] architecture with categorical cross-entropy value learning. Unlike clipped double Q-learning, which uses the minimum of two critics to form the target, BRC maintains two critics and uses their average as the critic estimate. Its critic backbone is a residual MLP with layer normalization LN and ReLU activations:

$$h^0 = \mathrm{ReLU}(\mathrm{LN}(W_{\text{in}} x)), \qquad h^{\ell+1} = h^\ell + \mathrm{LN}\big(W_2^\ell\, \mathrm{ReLU}(\mathrm{LN}(W_1^\ell h^\ell))\big). \tag{10}$$

The final linear layer maps the last hidden representation to categorical logits over the value support.

4 Motivating Example: Low-Rank Updates under Off-Policy Bootstrapping

To illustrate when low-rank critic updates can help in off-policy learning, we construct a minimal example that captures three key ingredients of practical critic training: bootstrapped targets, replay-induced distribution shift, and overparameterized value approximation. The example contrasts low-rank adaptation with full-parameter critic learning, showing how restricting the update space can regularize bootstrapped value learning under nonstationary data. Full details are provided in Appendix A. Figure 1 shows a clear contrast between two training regimes. Under *static regression*, where the critic is trained directly on the true target values, the dense parameterization achieves the lowest final error, indicating that low-rank updates mainly act as a capacity bottleneck in this supervised setting. Under *bootstrapped off-policy TD*, however, the trend reverses: small-rank LoRA attains lower final true-Q error than the dense baseline, with the best result achieved at $r = 1$. Small-rank LoRA also settles to a lower late-stage Bellman-residual plateau. These results suggest

that the benefit of low-rank updates in off-policy RL is not reduced model capacity; rather, under bootstrapping and distribution shift, the low-rank constraint acts as an optimization regularizer that can improve critic learning.

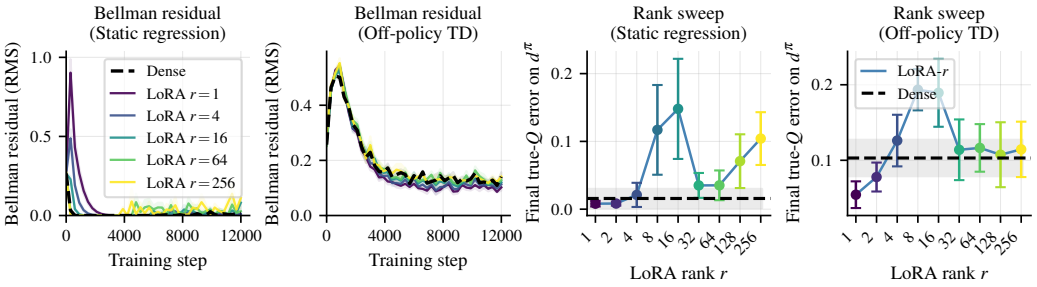

Figure 1: A motivating example: **Left:** Bellman residual (RMS) over training. **Right:** Final true-Q error on d^π vs. LoRA rank with 5 seeds; dashed line is the dense baseline.

5 Low-Rank Adaptation for Critic Learning

In this work, we propose to use Low-Rank Adaptation (LoRA) [14] as a structural constraint for critic learning. Figure 2 illustrates how the proposed framework can be integrated with different critic backbones, including SimbaV2 and BRC. Across architectures, the dense forward backbone is preserved, while learning is restricted to low-rank residual updates. This makes the approach architecture-agnostic and easy to adapt to existing off-policy critic designs.

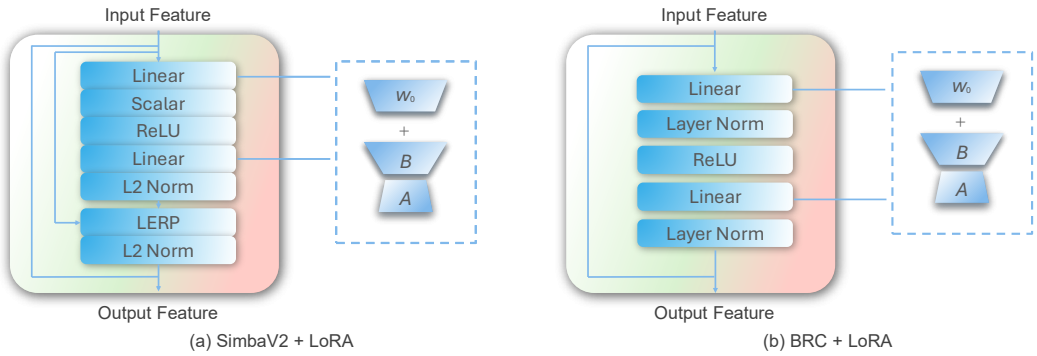

Figure 2: **Overview of the proposed LoRA-based critic architecture.** For each linear layer in the critic residual block, we freeze the randomly initialized dense base matrix W_0 and train only the low-rank adapters A and B. Panel (a) represents the LoRA method applied on the SimbaV2 architecture. Panel (b) illustrates the LoRA method for the BRC with BroNet structure.

5.1 LoRA Parameterization for Critic Networks

Consider a generic linear map in a critic residual block,

$$y = Wx, \qquad W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}.$$

In the dense baseline, W is directly optimized. In our LoRA critic, we instead use

$$W_{\text{eff}} = W_0 + \Delta W, \qquad \Delta W = \frac{\alpha_{\text{lora}}}{r} BA, \tag{11}$$

where W_0 is frozen after initialization, $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable adapter factors, r is the LoRA rank, and α_{lora} is LoRA scaling factor. Throughout all reported experiments, we set $\alpha_{\text{lora}} = r$, so that $\alpha_{\text{lora}}/r = 1$. This avoids introducing an additional scaling hyperparameter while retaining the standard LoRA parameterization.

The critic forward pass uses

$$y = W_{\text{eff}} x = \left(W_0 + \frac{\alpha_{\text{lora}}}{r} BA \right) x, \tag{12}$$

and gradients are applied only to A and B for LoRA-modified matrices.

We apply this replacement only to critic residual-block linear maps. For SimbaV2 critics, these are the inverted-bottleneck MLP weights $\{W^{\ell}_{h,1}, W^{\ell}_{h,2}\}_{\ell=0}^{L-1}$ in Eq. (8). For BRC, these are the corresponding residual-block linear maps. We keep the actor, input embedding layer, output value head, temperature parameters, and normalization statistics unchanged from the original baseline implementations. This design applies LoRA only to the dominant scaled critic parameters, while avoiding task-specific rank choices for environment-dependent input and output layers.

LoRA with Weight hyperspherical Normalization. LoRA can be directly applied to BRC, since its critic backbone does not impose additional constraints on the weight matrices. The SAC+SimbaV2 implementation, however, applies a post-update hyperspherical projection that constrains each weight vector to have unit norm. This creates a compatibility issue for LoRA: directly normalizing the effective weight $W_0 + \Delta W$ would also rescale the frozen base matrix W_0, violating the intended LoRA parameterization. To address this, we design a LoRA-compatible projection, described in Appendix B, that enforces the unit-norm constraint on the effective weight while keeping the frozen base matrix unchanged.

For each weight vector, instead of normalizing $w_0 + \Delta w$ directly, our projection keeps w_0 fixed and solves for a scalar s such that

$$\|w_0 + s\Delta w\|_2 = 1. \tag{13}$$

We initialize each frozen base vector with squared Euclidean norm $\kappa = \|w_0\|_2^2 = 0.5$, so $\|w_0\|_2 = \sqrt{0.5} < 1$. After each update, we rescale the LoRA residual while keeping the base fixed to enforce the unit-norm constraint in Eq. (13). The scalar s is absorbed into the LoRA update.

This projection also affects the LoRA initialization. Standard LoRA initializes one adapter factor to zero, yielding $\Delta W = 0$ so that the model initially matches the pretrained backbone. In our setting, however, the SimbaV2-compatible projection requires a nonzero update direction to solve for the scalar s in Eq. (13). If $\Delta w = 0$, this direction is undefined and the effective weight norm cannot be adjusted while keeping w_0 fixed.

For the SimbaV2-compatible projection described above, we therefore initialize both adapter factors with small random values:

$$A_{ij} \sim \mathcal{N}\left(0, \frac{1}{r}\right), \qquad B_{jk} \sim \mathcal{N}\left(0, \frac{1}{d_{\text{out}}}\right), \tag{14}$$

so that $\Delta W = (\alpha_{\text{lora}}/r)BA$ is nonzero at initialization and provides a valid projection direction. The motivating toy experiment instead uses standard LoRA initialization with $B_i = 0$, as detailed in Appendix A.

6 Experiments

In this section, we demonstrate that LoRA-regularized critics reduce overfitting in off-policy reinforcement learning and improve performance across diverse architectures and environments. We evaluate our method in three settings: (i) SAC+SimbaV2 [8] on seven DeepMind Control (DMC)-Hard locomotion tasks [28]; (ii) SAC+BRC [15] with BroNet [12] on the same seven DMC-Hard tasks; and (iii) FastTD3+SimbaV2* [16] on six IsaacLab robotics tasks [29]. Here, * indicates that the official FastTD3 implementation [16] does not use hyperspherical weight normalization. Detailed experimental settings are provided in Appendix D, and complete results are reported in Appendix E.

Baselines. In each setting, we compare our method against the corresponding dense reference implementation and a static sparse baseline. The dense references include **SimbaV2** [8], a state-of-the-art architecture that has been widely adopted in recent reinforcement learning algorithms, and **BRC** [15], a recently proposed Bigger, Regularized, Categorical neural network architecture. The **Sparse** baselines apply one-shot random pruning [13], with the sparsity level chosen to approximately match the trainable critic-parameter budget of LoRA. This baseline reflects prior evidence that

sparsity can unlock improved performance in large critic networks. In contrast, our approach freezes the dense base matrices in the critic residual blocks and trains only the low-rank adapters.

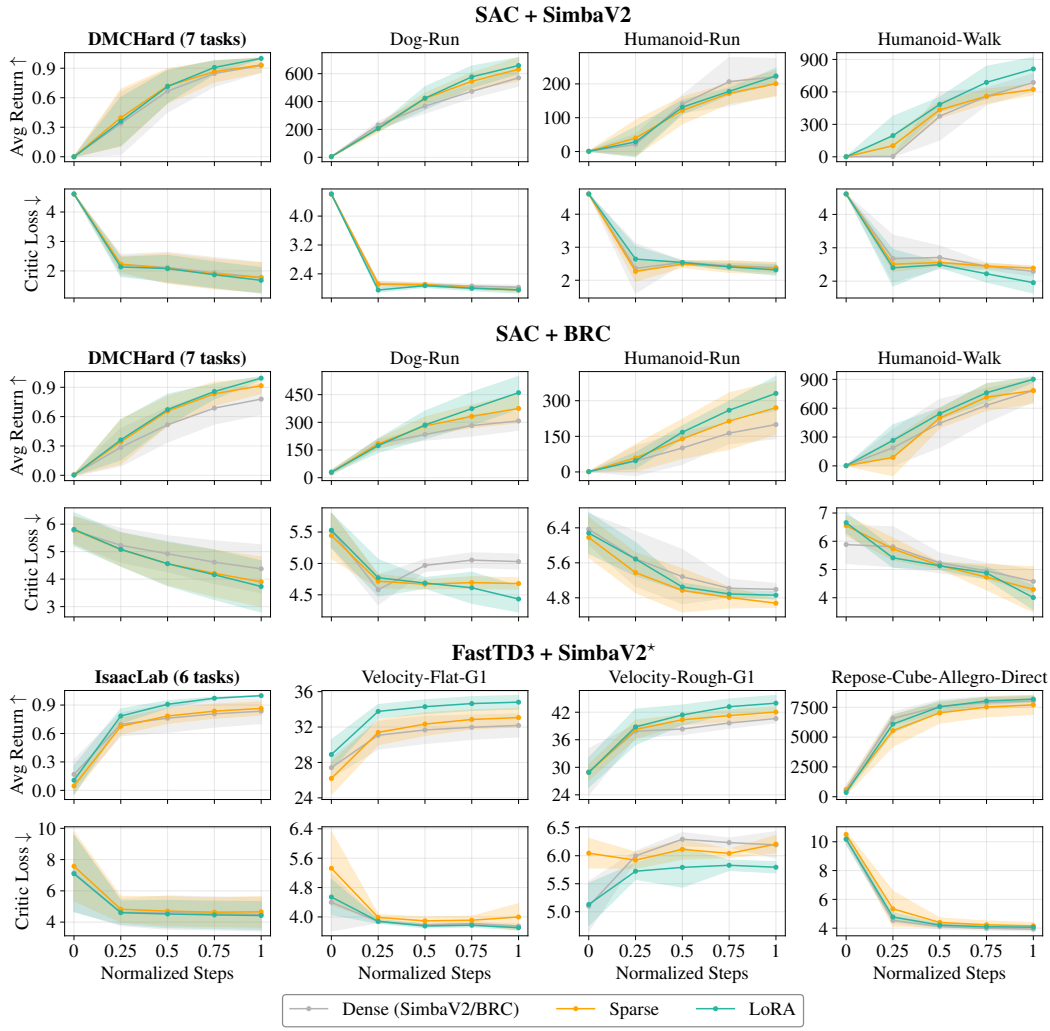

Figure 3: **Performance comparison across RL algorithms and network architectures.** We evaluate three settings: SAC+SimbaV2 on DMC-Hard tasks, SAC+BRC on DMC-Hard tasks, and FastTD3+SimbaV2* on IsaacLab robotics tasks. For each setting, the leftmost column reports the average normalized return and critic loss across all tasks, while the remaining columns show representative individual tasks. LoRA consistently achieves higher returns than the dense and sparse baselines across most settings and tasks, while maintaining comparable or lower critic loss during training. Results are averaged over 5 seeds, with shaded regions indicating standard deviation. The full per-environment learning curves can be found in Fig 8.

Main results. Figure 3 presents the main performance comparison across algorithms, architectures, and environments. For each setting, we report both policy performance measured by average return, and the critic loss. For a fair comparison, we choose the LoRA rank and sparse ratio so that the sparse baselines have a comparable number of trainable critic parameters to our method, and the detailed parameter count can be found in Table 7. In the SimbaV2 experiments, LoRA uses rank r=96, while the sparse baseline uses a sparsity ratio of 0.85. In the BRC experiments, LoRA uses rank r=128, while the sparse baseline uses a sparsity ratio of 0.9. Complete numerical results are provided in Appendix E, with detailed performance summaries in Tables 8, 9, and 10. Across all settings, LoRA achieves the strongest overall performance. Static sparsity already improves over the full-parameter SimbaV2 and BRC baselines, often accelerating early learning and yielding modest

gains in final return. This trend is also reflected in the critic-loss curves, where Sparse typically reduces the loss faster than the full-parameter baseline early in training and then maintains a similar or slightly lower loss level. LoRA provides a clear additional gain beyond static sparsity. Across SAC and FastTD3, LoRA achieves the best return on nearly all aggregate and representative tasks, with particularly clear improvements on IsaacLab and challenging locomotion domains. LoRA also generally yields a stronger critic-loss profile than SimbaV2, BRC, and their Sparse variants, often reaching the lowest loss in later training. We attribute these gains to the regularizing effect of low-rank critic adaptation: under the non-stationary and distribution-shifted data encountered in reinforcement learning, unconstrained critics can overfit to replay samples and learn brittle value estimates. In contrast, LoRA restricts the critic update space, improving fitting stability and enabling more robust value learning.

Held-out TD RMSE and critic training loss. Training critic loss alone provides only indirect evidence of reduced overfitting. Unlike conventional supervised learning on a fixed dataset, off-policy critic learning is inherently nonstationary: the replay-buffer distribution, policy, and bootstrapped TD targets evolve throughout training. Consequently, differences in critic loss may also reflect differences in how critics track changing Bellman targets under distribution shift.

To provide more direct evidence, we additionally evaluate the TD RMSE on a held-out set of replay transitions excluded from all gradient updates. This metric measures out-of-sample Bellman consistency rather than performance on the minibatches used for optimization. Figure 4 reports these diagnostics, and Table 11 in Appendix G reports the corresponding final returns. On Dog Run, LoRA achieves the lowest held-out TD RMSE and training critic loss, together with the highest mean return. On Velocity-Rough-G1, LoRA achieves the highest mean return and a lower training critic loss than the dense baseline, although it does not achieve the lowest held-out TD RMSE. The relationship between policy performance and either critic loss or held-out TD error is therefore not strictly monotonic across methods and tasks, suggesting that the interplay among critic generalization, regularization, and optimization dynamics may also influence performance.

LoRA from Scratch vs. Dense Warm-Up. Figure 5 compares pure LoRA training with hybrid schemes that first train the critic densely for an initial fraction of training and then switch to LoRA. Across all four tasks, pure LoRA remains consistently competitive and often achieves the strongest final return. Dense warm-up can occasionally reduce the early performance gap or provide small transient gains, but these benefits are not consistent and do not reliably improve final performance. In particular, even when one-quarter or one-half of training uses unconstrained dense updates, the resulting hybrid methods do not systematically outperform pure LoRA. This result suggests that LoRA is not merely useful as a lightweight fine-tuning stage after dense critic training. Instead, its benefit appears to come from constraining critic updates throughout learning. Applying the low-rank constraint from the start shapes the optimization trajectory by limiting the critic's tendency to fit noisy bootstrapped targets, over-adapt to transient replay-buffer artifacts, or drift under non-stationary data. Once the critic has already undergone a long dense-training phase, these effects may be difficult for a later LoRA phase to correct. Overall, these results support LoRA as a training-time structural regularizer.

7 Ablation Study

In this section, we conduct a series of ablation studies using the SimbaV2 backbone at two model scales: a smaller critic with $L = 2$ and $d_h = 512$, and a larger critic with $L = 4$ and $d_h = 1024$. These studies are designed to isolate the key design choices of our method, including the frozen base matrix, hyperspherical weight normalization, and low-rank parameterization.

Effect of the frozen base-matrix rank. Following [8], we initialize the frozen base matrix as an orthogonal matrix, which is full rank by default. To study the role of this fixed random basis, we vary the rank of the frozen base matrix while keeping the LoRA update rank unchanged. Figure 6(a) shows that LoRA is generally robust to this choice, especially with the larger critic. The smaller critic is more sensitive to the base-matrix rank, particularly on `Dog-Trot`, suggesting that the fixed base representation becomes more important when critic capacity is limited. In contrast, for deeper critics with residual connections, performance is much less affected by the base-matrix rank, likely because

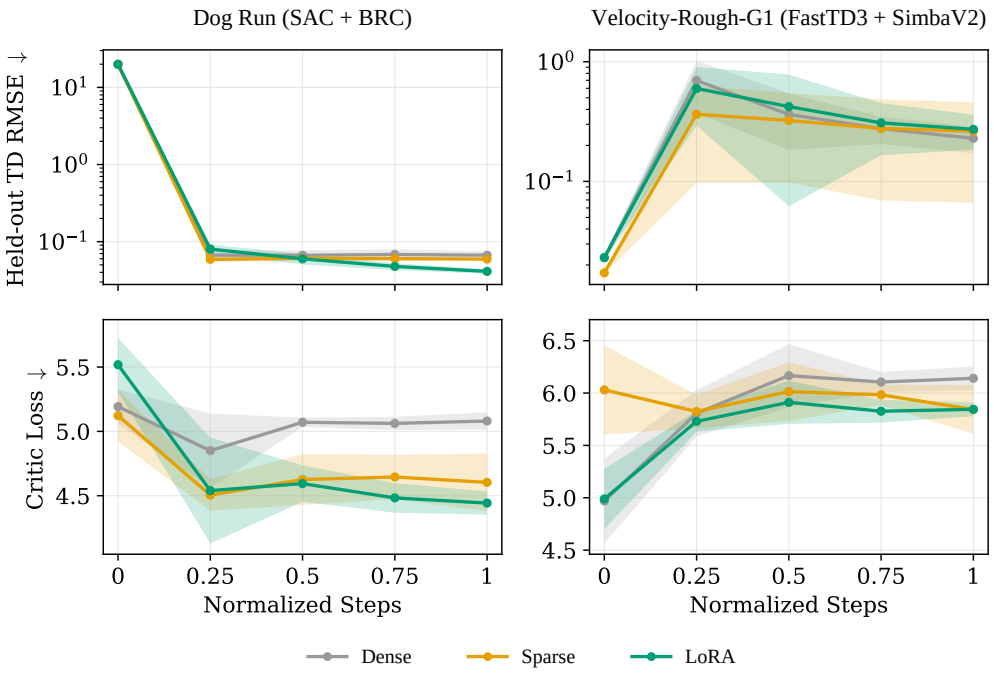

Figure 4: **Held-out TD RMSE and critic training loss.** Comparison of Dense, Sparse, and LoRA on Dog Run with SAC+BRC (left) and Velocity-Rough-G1 with FastTD3+SimbaV2 (right). The top row shows TD RMSE on held-out replay transitions excluded from gradient updates, with a logarithmic vertical scale. The bottom row shows critic training loss. Curves report the mean over five random seeds (5–9), and shaded regions indicate ± one standard deviation. The horizontal axis shows normalized training steps.

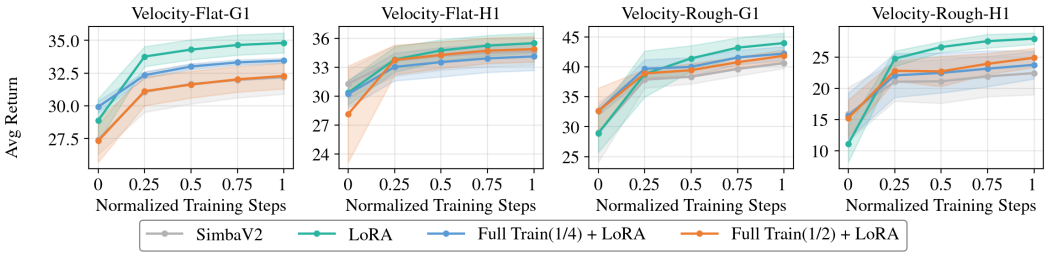

Figure 5: Comparison of pure LoRA training and hybrid schemes that perform dense updates for a fraction of training before switching to LoRA. Pure LoRA achieves competitive or superior performance, while additional dense pretraining provides only modest gains. Results are averaged over 5 seeds, with shaded regions indicating standard deviation.

the architecture already provides sufficient representational capacity. Overall, these results indicate that our method does not rely on a narrowly tuned base-matrix rank.

Effect of the base-matrix norm. Figure 6(b) examines how the initialization scale of the frozen base-matrix norm $\|w_0\|_2$ affects LoRA training. Our default setting is $\kappa = \|w_0\|_2^2 = 0.5$, corresponding to $\|w_0\|_2 = \sqrt{0.5}$. Overall, the larger critic achieves higher returns than the smaller critic and is less sensitive to this choice. The strongest sensitivity appears on `Humanoid-Run`: the larger critic performs best with intermediate norms, whereas the smaller critic favors smaller norms and degrades as $\|w_0\|_2$ increases. These results suggest that the initialization scale of the frozen backbone interacts with both task difficulty and critic capacity. In practice, intermediate sub-unit norms provide sufficient headroom for the LoRA residual and tend to yield stronger RL performance.

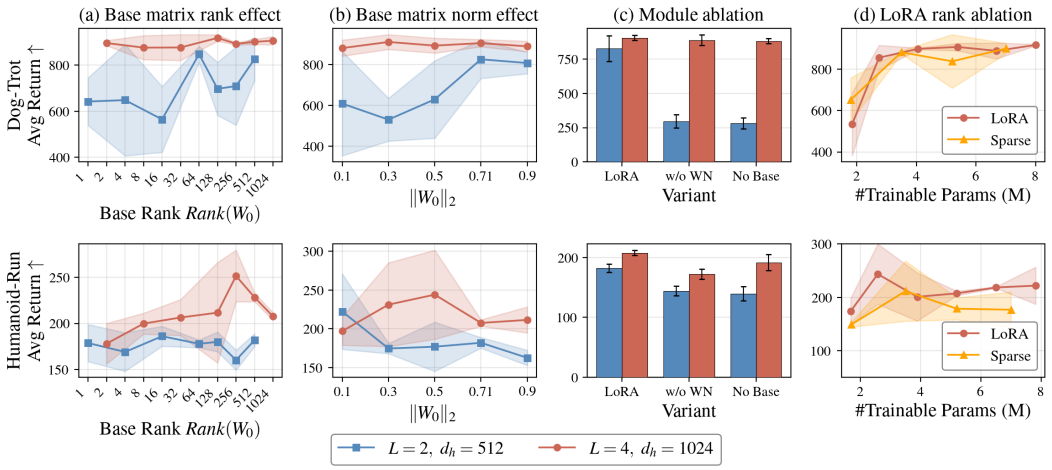

Figure 6: **Ablation study of the LoRA critic design.** We evaluate four key design choices on `Dog-Trot` (top) and `Humanoid-Run` (bottom): **(a)** the rank of the frozen base matrix W_0, **(b)** the initialization norm $\|w_0\|_2$, **(c)** the effect of removing hyperspherical weight normalization or the frozen base matrix, and **(d)** the trainable-parameter trade-off obtained by varying the LoRA rank and comparing against static sparsity. Results are shown for two critic scales

Effect of module design. Figure 6 (c) compares our proposed normalized LoRA against two ablations: *LoRA-NoWN*, which removes the weight normalization mechanism introduced in (19), and *NoBase*, which parameterizes the critic as $W = BA$ without a frozen backbone W_0. Introducing weight normalization consistently improves RL performance and robustness across tasks and model sizes; the effect is most pronounced for the smaller critic, where removing either component leads to a substantial drop in return. For the larger critic, the three variants perform more similarly, and normalized LoRA remains among the strongest variants, although the gap becomes smaller.

Parameter efficiency at matched budget. Figure 6 (d) fixes the critic architecture at L=4, d_h=1024 and varies the LoRA rank $r \in \{16, 32, 64, 96, 128, 160\}$, plotting final return against the number of trainable parameters. As a controlled baseline, we compare against Sparse [13], which achieves a similar parameter budget by pruning a dense critic rather than restricting updates to a low-rank subspace. We observe that LoRA remains broadly robust across ranks in both settings and outperforms Sparse on both tasks over most of the parameter range we evaluate. These results suggest that, under a fixed capacity budget, structured low-rank updates provide a more effective parameterization than unstructured sparsity.

8 Conclusion

In this work, we study Low-Rank Adaptation (LoRA) as a structural regularizer for critic learning in off-policy reinforcement learning. By freezing randomly initialized base matrices and optimizing only low-rank adapters, our method constrains critic updates to a low-dimensional subspace, helping mitigate overfitting in replay-based bootstrapped training. Empirically, across SAC and FastTD3 with SimbaV2 critics, as well as SAC with BRC critics, our method consistently reduces critic loss and improves policy performance over strong full-parameter and sparse baselines. These results suggest that controlling critic overfitting is itself an effective principle for improving off-policy RL, and opens a broader direction for designing architectures and training schemes that explicitly regularize value learning under non-stationary replay data.

Acknowledgments and Disclosure of Funding

We gratefully acknowledge the generous support of a Qualcomm gift fund for this project.

Jonathan Petit is affiliated with Qualcomm, which provided a gift fund supporting this work. The authors declare no other competing interests.

References

[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International conference on machine learning*, 2018.

[2] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*, 2018.

[3] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024.

[4] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[5] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[6] Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. *The Twelfth International Conference on Learning Representations*, 2024.

[7] 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. *The Thirteenth International Conference on Learning Representations*, 2025.

[8] Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. *International Conference on Machine Learning*, 2025.

[9] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. Pmlr, 2017.

[10] Daniel Palenicek, Florian Vogt, Joe Watson, and Jan Peters. Scaling off-policy reinforcement learning with batch and weight normalization. *arXiv preprint arXiv:2502.07523*, 2025.

[11] Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. *The Twelfth International Conference on Learning Representations*, 2024.

[12] Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. *Advances in neural information processing systems*, 2024.

[13] Guozheng Ma, Lu Li, Zilin Wang, Li Shen, Pierre-Luc Bacon, and Dacheng Tao. Network sparsity unlocks the scaling potential of deep reinforcement learning. *International Conference on Machine Learning*, 2025.

[14] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 2022.

[15] Michal Nauman, Marek Cygan, Carmelo Sferrazza, Aviral Kumar, and Pieter Abbeel. Bigger, regularized, categorical: High-capacity value functions are efficient multi-task learners. *Advances in Neural Information Processing Systems*, 2025.

[16] Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control, 2025.

[17] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[18] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[19] Yuexin Bian, Jie Feng, Tao Wang, Yijiang Li, Sicun Gao, and Yuanyuan Shi. Rn-d: Discretized categorical actors with regularized networks for on-policy reinforcement learning. *arXiv preprint arXiv:2601.23075*, 2026.

[20] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *International Conference on Learning Representations*, 2021.

[21] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. *International Conference on Machine Learning*, 2023.

[22] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. *International Conference on Machine Learning*, 2022.

[23] Nils Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning with spectral normalization. *Advances in neural information processing systems*, 34:8242–8255, 2021.

[24] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[26] Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 12799–12807, 2023.

[27] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning, 2022.

[28] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[29] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

A Details of the Motivating Toy Experiment

A.1 Environment

The MDP is a 15-state chain $\mathcal{S} = \{0, \ldots, 14\}$ with two actions $\mathcal{A} = \{0 \text{ (left)}, 1 \text{ (right)}\}$. Transitions are stochastic: action a succeeds with probability $p = 0.9$, moving the agent one step in the intended direction, and fails (the agent stays) with probability $1 - p = 0.1$. States are reflected at both endpoints. The reward is $R(s, a) = p \cdot \mathbf{1}[s + 1 = 14]$ for the right action from state 13, and zero otherwise; the discount factor is $\gamma = 0.97$.

The **behaviour policy** μ selects each action uniformly at random. The **target policy** π always selects *right*. The distributional shift between μ and π is the source of off-policy bias: the replay buffer reflects the random-walk stationary distribution of μ (concentrated near the centre of the chain), while the evaluation distribution d^π is concentrated near the goal state.

The true value function Q^π is computed analytically by solving the Bellman system $(I - \gamma P^\pi)V^\pi = R^\pi$ and then $Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a)V^\pi(s')$.

A.2 Feature Map

Both the dense and LoRA critics operate on a shared fixed random feature map

$$\phi(s, a) = \tanh\big(W_\phi \, e_{(s,a)}\big) \in \mathbb{R}^D, \quad D = 64, \tag{15}$$

where $e_{(s,a)} \in \mathbb{R}^{|\mathcal{S}|+|\mathcal{A}|}$ is the concatenation of one-hot encodings of s and a, and $W_\phi \in \mathbb{R}^{D\times(|\mathcal{S}|+|\mathcal{A}|)}$ is drawn once from $\mathcal{N}(0, 1/\sqrt{D})$ and frozen throughout. The feature table is precomputed for all (s, a) pairs.

A.3 Critic Architectures

The critic is a two-hidden-layer MLP with hidden width $H = 256$ and ReLU activations:

$$Q_\theta(s, a) = w_{\text{out}}^\top \, \mathrm{ReLU}\big(W_1^{\text{eff}} \, \mathrm{ReLU}\big(W_0^{\text{eff}} \, \phi(s, a)\big)\big), \tag{16}$$

with $W_0 \in \mathbb{R}^{H\times D}$, $W_1 \in \mathbb{R}^{H\times H}$, and $w_{\text{out}} \in \mathbb{R}^H$. No bias terms are used.

Dense parameterisation. $W_0^{\text{eff}} = W_0$ and $W_1^{\text{eff}} = W_1$ are both trained freely; w_{out} is also trained. Total trainable parameters: $HD + H^2 + H$.

LoRA-r parameterisation. Both weight matrices are decomposed as

$$W_i^{\text{eff}} = W_i^{\text{base}} + B_i A_i, \qquad i \in \{0, 1\}, \tag{17}$$

where W_i^{base} is frozen at its initialisation value (identical to the dense critic's starting point), and A_i, B_i are the trainable low-rank factors with $A_0 \in \mathbb{R}^{r\times D}$, $A_1 \in \mathbb{R}^{r\times H}$, $B_0, B_1 \in \mathbb{R}^{H\times r}$. Following standard LoRA initialisation, A_i is drawn from $\mathcal{N}(0, 1/\sqrt{\text{fan_in}})$ and B_i is initialised to zero, so $W_i^{\text{eff}} = W_i^{\text{base}}$ at step 0 and both critics produce identical predictions at initialisation. The output layer w_{out} is trained in both cases. The rank is swept over $r \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$.

A.4 Training Regimes

Static regression. At each step a mini-batch of (s, a) pairs is drawn uniformly from $\mathcal{S} \times \mathcal{A}$. The regression target is the precomputed true value $y = Q^\pi(s, a)$, and the loss is $\mathcal{L} = \frac{1}{2}\,\mathbb{E}\big[(Q_\theta(s, a) - y)^2\big]$. There is no bootstrapping and no distributional shift in this regime; it serves as a controlled baseline to measure the pure capacity effect of the rank constraint.

Bootstrapped off-policy TD. Transitions (s, a, r, s') are sampled from a fixed replay buffer of 500 transitions collected under μ. The TD target is

$$y = r + \gamma \, \bar{Q}(s', \, \pi(s')), \tag{18}$$

where $\bar{Q}$ is a *target network*—a slow-moving copy of Q_θ updated via Polyak averaging $\bar{\theta} \leftarrow (1 - \tau)\,\bar{\theta} + \tau\,\theta$ with $\tau = 0.02$. The action used in the bootstrap is always $\pi(s')$ (the target policy), even though the stored transition was generated under μ, which is the source of the off-policy distributional shift. The loss is $\mathcal{L} = \frac{1}{2}\,\mathbb{E}\big[(Q_\theta(s, a) - \mathrm{sg}(y))^2\big]$, where $\mathrm{sg}(\cdot)$ denotes stop-gradient.

A.5 Evaluation Metrics

All metrics are computed every 300 training steps.

- **True-Q error on d^π:** $\varepsilon_Q = \sqrt{\sum_s d^\pi(s) \left(Q_\theta(s, \pi(s)) - Q^\pi(s, \pi(s))\right)^2}$. This is the primary metric; it measures critic quality under the target policy's state distribution.
- **Bellman residual (RMS):** $\varepsilon_B = \sqrt{\frac{1}{|\mathcal{S}||\mathcal{A}|} \sum_{s,a} \left(Q_\theta(s,a) - \mathcal{T}^\pi Q_\theta(s,a)\right)^2}$, where $\mathcal{T}^\pi Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, Q(s', \pi(s'))$ is the exact Bellman operator. A low Bellman residual means the critic is self-consistent with its own Bellman equation; note that this is computed with the true transition model and is *not* the training loss.

A.6 Hyperparameters

The hyperparameters are listed in Table 1.

Table 1: Hyperparameters for the toy motivating experiment.

Hyperparameter	Value
MDP	
Number of states $\lvert\mathcal{S}\rvert$	15
Number of actions $\lvert\mathcal{A}\rvert$	2
Transition success probability p	0.9
Discount factor γ	0.97
Feature map & critic	
Feature dimension D	64
Hidden width H	256
Activation	ReLU
Replay & policies	
Replay buffer size	500
Behaviour policy μ	Uniform random
Target policy π	Always right
Optimisation	
Optimiser	Adam
Learning rate	10^{-3}
Batch size	64
Training steps	12 000
Target network Polyak rate τ	0.02
Evaluation interval	300 steps
LoRA	
Rank sweep r	$\{1, 2, 4, 8, 16, 32, 64, 128, 256\}$
A_i initialisation	$\mathcal{N}(0, 1/\sqrt{\text{fan_in}})$
B_i initialisation	0
Random seeds	5

B Hyperspherical Weight Normalization with LoRA

This section applies only to the SAC+SimbaV2 experiments. The SAC with SimbaV2 reference implementation performs post-update hyperspherical projection of linear weights while FastTD3 with SimbaV2 and SAC with BRC do not use this post-update weight projection. The same projection could be used with different backbones that impose row-wise hyperspherical weight constraints, but it is not used for the FastTD3 and BRC configurations evaluated here.

SimbaV2 uses hyperspherical weight normalization, projecting each row of a weight matrix to unit norm after every update. Under LoRA, maintaining this constraint is non-trivial (see Appendix C),

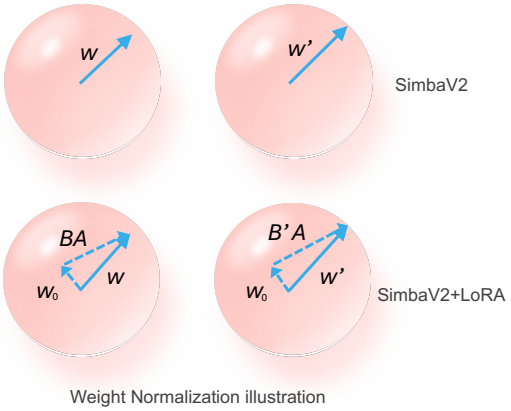

Figure 7: **LoRA-compatible hyperspherical projection.** Top: standard SimbaV2 projects each updated weight vector W onto the unit hypersphere to obtain W'. Bottom: in the LoRA parameterization, the base vector W_0 is frozen, so directly normalizing $W_0 + \Delta W$ would also rescale the frozen base. Instead, for the SAC+SimbaV2 setting, we solve for a scalar $s_j > 0$ and absorb it into the LoRA update so that $W_0 + s_j \Delta W$ lies on the unit hypersphere while W_0 remains fixed.

because the base matrix W_0 is frozen and cannot be directly renormalized. To address this, we propose an effective-weight projection rule for LoRA. After each update to (B, A), we form the residual ΔW in Eq. (11) and solve Eq. (19) for a per-output scaling factor s_j. This keeps the effective row on the unit hypersphere while leaving the frozen base row unchanged:

$$\left\| [W_0]_{j,:} + s_j [\Delta W]_{j,:} \right\|_2 \;=\; 1. \tag{19}$$

Let $w_j = [W_0]_{j,:}$ and $\delta_j = [\Delta W]_{j,:}$ denote the j-th rows of W_0 and ΔW, respectively. Squaring Eq. (19) yields a quadratic equation in s_j:

$$(\|\delta_j\|_2^2) s_j^2 + 2\langle w_j, \delta_j \rangle s_j + (\|w_j\|_2^2 - 1) \;=\; 0. \tag{20}$$

When $\|\delta_j\|_2 > 0$, the solutions are

$$s_j = \frac{-\langle w_j, \delta_j \rangle \;\pm\; \sqrt{\langle w_j, \delta_j \rangle^2 - \|\delta_j\|_2^2 (\|w_j\|_2^2 - 1)}}{\|\delta_j\|_2^2}, \tag{21}$$

In practice, we clamp the denominator by a small ε only for numerical stability. In our implementation, we select the positive root $s_j > 0$ so that the LoRA update preserves its direction throughout training.

The closed-form solution in Eq. (21) depends explicitly on both the norm of the base matrix row, $\|w_j\|_2$, and the squared norm of the LoRA update, $\|\delta_j\|_2^2$. This observation motivates the initialization scheme for the frozen base matrix and the LoRA adapters described below.

C Why standard row-wise weight normalization is incompatible with frozen LoRA bases

A natural approach to combining LoRA with hyperspherical weight normalization is to apply the standard row-wise normalization directly to the effective weight,

$$W_{\text{eff}} \leftarrow \frac{W_0 + \Delta W}{\|W_0 + \Delta W\|_2 + \varepsilon}, \tag{22}$$

where the normalization is applied independently to each row. Although this projects the effective weight onto the unit hypersphere in one step, it is incompatible with the frozen-base parameterization used by LoRA. To see this, consider a single row and let

$$w_j = [W_0]_{j,:}, \qquad \delta_j = [\Delta W]_{j,:}.$$

Then standard row-wise normalization gives

$$w_j + \delta_j \;\mapsto\; \frac{w_j + \delta_j}{\|w_j + \delta_j\|_2 + \varepsilon} \;=\; c_j w_j + c_j \delta_j, \tag{23}$$

where

$$c_j = \frac{1}{\|w_j + \delta_j\|_2 + \varepsilon}. \tag{24}$$

Thus, both the frozen base component and the LoRA update are rescaled by the same factor c_j. In particular, the base row is implicitly changed from w_j to $c_j w_j$, which means that W_0 is no longer truly frozen whenever $c_j \neq 1$. This creates two problems. First, it violates the intended LoRA assumption that the base matrix remains fixed throughout training. Second, it silently breaks the additive low-rank decomposition

$$W_{\text{eff}} = W_0 + \frac{\alpha_{\text{lora}}}{r} BA, \tag{25}$$

because after normalization the effective row can no longer be interpreted as the sum of the original frozen base row and a pure low-rank update. Instead, the normalization couples the frozen component and the trainable component through a shared multiplicative factor.

Our normalization rule avoids this issue by solving, for each row, a scalar s_j that is absorbed only into the LoRA update. Concretely, we enforce the unit-norm constraint on

$$w_j + s_j \delta_j, \tag{26}$$

while keeping w_j literally unchanged. In implementation, this rescaling is absorbed into the LoRA factor B, so the frozen base matrix W_0 remains untouched at every optimization step and the low-rank structure is preserved throughout training.

D Training Hyperparameters

Tables 2, 3, and 4 summarize the training hyperparameters for the SAC+SimbaV2 DMC-Hard experiments, the FastTD3+SimbaV2 IsaacLab experiments, and the SAC+BRC BroNet experiments, respectively. Within each setting, the dense, sparse, and LoRA variants use the same environment, replay buffer, batch size, update schedule, optimizer family, and learning-rate schedule, following the corresponding reference implementations [8, 13, 16, 15, 12]. We note that, in our setting, BRC follows the single-task training setup described in the original BRC paper.

For our LoRA-based approach, all trainable parameters outside the LoRA modules use the same optimizer and learning-rate schedule as the corresponding SimbaV2 and BRC baselines. For the LoRA modules, we use AdamW with weight decay applied to the trainable factors A and B. In the SAC+BRC and FastTD3+SimbaV2 configurations, which do not use post-update hyperspherical weight projection, this regularization is intended to discourage the residual BA from becoming large relative to the frozen base W_0. In SAC+SimbaV2, a separate post-update projection enforces unit norm on the effective weight vectors.

Table 5 describes the seven DMControl hard tasks used in our SAC experiments. Table 6 describes the six IsaacLab tasks used in our FastTD3 experiments. These span locomotion (Unitree H1 and G1 robots on flat and rough terrain) and dexterous in-hand manipulation (Allegro and Shadow hands).

Table 7 reports the actor parameter count, critic total parameter count, and critic trainable parameter count for all three methods under different settings. Since FastTD3 uses double Q-networks whereas our SAC+SimbaV2 configuration uses a single Q-network, the critic parameter count in FastTD3 is approximately doubled even when both methods use the same SimbaV2 critic configuration, i.e., $d_h = 1024$ and $L = 4$.

E Detailed Experimental Results

Computational resources. All experiments were run on an internal Linux server with 8 NVIDIA RTX 6000 Ada Generation GPUs, each with 48GB of GPU memory. Each training run used one GPU unless otherwise stated. SAC experiments on DMC-Hard used CPU-based environment stepping and one GPU for network updates; FastTD3 experiments on IsaacLab used GPU-parallel simulation with 4,096 environments and bf16 mixed precision.

Table 2: SAC+SimbaV2 training hyperparameters (DMControl). All methods share the same base configuration from SimbaV2 [8]. The bottom section lists LoRA-specific parameters that apply only to the adapter factors.

Shared across all methods (SimbaV2, Sparse, LoRA)	
Optimizer (base params)	Adam
Learning rate (start $\rightarrow$ end)	$1 \times 10^{-4} \rightarrow 5 \times 10^{-5}$
LR schedule	Linear decay
Batch size	256
UTD (updates-to-data) ratio	2 (action repeat = 2)
Replay buffer size	1×10^{6}
Warmup steps (min buffer)	5,000
Discount γ	Heuristic from TD-MPC2 (≈ 0.99)
Target EMA ρ	0.005
Temperature init	0.01
Target entropy coeff	-0.5
Observation normalization	Running statistics
Reward normalization	Yes
Normalized $G_{\max}$	5.0
Critic value head	Categorical ($n_{\text{atom}} = 101$, support $[-5, 5]$)
Feature ℓ_2 normalization	Yes
Post-update weight hyperspherical projection	Yes
Actor architecture	1 block, $d_h = 128$
Total env steps	1×10^{6}
n-step returns	1
Double-Q	No
Sparse-specific	
Sparsity ratio ($d_h = 512$, $L = 2$ blocks)	0.7
Sparsity ratio ($d_h = 1024$, $L = 4$ blocks)	0.85
Pruning method	One-shot random (Erdős–Rényi)
LoRA-specific (adapter factors A, B only)	
Optimizer (LoRA params)	AdamW
Weight decay	2×10^{-4}
LoRA rank r	96
LoRA scaling α_{lora}	96
A initialization	Normal
B initialization	Normal
base matrix init scale κ	0.5
LoRA applied to	Critic residual blocks only
base matrix W_0	Frozen (zero gradient)

DMC-Hard results with SAC+SimbaV2. Table 8 reports the final episode return for SAC+SimbaV2 on the seven DMC-Hard tasks. We compare dense SimbaV2, Sparse-SimbaV2 with static one-shot random pruning, and LoRA-SimbaV2 with frozen dense critic weights and trainable low-rank adapters. All numbers are computed over 5 seeds (seeds 0–4), and we report mean $\pm$ one standard deviation. Bold entries indicate the best mean return among the three methods for each task and scale.

In the $d_h = 512$, $L = 2$ blocks settings, LoRA matches SimbaV2 on average (701.3 vs. 701.1) while substantially outperforming the sparse baseline (607.3).

In the $d_h = 1024$, $L = 4$ blocks settings, the benefit of low-rank regularization becomes clearer: LoRA achieves an average of 777.2, outperforming both SimbaV2 (719.7) and Sparse (731.2) by a substantial margin. This widening gap at larger scale supports our central claim that constraining the update space via LoRA provides increasingly effective regularization as critic capacity grows and overfitting becomes more severe.

DMC-Hard results with SAC+BRC. Table 9 reports DMC-Hard evaluation using SAC+BRC with BroNet. This setting is included as an architecture general validation: unlike SimbaV2, BroNet

Table 3: FastTD3 training hyperparameters (IsaacLab). All methods share the same base configuration from FastTD3 [16] with the SimbaV2 backbone. Task-specific defaults (e.g., n-step, $v_{\min}/v_{\max}$, total steps) follow the FastTD3 reference implementation.

Shared across all methods (SimbaV2, Sparse, LoRA)	
Optimizer (base params)	AdamW
Weight decay (base params)	0.0
Learning rate (start $\rightarrow$ end)	$3 \times 10^{-4} \rightarrow 3 \times 10^{-5}$ (cosine)
Batch size	8,192
Number of parallel envs	4,096
Update interval	4 env steps
Gradient updates per interval	16 (locomotion) / 8 (manipulation)
Replay buffer size	3,072 transitions per env
n-step returns	8 (locomotion) / 1 (manipulation)
Discount γ	0.99
Target EMA τ	0.1
Exploration noise σ	$[0.001, 0.4]$ (linear anneal)
Action bounds	$[-1, 1]$
Observation normalization	Running statistics
Critic value head	Categorical ($n_{\text{atom}} = 251$, see below)
Feature ℓ_2 normalization	Yes
Post-update weight hyperspherical projection	No
Actor architecture	1 block, $d_h = 256$
Critic architecture	4 blocks, $d_h = 1024$
Mixed-precision (AMP)	Yes (bf16)
Double-Q	Yes
Task-specific value support	
Locomotion tasks	$v_{\min} = -10$, $v_{\max} = 10$
Manipulation tasks	$v_{\min} = -500$, $v_{\max} = 500$
Sparse-specific	
Sparsity ratio ($d_h = 1024$, $L = 4$ blocks)	0.85
Pruning method	One-shot random (Erdős–Rényi)
LoRA-specific (adapter factors A, B only)	
Optimizer (LoRA params)	AdamW
Weight decay	6×10^{-4}
LoRA rank r	96
LoRA scaling α_{lora}	96
A initialization	Normal
B initialization	Normal
base matrix init scale κ	0.5
LoRA applied to	Critic residual blocks only
base matrix W_0	Frozen (zero gradient)

uses LayerNorm-based residual blocks and does not rely on hyperspherical row projection. Thus, improvements in this setting indicate that LoRA is not merely correcting a SimbaV2-specific parameterization.

LoRA-BRC achieves the best average return, improving from 625.1 for dense BroNet-BRC and 713.4 for Sparse-BRC to 755.1. It obtains the highest mean score on six of the seven tasks, with especially large gains on `dog-run`, `humanoid-run`, and `humanoid-walk`. Together with the SimbaV2 results, this suggests that low-rank critic adaptation acts as a general update-space regularizer rather than a mechanism tied to one specific model design.

IsaacLab results with FastTD3+SimbaV2. Table 10 reports the final episode return for FastTD3+SimbaV2 on six IsaacLab tasks. We compare dense SimbaV2, Sparse-SimbaV2, and LoRA-SimbaV2 using the same FastTD3 training setup and critic architecture. All numbers are computed over 5 seeds (seeds 0–4), and we report mean $\pm$ one standard deviation. Bold entries indicate the best mean return among the three methods for each task.

Table 4: SAC+BRC hyperparameters for DMC-Hard with BroNet. These runs follow single-task BRC [15] settings with the BroNet [12] backbone

Shared across BRC, Sparse-BRC, and LoRA-BRC	
Algorithm	SAC+BRC
Critic target aggregation	Ensemble average, no clipped double Q
Critic ensemble size	2
Optimizer	AdamW
Actor / critic learning rate	3×10^{-4}
Weight decay	1×10^{-4}
Temperature optimizer	Adam, learning rate 3×10^{-4}, $\beta_1 = 0.5$
Initial temperature	0.1
Target entropy	$-\lvert\mathcal{A}\rvert/2$
Discount γ	0.99
Target EMA τ	0.005
Batch size	256
UTD ratio	2
Replay buffer size	10^6
Random action warmup	5,000 steps
Total environment steps	10^6
Evaluation episodes	10
Categorical critic	101 atoms, support $[-10, 10]$
Reward normalization	BRC return normalization
Post-update weight hyperspherical projection	No
Actor architecture	1 block, $d_h = 256$
Critic architecture	2 blocks, $d_h = 4096$
Sparse-specific	
Critic sparsity	0.90
Pruning method	One-shot random (Erdős–Rényi)
LoRA-specific	
Optimizer (LoRA params)	AdamW
LoRA rank r	128
LoRA scaling α_{lora}	128
weight decay	6×10^{-4}
A initialization	Normal
B initialization	Normal
base matrix init scale κ	0.5
LoRA applied to	Critic residual blocks only
base matrix W_0	Frozen (zero gradient)

LoRA-SimbaV2 achieves the best average return on both IsaacLab locomotion and in-hand manipulation. On locomotion, LoRA improves the average return from 32.5 for dense SimbaV2 and 33.4 for Sparse-SimbaV2 to 35.6. On manipulation, LoRA obtains the best average return of 10135.6, while remaining competitive on both individual manipulation tasks. These results show that the benefit of low-rank critic adaptation transfers from SAC on CPU-stepped DMC-Hard tasks to FastTD3 with massively parallel GPU simulation.

Full per-environment learning curves for all three main settings are provided in Figure 8. All the curves are mean $\pm$ std across 5 seeds; x-axis is normalized training steps.

Computational cost analysis. During experiments, SimbaV2, Sparse, and LoRA exhibit comparable GPU VRAM usage and training time within the same algorithm. All methods share the same dense architecture, leading to nearly identical per-step computation. Training time is primarily dominated by environment interaction (CPU-based stepping for SAC on DMC; GPU-parallel rollouts for FastTD3 on IsaacLab) and large-batch updates of distributional critics [9], rather than the number of trainable parameters. Similarly, VRAM consumption is largely driven by replay buffers and activation storage instead of the critic network parameters. Therefore, LoRA is not motivated by parameter or memory efficiency in RL, but by the regularization effect of constraining updates to a low-dimensional subspace, as demonstrated in our experiments.

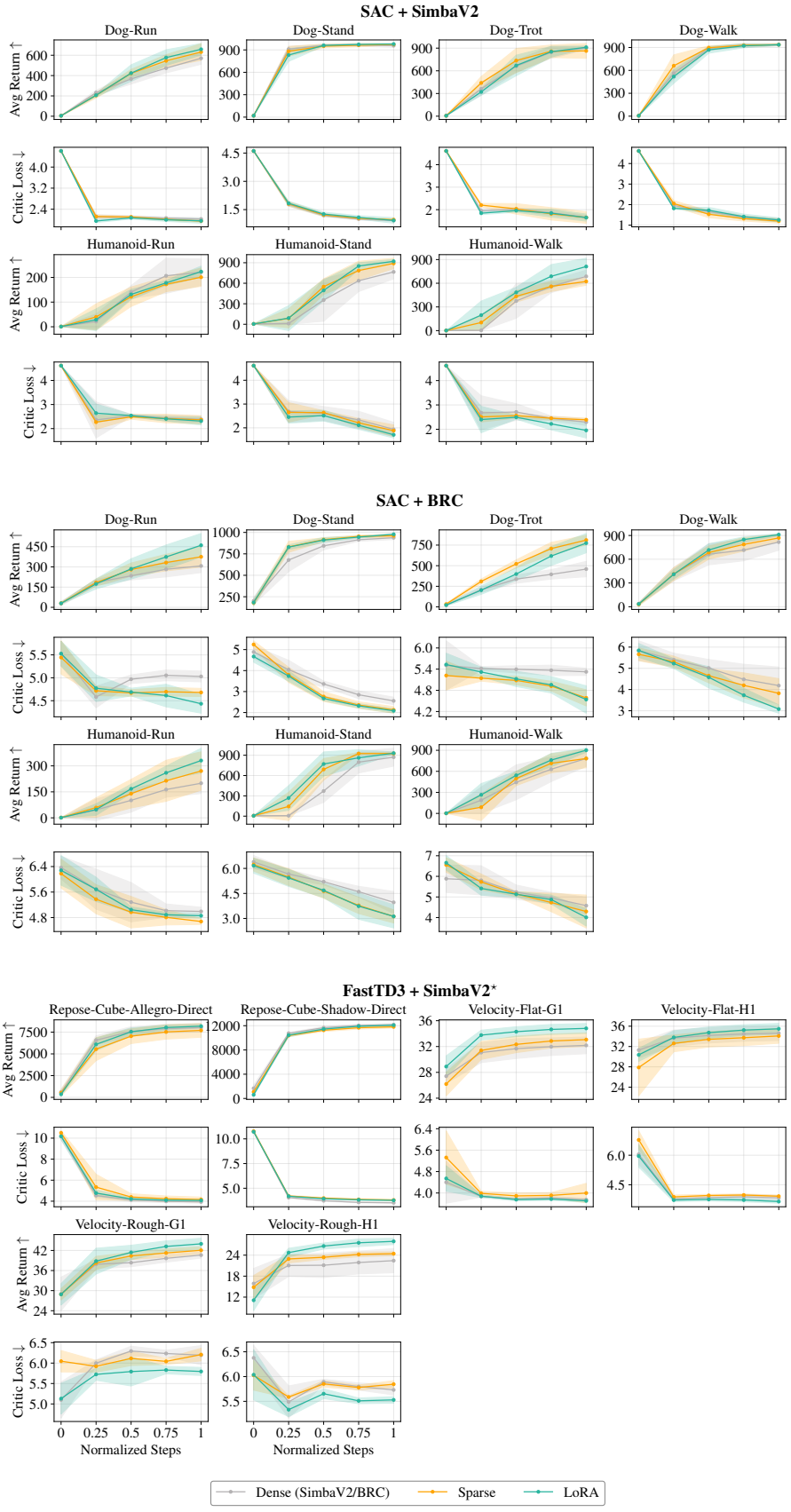

Figure 8: **Per-environment learning curves.**

Table 5: **DMC-Hard Complete List.** We evaluate a total of 7 continuous control tasks from the DMC-Hard benchmark. Below, we provide a list of all the tasks considered. The baseline performance for each task is reported at 1M environment steps.

Task	Observation dim $\lvert\mathcal{O}\rvert$	Action dim $\lvert\mathcal{A}\rvert$
`dog-run`	223	38
`dog-trot`	223	38
`dog-stand`	223	38
`dog-walk`	223	38
`humanoid-run`	67	24
`humanoid-stand`	67	24
`humanoid-walk`	67	24

Table 6: **IsaacLab Complete List.** We evaluate a total of 6 continuous control tasks from IsaacLab. Observation and action dimensions are determined by the robot morphology and task configuration. The baseline performance for each task is reported at 1M environment steps.

Task	Observation dim $\lvert\mathcal{O}\rvert$	Action dim $\lvert\mathcal{A}\rvert$
`Isaac-Velocity-Flat-H1-v0`	69	19
`Isaac-Velocity-Flat-G1-v0`	123	37
`Isaac-Velocity-Rough-H1-v0`	256	19
`Isaac-Velocity-Rough-G1-v0`	310	37
`Isaac-Repose-Cube-Allegro-Direct-v0`	124	16
`Isaac-Repose-Cube-Shadow-Direct-v0`	157	20

F Theoretical Proof

Lemma 1 (Existence of a positive scaling root under initialization)**.** *Consider the scaling equation in Eq. (21) for row j, induced by the base weight w_j and LoRA update δ_j, with $\delta_j \neq 0$. Under the initialization with $\|w_j\|_2^2 = \kappa$ with $\kappa \in (0,1)$, the corresponding quadratic equation in s_j admits two real roots $s_j^{(0)}$ and $s_j^{(1)}$ satisfying*

$$s_j^{(0)} > 0, \qquad s_j^{(1)} < 0.$$

In particular, there always exists a positive scaling root.

Proof. For each row j, the scaling coefficient s_j is obtained by solving the quadratic equation associated with Eq. (21), which can be written in the form

$$\|\delta_j\|_2^2 \, s_j^2 + 2\langle w_j, \delta_j \rangle s_j + (\|w_j\|_2^2 - 1) = 0. \tag{27}$$

Under our initialization, $\delta_j \neq 0$, so the quadratic coefficient satisfies

$$\|\delta_j\|_2^2 > 0.$$

Moreover, each base weight row is initialized to have norm

$$\|w_j\|_2^2 = \kappa, \qquad \kappa \in (0,1),$$

and therefore the constant term is

$$\|w_j\|_2^2 - 1 = \kappa - 1 < 0.$$

Since the product of the two roots of Eq. (27) is

$$s_j^{(0)} s_j^{(1)} = \frac{\|w_j\|_2^2 - 1}{\|\delta_j\|_2^2} = \frac{\kappa - 1}{\|\delta_j\|_2^2} < 0,$$

the two roots must have opposite signs. Hence one root is positive and the other is negative. Denoting the positive root by $s_j^{(0)}$ and the negative root by $s_j^{(1)}$, we obtain

$$s_j^{(0)} > 0, \qquad s_j^{(1)} < 0.$$

This proves that a positive scaling root always exists. □

Table 7: Parameter counts for dense, sparse, and LoRA across the SimbaV2 and BroNet+BRC settings. Actor parameters are unchanged across methods within each setting. Since tasks within each setting use the same SimbaV2 or BroNet hidden dimension and number of residual blocks, the parameter counts are approximately the same across environments, up to small differences caused by observation and action dimensions.

Algorithm / Env.	Method	Critic Config.	Actor Params	Critic Total Params	Critic Trainable Params
SAC+SimbaV2 / Humanoid	SimbaV2	$d_h = 512, L = 2$	0.18M	4.56M	4.56M
	Sparse-SimbaV2	$d_h = 512, L = 2, s = 0.70$	0.18M	4.56M	1.37M
	LoRA	$d_h = 512, L = 2, r = 96$	0.18M	5.54M	1.35M
	SimbaV2	$d_h = 1024, L = 4$	0.18M	34.82M	34.82M
	Sparse-SimbaV2	$d_h = 1024, L = 4, s = 0.85$	0.18M	34.82M	5.22M
	LoRA	$d_h = 1024, L = 4, r = 96$	0.18M	38.75M	5.20M
FastTD3 / Isaac-Velocity-Flat-H1	SimbaV2	$d_h = 1024, L = 4$	0.61M	69.95M	69.95M
	Sparse-SimbaV2	$d_h = 1024, L = 4, s = 0.85$	0.61M	69.95M	10.53M
	LoRA	$d_h = 1024, L = 4, r = 96$	0.61M	77.81M	10.70M
SAC+BRC / Humanoid	BroNet-BRC	$d_h = 4096, L = 2$	0.16M	135.89M	135.89M
	Sparse-BroNet-BRC	$d_h = 4096, L = 2, s = 0.90$	0.16M	135.89M	13.59M
	LoRA-BroNet-BRC	$d_h = 4096, L = 2, r = 128$	0.16M	144.28M	10.06M

Table 8: Final episode return on DMControl hard locomotion tasks. Three methods are compared: SimbaV2 (full-parameter baseline), Sparse (random one-shot pruning [13]), and LoRA (rank-96 adapters with frozen base matrices). Results are mean ± std over 5 seeds.

Task	**SimbaV2**	**Sparse**	**LoRA (ours)**
	$d_h = 512$, $L = 2$ *blocks*		
dog-run	**554.9** ± 96.4	348.8 ± 110.3	504.4 ± 56.8
dog-trot	**864.3** ± 37.5	621.3 ± 233.7	837.4 ± 68.1
dog-stand	**969.8** ± 12.2	950.3 ± 22.9	955.7 ± 22.5
dog-walk	**926.7** ± 26.2	848.9 ± 46.7	908.2 ± 25.0
humanoid-run	182.4 ± 20.9	151.1 ± 9.5	**221.3** ± 54.4
humanoid-stand	814.9 ± 173.5	781.5 ± 133.1	**861.8** ± 127.5
humanoid-walk	594.3 ± 55.2	549.0 ± 41.2	**620.3** ± 16.4
Average	701.1	607.3	**701.3**
	$d_h = 1024$, $L = 4$ *blocks*		
dog-run	570.4 ± 59.1	631.8 ± 82.2	**658.6** ± 56.6
dog-trot	901.3 ± 21.4	862.8 ± 97.8	**911.1** ± 16.0
dog-stand	957.0 ± 23.9	973.1 ± 9.2	**978.7** ± 6.4
dog-walk	935.1 ± 12.3	**937.0** ± 12.4	936.1 ± 11.0
humanoid-run	221.2 ± 53.6	200.8 ± 36.9	**223.5** ± 22.2
humanoid-stand	765.7 ± 100.9	890.1 ± 75.3	**920.4** ± 28.1
humanoid-walk	687.1 ± 78.0	622.9 ± 48.9	**811.8** ± 104.6
Average	719.7	731.2	**777.2**

Table 9: Final episode return on DMC-Hard using SAC+BRC with a BroNet critic ($d_h = 4096$, $L = 2$). Results are mean ± standard deviation over 5 seeds. LoRA uses rank $r = 128$ and $\alpha_{\text{lora}} = 128$.

Task	**BroNet-BRC**	**Sparse-BRC**	**LoRA-BRC**
`dog-run`	307.3 ± 44.4	375.0 ± 68.3	**460.6 ± 78.2**
`dog-trot`	458.9 ± 83.3	**810.8 ± 52.2**	773.8 ± 104.4
`dog-stand`	938.2 ± 19.2	961.0 ± 16.2	**978.0 ± 8.3**
`dog-walk`	816.6 ± 92.1	868.9 ± 39.0	**910.5 ± 12.8**
`humanoid-run`	199.6 ± 53.1	270.4 ± 99.1	**330.8 ± 63.5**
`humanoid-stand`	871.4 ± 118.0	925.0 ± 18.3	**930.3 ± 15.2**
`humanoid-walk`	783.3 ± 100.6	782.9 ± 117.8	**901.5 ± 25.0**
Average	625.1	713.4	**755.1**

Table 10: Final episode return on IsaacLab tasks with FastTD3. Critic architecture: $d_h = 1024$, $L = 4$ blocks. Results are mean $\pm$ std over 5 seeds.

Task	**SimbaV2**	**Sparse**	**LoRA (ours)**
	Locomotion		
Velocity-Flat-H1	34.6 ± 1.4	34.1 ± 1.4	**35.5** ± 1.1
Velocity-Flat-G1	32.2 ± 1.3	33.1 ± 1.0	**34.8** ± 0.8
Velocity-Rough-H1	22.4 ± 3.5	24.4 ± 0.5	**27.9** ± 0.9
Velocity-Rough-G1	40.6 ± 0.9	42.0 ± 1.1	**44.0** ± 1.7
Locomotion Avg	32.5	33.4	**35.6**
	In-hand manipulation		
Repose-Allegro	8022.1 ± 473.0	7703.1 ± 791.6	**8184.0** ± 203.9
Repose-Shadow	**12163.2** ± 202.1	11811.1 ± 192.1	12087.1 ± 116.3
Manipulation Avg	10092.7	9757.1	**10135.6**

G Comparison with Dropout-Q

We additionally compare with Dropout-Q, a representative critic-regularization method that applies dropout during critic training [27]. For its implementation and hyperparameter settings, we follow the original paper and the accompanying reference code. We evaluate Dense, Sparse, LoRA, and Dropout-Q using the same five random seeds (5–9) on Dog Run with SAC+BRC and Velocity-Rough-G1 with FastTD3+SimbaV2.

As shown in Table 11, Dropout-Q achieves a higher mean final return than the dense baseline on Velocity-Rough-G1. However, it does not provide a similar improvement on Dog Run, where its mean final return is slightly below that of the dense baseline. In contrast, LoRA achieves the highest mean final return among all four methods on both tasks.

Table 11: **Final policy return with Dropout-Q as an additional baseline.** Results are evaluated at the final normalized training step and reported as mean $\pm$ standard deviation over five random seeds (5–9). Bold entries indicate the highest mean return for each task.

Method	Dog Run SAC+BRC	Velocity-Rough-G1 FastTD3+SimbaV2
Dense	321.8 ± 35.1	40.73 ± 3.39
Sparse	409.0 ± 56.9	42.14 ± 4.96
LoRA	$\mathbf{476.1 \pm 46.0}$	$\mathbf{43.96 \pm 1.53}$
Dropout-Q	311.5 ± 45.5	42.69 ± 1.75

H Limitations

Our benchmark experiments focus on state-based continuous-control tasks, following the experimental settings of SimbaV2 and FastTD3. Although the proposed low-rank critic framework can be extended to other architectures and learning settings, we have not systematically evaluated pixel-based observations, offline reinforcement learning, or discrete-action domains. Evaluating these extensions remains an important direction for future work.

Our architecture-specific design ablations focus on SimbaV2, so the effects of LoRA rank and frozen-base initialization across other architectures remain to be systematically evaluated. Our experiments also do not fully disentangle the effects of low-rank parameterization and constrained optimization on the observed performance gains.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the method, its scope, and the main contributions: LoRA as critic regularization, a compatible hyperspherical normalization rule, and evaluation with SAC and FastTD3. The claims are supported by the method section, experiments in Section 6, and additional results in Appendix E.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Appendix H discusses the scope of the benchmark experiments, architecture-specific ablations, and remaining uncertainty about the mechanisms behind the performance gains. Appendix E also notes that LoRA is not intended to reduce wall-clock or memory cost in these settings' RL agents.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: The assumptions and complete proof for the existence of a positive scaling root are provided with Lemma 1.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: The method is specified in Section 5, and Appendix E reports task lists, architectures, optimizer settings, LoRA and sparse hyperparameters, evaluation protocol, and seed counts. The supplemental code contains all implementations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: The submission includes all the necessary codes in supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: Section 6 describes the algorithms, benchmarks, baselines, model scale, and comparison protocol. Appendix E gives the full task lists, hyperparameter tables with all settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes].

   Justification: The main learning curves and ablations report mean performance with shaded standard deviations over 5 seeds where shown. Appendix E states that final-return tables are computed over seeds 0–4 and report mean $\pm$ one standard deviation.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Appendix E includes the description of computational resources we used for this paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The research uses standard benchmarks and does not involve human subjects, private data, scraped datasets, or deployment on physical systems. We are not aware of any aspect of the work that conflicts with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes].

Justification: The work is foundational RL research whose likely positive impact is improved stability of critic learning in simulated control. But for specific downstream tasks, the proposed methods might need to be tuned(including LoRA rank and weight decay for LoRA) to achieve the best performance.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A].

Justification: The paper does not release pretrained generative models, scraped datasets, or other high-risk artifacts. The released asset is research code for simulated reinforcement-learning experiments.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes].

Justification: The manuscript cites the relevant prior codebases, algorithms, and benchmark suites.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes].

Justification: The new asset is the code implementing LoRA-regularized critics and the corresponding experiment configurations. Documentation, dependency files, hyperparameter tables, and run scripts are provided alongside the code.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A].

Justification: The paper does not involve crowdsourcing or any research with human subjects. All experiments are conducted in simulated reinforcement-learning environments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve human subjects, crowdsourcing, or participant data. Therefore, IRB or equivalent human-subjects approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: The core method, experiments, and baselines do not use LLMs as algorithmic components. Any LLM use limited to writing, editing, or formatting would not affect the methodology, scientific claims, or originality of the research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.