WARPSAC: TOWARDS THE EXTREMITY OF SCALABLE OFF-POLICY RL BY RETHINKING EXPLORATION AND EXPLOITATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling off-policy reinforcement learning (RL) through massively parallel simulation changes the data-regime assumptions behind these algorithms: canonical stabilizers are motivated by data-limited training, where replay buffers provide narrow state–action coverage, whereas parallel simulation offers diverse experience at high throughput, challenging their canonical roles. Across eight benchmark families spanning CPU-scale locomotion, GPU-parallel robotic simulation, dexterous manipulation, and humanoid whole-body control, we find that these stabilizers are strongly *data-regime-dependent*: parameter normalization helps under narrow replay coverage but restricts value fitting when data are abundant, clipped double-Q can be safely relaxed in high-throughput manipulation, and age-biased replay weighting is broadly useful, especially under limited network capacity. Turning this analysis into a prescription, we build **WarpSAC**, a regime-aware family that uses Sample Weight Decay as a regime-agnostic component and matches each regime with a prescribed variant: **WarpSAC-L** (Norm ON, clipped double-Q) for data-limited CPU-scale training, and **WarpSAC-A** (Norm OFF, single-Q) for data-abundant GPU-parallel training. **WarpSAC** improves normalized score–step AUC over FlashSAC by 4.5% across **nine CPU-scale environments** and 23.1% across **fourteen GPU-parallel environments**; lifts UnitreeG1TransportBox-v1 success rate from 19.8% to 96.4%; gains 19.1% in mean normalized wall-time AUC on **MuJoCo Playground**; and achieves **faster sim-to-real deployment** on `Unitree G1` than FlashSAC by 36.4% in wall time. These results argue that scalable off-policy RL should adapt its stabilizers to the available data regime; under this principle, **WarpSAC** advances the state of the art of scalable off-policy RL across regimes.

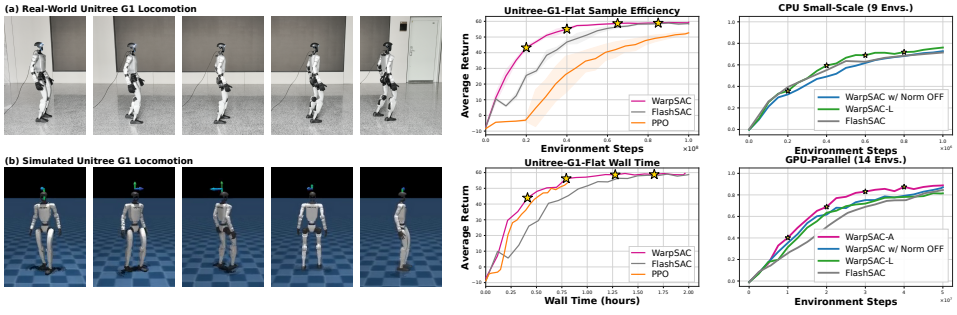

Figure 1: **Benchmark-scale and sim-to-real overview.** The left strips show real-world and simulated Unitree G1 locomotion, the middle panels report Unitree-G1-Flat sample efficiency and training wall time measured on a single NVIDIA A800 GPU, and the right panels summarize normalized CPU-scale and GPU-parallel learning curves. Across the benchmark curves, **WarpSAC** improves over FlashSAC, while the preferred stabilizing configuration depends on the data-collection regime.

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

1 INTRODUCTION

Scaling has become a central force in modern reinforcement learning (RL) for robot control: GPU-accelerated simulators and massively parallel environments now let agents collect diverse experience at rates unattainable in conventional single-environment training (Mittal et al., 2025; Zakka et al., 2025; 2026; Tao et al., 2024). This makes off-policy actor–critic methods attractive, since replay buffers reuse large volumes of interaction data and scalable implementations such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018a;b) and FlashSAC (Kim et al., 2026) are effective in high-dimensional robotic domains.

However, most classical off-policy stabilizers were designed under the opposite assumption: that replay coverage is narrow, and the learner must protect itself against extrapolation. In this limited data regime, exploration and conservative value estimation are essential. Entropy regularization pushes the policy toward unseen actions, clipped double-Q suppresses overestimation on poorly covered transitions, and normalization constrains the function class (Fujimoto et al., 2018; Haarnoja et al., 2018a; Lee et al., 2025). When massively parallel simulation replaces this scarcity with abundance, the same mechanisms may no longer be uniformly beneficial, as the bottleneck shifts from exploration and conservatism to exploiting abundant data efficiently. Strong parameter normalization can restrict expressive freedom; clipped double-Q can be overly pessimistic; and every extra conservative component adds wall-clock cost. Yet almost all scalable off-policy pipelines still inherit these stabilizers unchanged. The central question we ask in this paper is — *not whether these stabilizers are good in isolation, but when their benefits outweigh their restrictions.*

We answer this question through a controlled component-wise analysis based on FlashSAC (Kim et al., 2026). We isolate three axes that scalable off-policy learners usually bundle together: (i) replay-side data utilization, instantiated by Sample Weight Decay (SWD) (Wu et al., 2026), an age-biased sampler that concentrates updates on policy-relevant transitions; (ii) parameter-projection normalization, which controls the effective function class of the actor and critic; and (iii) critic multiplicity, embodied by clipped double-Q versus single-Q targets. Holding the training backbone, optimizer, environment interface, and network fixed, we vary each axis independently and measure its effect under data-limited (CPU-scale) versus data-abundant (GPU-parallel) regimes across eight benchmark families, 67 environments/tasks in total: DeepMind Control Suite hard tasks (Tassa et al., 2018), HumanoidBench (Sferrazza et al., 2024), Gym–MuJoCo (Todorov et al., 2012), MyoSuite (Caggiano et al., 2022), MuJoCo Playground (Zakka et al., 2025), MJLab (Zakka et al., 2026), IsaacLab (Mittal et al., 2025), and ManiSkill (Tao et al., 2024). The analysis yields a consistent message: **these stabilizers are data-regime-dependent, not universal**. Normalization helps when replay is narrow but restricts value fitting when replay is broad; clipped double-Q can be relaxed in high-throughput manipulation without degrading performance; SWD, in contrast, remains useful across both regimes.

Turning this analysis into a prescription, we build **WarpSAC**, a regime-aware family of off-policy RL algorithm. **WarpSAC** uses SWD as a consistent component and matches each stabilizer choice to the target data regime: **WarpSAC-L** (Norm ON, double-Q) for data-limited CPU-scale training, and **WarpSAC-A** (Norm OFF, Single-Q) for GPU-parallel training. Across nine CPU-scale environments, **WarpSAC** improves mean normalized score–step AUC over FlashSAC by 4.5%; across fourteen GPU-parallel environments, the corresponding AUC improves by 23.1%. On UnitreeG1TransportBox-v1, **WarpSAC**-A lifts success rate from 19.8% to 96.4%, and on MuJoCo Playground it gains 19.1% in mean normalized wall-time AUC. We further close the sim-to-real loop for `Unitree G1` in 35 minutes on a single A800 GPU, a 36.4% wall-clock reduction over FlashSAC. These gains require no change to the training backbone and no auxiliary networks, and often come from removing components: a regime-matched selection of stabilizers outperforms uniform inheritance.

Our contributions are threefold:

- **A regime-aware design principle for scalable off-policy RL.** Controlled component-wise ablations show that parameter normalization and clipped double-Q are data-regime-dependent stabilizers, whereas age-biased replay weighting is broadly beneficial—reframing scalable off-policy RL as a regime-matching rather than a stabilizer-stacking problem.

- **WarpSAC, a regime-aware refinement of FlashSAC that wins across regimes.** Guided by the analysis, **WarpSAC** pairs each regime with its stabilizer configuration—**WarpSAC**-L for CPU-scale and **WarpSAC**-A for GPU-parallel training—while exploiting data better with SWD under a shared backbone.
- **State-of-the-art results across eight benchmark families. WarpSAC** improves normalized score–step AUC over FlashSAC by 4.5% across nine CPU-scale environments and 23.1% across fourteen GPU-parallel environments, lifts UnitreeG1TransportBox-v1 success rate from 19.8% to 96.4%, and gains 19.1% in mean normalized wall-time AUC on MuJoCo Playground.

2 Related Work

Scalable off-policy reinforcement learning. Off-policy RL reuses past experience through replay and is therefore attractive for continuous control. Soft Actor-Critic (SAC) combines off-policy actor–critic learning with entropy regularization and is a standard continuous-control baseline (Haarnoja et al., 2018a;b); later methods improve stability or sample efficiency via lower value-estimation error, higher update-to-data ratios, critic ensembles, or latent dynamics (Fujimoto et al., 2018; Chen et al., 2021; Hansen et al., 2024), and recent robot-learning systems show that off-policy RL benefits from high-throughput simulation, large replay buffers, and larger networks (Lee et al., 2025; Kim et al., 2026). We follow this direction but ask a different question: which classical stabilizers remain useful when parallel data collection broadens replay coverage.

Replay weighting, data reuse, and exploitation. Replay buffers are commonly sampled uniformly, but non-uniform replay improves data reuse: prioritized experience replay emphasizes transitions with larger temporal-difference errors (Schaul et al., 2016), and recent work studies replay through non-stationarity and plasticity loss, where agents lose adaptability via primacy bias, dormant neurons, rank collapse, or value and policy churn (Nikishin et al., 2022; Sokar et al., 2023; Wu et al., 2026; Tang & Berseth, 2024; Tang et al., 2025). Sample Weight Decay (SWD) reweights samples by age (Wu et al., 2026); while originally motivated by plasticity preservation, we interpret it as an exploitation-oriented replay mechanism that changes how collected data are reused, rather than adding exploration or stronger conservatism.

Critic conservatism and clipped double-Q learning. A central challenge in off-policy actor–critic learning is value-estimation error. TD3 and SAC take the minimum over two target critics to reduce overestimation (Fujimoto et al., 2018; Haarnoja et al., 2018a), which is especially useful when the critic must evaluate poorly covered actions, where extrapolation error otherwise yields spuriously high values; related work addresses such errors via representation-level generalization (Ma et al., 2023) or by reducing out-of-batch churn (Tang & Berseth, 2024). Yet this target also introduces pessimism bias: when coverage is broad and value fitting dominates the bottleneck, clipped double-Q may be overly pessimistic or computationally redundant. Our single-Q experiments ask whether critic-side conservatism can be relaxed when parallel simulation already provides diverse replay.

Normalization and function-class control. Normalization and norm control are widely used to stabilize deep networks: batch, layer, weight, spectral, and RMS normalization control different aspects of optimization, activation statistics, or parameter geometry (Ioffe & Szegedy, 2015; Ba et al., 2016; Salimans & Kingma, 2016; Miyato et al., 2018; Zhang & Sennrich, 2019). In RL they matter because bootstrapped targets and non-stationary replay amplify value-function instability; recent architectures such as SimbaV2 use hyperspherical normalization for scalable RL, while XQC studies how critic conditioning affects optimization geometry (Lee et al., 2025; Palenicek et al., 2026). Rather than proposing a new normalization layer, we study when parameter projection normalization should be retained or relaxed, and find that norm control helps in data-limited regimes but can restrict exploitation when replay coverage is already broad.

Exploration complexity and data coverage. Theoretical analyses of exploration relate sample complexity to uncertainty over the value-function class: eluder dimension measures sequential dependence within a class and appears in optimistic-exploration analyses under function approximation (Russo & Van Roy, 2013; Osband & Van Roy, 2014), while norm and Lipschitz constraints control the effective complexity of neural function classes and spectral-norm measures analyze their generalization (Bartlett et al., 2017; Miyato et al., 2018). This motivates parameter projection normalization as an exploration- and stability-oriented mechanism that reduces harmful extrapolation

on under-covered regions; our results show that its value is regime-dependent, since the same constraint can reduce expressive freedom and weaken exploitation when parallel simulation already provides broad coverage.

3 PRELIMINARIES

Markov decision process. We consider continuous-control reinforcement learning in a discounted Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(s'|s,a)$ is the transition kernel, $r(s,a)$ is the reward function, and $\gamma \in [0,1)$ is the discount factor. At each timestep, the agent observes s_t, samples an action $a_t \sim \pi(\cdot|s_t)$, receives reward r_t, and transitions to s_{t+1}. We use $\rho_\pi(s,a)$ to denote the discounted state-action visitation distribution induced by policy π, and write the standard objective as $J(\pi) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[\sum_{t\geq 0} \gamma^t r(s_t,a_t)]$.

Soft actor-critic. Our study builds on Soft Actor-Critic (SAC), an off-policy actor–critic algorithm under the maximum-entropy RL framework (Haarnoja et al., 2018a;b) that augments the return objective with an entropy term weighted by α, i.e., $J_{\mathrm{SAC}}(\pi) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[\sum_{t\geq 0} \gamma^t (r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t)))]$. In practice, SAC maintains a replay buffer $\mathcal{D}$, a stochastic policy π_θ, and two critics Q_{ϕ_1}, Q_{ϕ_2} to counter overestimation bias via the *clipped double-Q target* (Fujimoto et al., 2018): given transitions $(s,a,r,s') \sim \mathcal{D}$ and $a' \sim \pi_\theta(\cdot|s')$,

$$y = r + \gamma \left(\min_{i=1,2} Q_{\bar{\phi}_i}(s',a') - \alpha \log \pi_\theta(a'|s') \right),$$

and each critic is trained by minimizing the Bellman error $\mathcal{L}_{Q_i}(\phi_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(Q_{\phi_i}(s,a) - y)^2]$. The clipped double-Q operator suppresses spuriously high Q estimates on poorly covered actions but adds a second critic and a pessimism bias; a *single-Q* variant simply removes the minimization and uses one critic.

FlashSAC extends this foundation to large-scale robotic control with high-throughput data collection, larger models, reduced update frequency, and norm control (Kim et al., 2026): a low gradient-update-to-environment-step ratio shifts wall-clock cost from gradient computation to environment interaction, making larger actor–critic networks affordable, while column-wise renormalization of linear weights after each optimizer step keeps such critics stable and controls value extrapolation on poorly covered actions. These choices make FlashSAC a strong scalable backbone on which we build **WarpSAC**.

Parameter projection normalization. We treat FlashSAC's norm control as a controllable factor: after each optimization step, parameter projection constrains every weight matrix of f_θ by projecting W_ℓ into a Frobenius-norm ball, $W_\ell \leftarrow \Pi_{\|W\|_F \leq c_\ell}(W_\ell)$. Since $\|W_\ell\|_2 \leq \|W_\ell\|_F$, this also bounds the spectral norms and, for 1-Lipschitz activations, yields the standard Lipschitz bound $\mathrm{Lip}(f_\theta) \leq \prod_{\ell=1}^{L} \|W_\ell\|_2 \leq \prod_{\ell=1}^{L} c_\ell$, connecting the projection to spectral-norm-based complexity control (Bartlett et al., 2017; Miyato et al., 2018). We therefore view it as an exploration- and stability-oriented constraint that controls the effective function class but may restrict expressive freedom when replay coverage is already broad.

Weighted replay. Uniform replay samples each stored transition with equal probability. More generally, a weighted scheme samples transition $\tau_i = (s_i, a_i, r_i, s_i')$ from the buffer $\mathcal{D}_t = \{\tau_i\}_{i=1}^{N_t}$ with probability $p_t(i) = w_t(i) / \sum_{j=1}^{N_t} w_t(j)$, where $w_t(i) \geq 0$ is a sample weight, and trains the critic on the reweighted objective $\mathcal{L}^w_{Q_i}(\phi_i) = \mathbb{E}_{\tau_i\sim p_t}[(Q_{\phi_i}(s_i,a_i) - y_i)^2]$. Different choices of $w_t(i)$ recover uniform sampling and prioritized replay (Schaul et al., 2016). Sample Weight Decay (SWD) instantiates this idea with age-aware weights and was proposed as a lightweight replay-side method for mitigating plasticity loss (Wu et al., 2026); here we study it not only as a plasticity-preserving method, but also as a mechanism that facilitates data exploitation during off-policy learning.

Eluder dimension and exploration complexity. Eluder dimension measures the degree of sequential dependence within a function class and is commonly used to characterize exploration complexity under function approximation (Russo & Van Roy, 2013; Osband & Van Roy, 2014): a class with lower eluder dimension requires fewer informative observations to resolve uncertainty about future predictions. Norm and Lipschitz constraints reduce the effective complexity of the value-function class, which offers a conceptual lens for why parameter projection normalization may support exploration and stable generalization; we use this connection as interpretation rather than as a formal sample-complexity guarantee.

4 WARPSAC

The preceding sections show that classical off-policy stabilizers carry implicit data-regime assumptions: their benefits depend on whether replay coverage is narrow or broad. Rather than applying them uniformly, we propose **WarpSAC**, a regime-aware family built on FlashSAC that explicitly matches each stabilizer choice to the available data coverage. Concretely, **WarpSAC** varies three axes: replay weighting $w_t(i)$, parameter projection normalization, and critic multiplicity, and pairs different settings of these axes with different data regimes. The rest of this section states the underlying hypothesis (Section 4.1), identifies the three axes (Section 4.2), defines the regime-agnostic component SWD (Section 4.3), and combines them into the concrete **WarpSAC** prescription (Section 4.4).

4.1 DATA-REGIME HYPOTHESIS

Off-policy RL stabilizers such as clipped double-Q, parameter projection normalization, and uniform replay were developed under CPU-scale assumptions, where replay coverage is narrow and value extrapolation is fragile. GPU-parallel simulation changes the data regime: thousands of actors populate the buffer with diverse trajectories at high throughput, so the bottleneck shifts from obtaining sufficient coverage to fitting and exploiting high-value behavior from abundant data. We hypothesize that this shift changes the relative value of the classical stabilizers. In the *data-limited* regime, parameter normalization and clipped double-Q help by constraining the effective function class and suppressing spuriously high Q values on poorly covered actions, while any replay-side mechanism should make better use of the narrow buffer. In the *data-abundant* regime, the same conservative mechanisms can restrict value fitting or add unnecessary pessimism, while replay-side gains persist because targeting policy-relevant transitions is orthogonal to coverage.

4.2 THE THREE AXES

Given this hypothesis, which design knobs most directly encode regime assumptions? We isolate three, all on top of the FlashSAC (Kim et al., 2026). **WarpSAC** varies only these axes:

- **(A)** **Replay weighting $w_t(i)$.** Whether transitions in the buffer are sampled uniformly or with an age-dependent weight (Section 4.3).
- **(B)** **Parameter projection normalization.** Whether the FlashSAC column-wise weight renormalization is applied after each optimizer step (*Norm ON*) or disabled (*Norm OFF*).
- **(C)** **Critic multiplicity.** Whether the clipped double-Q target is used (two critics, the FlashSAC default) or replaced by a single critic (*Single-Q*).

We fix data-collection throughput and optimizer schedule to compare regimes cleanly, and vary network capacity only in the separate scale ablation (Section 5). Varying **(A)** – **(C)** independently is what enables regime-aware pairing rather than collapsing **WarpSAC** into a single "strong" or "weak" SAC recipe.

4.3 SAMPLE WEIGHT DECAY

Among the three axes, replay weighting is the only one we apply regardless of data regime. SWD instantiates **(A)**: it biases minibatch sampling toward recent transitions using a linear age decay. Concretely, let $\tau_i = (s_i, a_i, r_i, s_i')$ be a transition inserted into the replay buffer at time t_i, and let $A_t(i) = t - t_i$ be its age at training step t. SWD assigns each transition the age-dependent weight

$$w_t(i) = \max\left(w_{\min},\ 1 - \frac{A_t(i)}{T_{\text{decay}}}\right), \qquad p_t(i) = \frac{w_t(i)}{\sum_j w_t(j)}, \tag{1}$$

where T_{decay} is the decay horizon and $w_{\min} > 0$ is a floor that prevents old transitions from being fully discarded. Setting $T_{\text{decay}} = 0$ recovers uniform replay. SWD is implemented inside the replay buffer with no auxiliary networks, additional Bellman targets, or loss changes.

Policy performance is dominated by Bellman errors on state–action regions visited by the current policy and on regions along high-value trajectories (Wu et al., 2026). Uniform replay ignores this

structure and spends equal update probability on transitions from much older policies. SWD redirects a fixed update budget toward more policy-relevant transitions by changing only the minibatch distribution, without touching the nominal UTD or the update rule; a nonzero floor $w_{\min}$ preserves coverage. Because this argument depends on *policy age*, not on data volume, we retain SWD in both regimes, which serves as a consistent design choice confirmed empirically in Section 5.

Policy performance is dominated by Bellman errors on state–action regions visited by the current policy and on regions along high-value trajectories (Wu et al., 2026). Uniform replay ignores this structure and spends equal update probability on transitions from much older policies. SWD redirects a fixed update budget toward more policy-relevant transitions by changing only the minibatch distribution, without touching the nominal UTD or the update rule; a nonzero floor $w_{\min}$ preserves coverage. Because this argument depends on *policy age*, not on data volume, we retain SWD in both regimes—a design choice confirmed empirically in Section 5.

4.4 Regime-Aware Variants and Prescription

With SWD established as the regime-agnostic replay component, the remaining design question is how to set the other two axes, i.e., normalization and critic multiplicity, for each data regime. Table 1 summarizes the resulting variants. **WarpSAC**-L and **WarpSAC**-A are the two prescribed variants for their respective regimes; **WarpSAC** w Norm OFF serves as an intermediate ablation point. FlashSAC (no SWD, Norm ON, clipped double-Q) is the shared baseline.

These variants form a conservatism spectrum—from full (Norm ON, double-Q) to minimal (Norm OFF, single-Q)—with SWD applied throughout, enabling controlled comparison across data regimes. A key feature of **WarpSAC** is that the GPU-parallel recipe achieves gains by *removing* components rather than adding them: dropping normalization frees the critic to fit abundant data, and dropping the second critic halves critic-side computation. The result is an algorithm that is both simpler and stronger than the fully stabilized baseline. This stands in contrast to the common pattern of stacking mechanisms for robustness; **WarpSAC** shows that matching stabilizers to the data regime is more effective than uniform inheritance.

Table 1: **WarpSAC regime-aware prescription.** SWD is applied in all variants. Normalization and critic multiplicity are set according to the data regime. FlashSAC (no SWD, Norm ON, double-Q) serves as the shared baseline.

Data Regime	Variant Name	Replay **(A)**	Normalization **(B)**	Critic **(C)**
Data-limited (CPU-scale)	**WarpSAC**-L	SWD	Norm ON	Clipped Double-Q
Data-abundant (GPU-parallel)	**WarpSAC**-A	SWD	Norm OFF	Single-Q
Ablation	**WarpSAC** w Norm OFF	SWD	Norm OFF	Clipped Double-Q

In short, **WarpSAC** is not a single algorithm but a design principle: the same three-axis framework produces different concrete recipes for different data regimes, and the experiments in Section 5 confirm that this regime-aware selection consistently outperforms uniform stabilizer inheritance.

Practitioner's Guide: Choosing a WarpSAC Variant

- **Use WarpSAC-L for Data-limited (CPU-scale) Regime**: Normalization and conservatism stabilizes value extrapolation under narrow replay coverage.
- **Use WarpSAC-A for Data-abundant (GPU-parallel) Regime**: Removing normalization and conservatism frees the critic to exploit broad, rapidly refreshed replay data.
- **Use SWD in Both Regimes**: Always enable SWD. Age-aware replay improves update efficiency regardless of data volume.

5 EXPERIMENTS

Earlier sections introduced the data-regime hypothesis and the **WarpSAC** family, which studies how canonical off-policy RL configurations should adapt to a shift in data regime: components that help under limited replay may become restrictive when data are abundant. In the experiments, we evaluate this hypothesis across eight benchmark families spanning CPU-scale and massively parallel GPU training, organizing the experiments around five questions. Sections 5.2 and 5.3 address Q1–Q3 in the two regimes. Further, we move on from simulation to real-world evaluation of **WarpSAC** for `Unitree-G1` to address Q4 in Section 5.4. Finally, we address Q5 through a capacity-focused mechanism analysis, summarized in Section 5.5 and detailed in Appendix F.10.

The experiments address five questions: (Q1) whether **WarpSAC**-L and **WarpSAC**-A excel in their intended regimes; (Q2) whether SWD remains beneficial as the regime-agnostic component in both regimes; (Q3) whether normalization and clipped double-Q shift from beneficial to restrictive as data scale increases; (Q4) whether **WarpSAC** achieves faster sim-to-real training than FlashSAC under the same setup; and (Q5) how replay weighting interacts with network capacity across the two regimes.

5.1 EXPERIMENTAL SETUP

We evaluate **WarpSAC** across eight benchmark families under two operational data-collection regimes:

- **Data-limited (CPU-scale):** MuJoCo (Todorov et al., 2012), DeepMind Control Suite hard tasks (Tassa et al., 2018), HumanoidBench (Sferrazza et al., 2024), and MyoSuite (Caggiano et al., 2022).
- **Data-abundant (GPU-parallel):** MuJoCo Playground (Zakka et al., 2025), IsaacLab (Mittal et al., 2025), MJLab (Zakka et al., 2026), and ManiSkill (Tao et al., 2024).

These domains cover standard locomotion, humanoid control, dexterous manipulation, and massively parallel GPU simulation. For return-based domains, we report episodic return; for manipulation domains with sparse success signals (ManiSkill), we report average success metrics following the benchmark convention.

We compare four variants. **FlashSAC** is the uniform-replay baseline (no SWD, Norm ON, clipped double-Q). **WarpSAC-L** adds SWD while keeping normalization and clipped double-Q enabled. **WarpSAC w Norm OFF** adds SWD and disables parameter projection normalization, retaining clipped double-Q. For GPU-parallel domains, we additionally evaluate **WarpSAC-A**, which further removes clipped double-Q and uses a single critic. All variants share the same training backbone, optimizer, environment interface, and logging pipeline; only the three axes **(A)** – **(C)** are varied. Full hyperparameters, network architecture, and training details are provided in Appendix D.

5.2 DATA-LIMITED REGIME: CPU-SCALE ENVIRONMENTS

We first examine representative CPU-scale environments where replay coverage is relatively limited and value extrapolation is more likely to affect learning stability. Figure 2 compares FlashSAC with **WarpSAC** variants on MuJoCo, DMC hard tasks, HumanoidBench, and MyoSuite tasks. These environments cover standard locomotion, humanoid control, whole-body control, and dexterous manipulation, while avoiding massively parallel GPU data collection.

Across these tasks, SWD consistently improves over FlashSAC or reaches stronger final performance, suggesting that age-aware replay weighting improves the effective use of collected data. On humanoid-run, h1-slide-v0, and myo-pen-twirl-hard, the normalized **WarpSAC** variant achieves the strongest final performance and often learns more stably. This supports our hypothesis that parameter projection normalization is useful when replay coverage is narrower: by constraining the effective function class, normalization helps control value extrapolation and stabilizes actor–critic updates. On Humanoid-v4, the norm-off variant reaches a slightly higher final return, but both **WarpSAC** variants remain clearly above FlashSAC, indicating that the replay-side benefit of SWD is robust even when the best normalization choice is task-dependent.

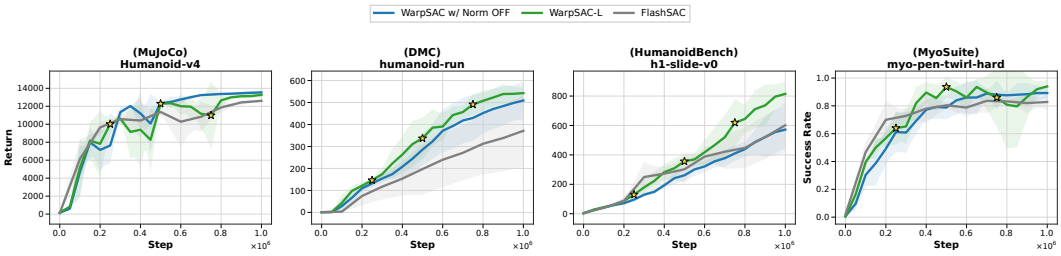

Figure 2: **Learning curves of WarpSAC and FlashSAC in the data-limited regime (CPU-scale).** Return (mean ± std over five seeds) versus environment steps for FlashSAC, **WarpSAC**-L, and **WarpSAC** w/ Norm OFF on representative tasks from MuJoCo, DMC, HumanoidBench, and MyoSuite. Both **WarpSAC** variants clearly surpass FlashSAC on all four tasks, and **WarpSAC**-L attains the strongest final return on humanoid-run, h1-slide-v0, and myo-pen-twirl-hard.

Overall, the CPU-scale results show that **WarpSAC**-L, i.e., the prescribed data-limited variant, is the strongest configuration (Q1), driven by SWD's broad benefit across all tasks (Q2) and normalization's role in constraining value extrapolation under limited replay (Q3). Normalization is thus not universally beneficial: its role depends on how much diverse replay data the learner has.

5.3 Data-Abundant Regime: GPU-Parallel Environments

We turn to GPU-parallel environments, where many actors collect data simultaneously and the replay buffer is refreshed with broad state–action coverage. Figure 3 compares FlashSAC, **WarpSAC** with and without parameter normalization, and a single-Q norm-off variant on IsaacLab, MuJoCo Playground, MJLab, and ManiSkill. This comparison tests whether conservative SAC components remain necessary when scalable simulation already provides diverse replay data.

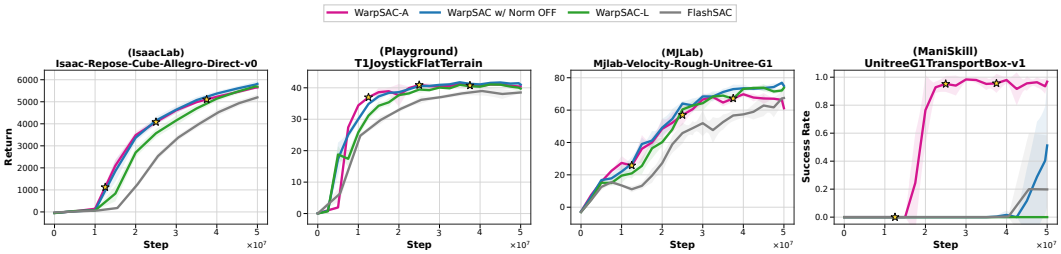

Figure 3: **Learning curves of WarpSAC and FlashSAC in the data-abundant regime (GPU-parallel).** Return (mean ± std over five seeds) versus environment steps for FlashSAC, **WarpSAC**-L, **WarpSAC** w/ Norm OFF, and **WarpSAC**-A on representative tasks from IsaacLab, MuJoCo Playground, MJLab, and ManiSkill. **WarpSAC** w/ Norm OFF matches or outperforms **WarpSAC**-L on IsaacLab, MuJoCo Playground, and MJLab, and **WarpSAC**-A attains the strongest final return on ManiSkill.

The results show a different pattern from the CPU-scale setting. In IsaacLab, MuJoCo Playground, and MJLab, the norm-off variant is competitive with or stronger than the normalized variant, indicating that parameter projection normalization can become restrictive when the replay buffer already provides sufficient coverage. On the MJLab rough-terrain Unitree G1 task, the two-critic **WarpSAC** variants outperform the uniform-replay FlashSAC baseline, while **WarpSAC**-A is less consistent. This shows that critic-side conservatism remains task-dependent even within GPU-parallel training. In ManiSkill, the single-Q norm-off variant achieves the strongest performance, suggesting that clipped double-Q conservatism can be relaxed in some high-throughput manipulation settings without degrading learning. In GPU-parallel regimes the bottleneck therefore shifts from conservative generalization to efficient exploitation of abundant data.

These results confirm that **WarpSAC**-A, i.e., the prescribed data-abundant variant, is a viable and often stronger choice at scale (Q1), because normalization and clipped double-Q shift from beneficial to restrictive as data volume increases (Q3). When parallel simulation supplies broad replay

coverage, strong conservative biases may reduce the flexibility of the value function or add unnecessary computation. SWD remains useful across both regimes because it reallocates updates toward policy-relevant samples without changing the actor–critic objective.

5.4 Sim-to-Real Evaluation: Training Unitree G1 to Walk in the Real World

We additionally run a sim-to-real case study on `Unitree-G1-Flat` with Unitree Robotics' `unitree_rl_mjlab` pipeline, using the asymmetric actor–critic formulation (98-dimensional actor observation, 211-dimensional privileged critic observation) and keeping the environment, reward design, and adaptation pipeline fixed across methods. On an end-to-end A800 run, **WarpSAC** reaches the target performance in roughly **35 minutes** versus about **55 minutes** for FlashSAC, a 36.4% wall-clock reduction. Full curves, the strengthened PPO reference, and the real-world deployment video are reported in Appendix F.12.

5.5 Mechanism Analysis: Capacity Across Regimes

To address Q5, we ablate how replay weighting and parameter normalization interact with model capacity in both regimes. The capacity sweeps for the CPU-scale tasks and for MuJoCo Playground, together with their full discussion, are reported in Appendix F.10.

6 Conclusion

Scaling data collection changes the fundamental assumption underlying canonical off-policy RL. We focus on the data-regime shift and formalized this observation as a *data-regime hypothesis*: the utility of an off-policy stabilizer is not a fixed property of the algorithm, but a function of how well replay coverage matches the class it is designed to protect. Building on this hypothesis, we proposed **WarpSAC**, a regime-aware family of off-policy RL algorithms that treats SWD as a regime-agnostic component and matches conservative stabilizers to the target data regime. **WarpSAC**-L (SWD, Norm ON, double-Q) targets data-limited CPU-scale training, while **WarpSAC**-A (SWD, Norm OFF, single-Q) targets data-abundant GPU-parallel training. Across eight benchmark families spanning both regimes, this prescription is validated by controlled ablations and consistent aggregate gains over FlashSAC. Among the three studied axes, SWD is the only component that remains consistently beneficial across all eight benchmark families and network capacities, justifying its role as the regime-agnostic core of **WarpSAC**. Notably, the aggregate gains are often obtained by *removing* conservative components rather than adding new ones, reframing scalable off-policy RL as a regime-matching problem rather than a stabilizer-stacking one.

Limitations and Future Work. Our current prescription selects between **WarpSAC**-L and **WarpSAC**-A offline based on the target regime; deployments that span regimes—for example, pretraining under narrow data followed by parallel fine-tuning—would benefit from an *online regime-adaptive* variant that monitors replay coverage or value-extrapolation signals and continuously modulates normalization strength and critic multiplicity. More broadly, our analysis is grounded in the FlashSAC backbone and focuses on three axes; a natural next step is to examine whether other canonical stabilizers, e.g., entropy weighting, target-network delay, gradient clipping, or replay-ratio schedules, share the same regime-dependent behavior.

Overall, scalable RL requires data-regime-aware algorithm design: classical stabilizers are valuable when data are scarce but can impose unnecessary bias or computation when parallel simulation supplies broad replay, and a promising direction is to adapt them online as coverage and uncertainty change.

AI USE STATEMENT

We used generative AI tools only for language polishing and grammar correction of author-written text. All ideas, method design, analysis, figures, and experimental results were produced by the authors. We did not use generative AI tools to generate research ideas, to design or implement the method, to write or debug code, or to produce and interpret experimental results. All AI-assisted text was reviewed by the authors, who take responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This work studies reinforcement-learning algorithms for simulated control. It does not introduce new deployment systems or human-subject data. The sim-to-real case study reports a laboratory deployment of a locomotion policy and involves no human participants and no personal data. Potential societal effects are therefore indirect and mainly relate to future uses of improved control algorithms.

REPRODUCIBILITY STATEMENT

We provide the details needed to reproduce the reported results. Appendix D gives the SWD implementation together with the exact age-biased sampling code, the hyperparameter profiles shared across environments, and the actor and critic network architectures. Appendix E lists every benchmark suite and task version together with its observation and action dimensions and the normalization references used to score our runs. Appendix F reports per-environment curves for all benchmark families in both regimes. All experiments use five seeds $\{1, 2, 3, 4, 5\}$ and figures report the mean with the standard deviation across seeds unless the caption states otherwise. Data-collection throughput, optimizer schedule, and evaluation protocol are held fixed across methods, so each comparison varies only the axes stated in the main text.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Peter L. Bartlett, Dylan J. Foster, and Matus J. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. MyoSuite: A contact-rich simulation suite for musculoskeletal motor control. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168 of *Proceedings of Machine Learning Research*, pp. 492–507. PMLR, 2022.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

Scott Fujimoto and Shixiang Shane Gu. TD7: A high-performance algorithm for continuous control, 2023. URL https://arxiv.org/abs/2306.02451.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1587–1596. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2024.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456. PMLR, 2015.

Donghu Kim, Youngdo Lee, Minho Park, Kinam Kim, I Made Aswin Nahendra, Takuma Seno, Sehee Min, Daniel Palenicek, Florian Vogt, Danica Kragic, Jan Peters, Jaegul Choo, and Hojoon Lee. FlashSAC: Fast and stable off-policy reinforcement learning for high-dimensional robot control. In *Robotics: Science and Systems*, 2026. URL https://arxiv.org/abs/2604.04539.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2502.15280.

Yi Ma, Hongyao Tang, Dong Li, and Zhaopeng Meng. Reining generalization in offline reinforcement learning via representation distinction. In *Advances in Neural Information Processing Systems*, volume 36, pp. 40773–40785, 2023.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen, Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi Jim Fan, Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025. URL https://arxiv.org/abs/2511.04831.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2022.

Ian Osband and Benjamin Van Roy. Model-based reinforcement learning and the eluder dimension. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Daniel Palenicek, Florian Vogt, Joe Watson, Ingmar Posner, and Jan Peters. XQC: Well-conditioned optimization accelerates deep reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.25174.

Daniel Russo and Benjamin Van Roy. Eluder dimension and the sample complexity of optimistic exploration. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoid-Bench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2023.

Hongyao Tang and Glen Berseth. Improving deep reinforcement learning by reducing the chain effect of value and policy churn. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Hongyao Tang, Johan S. Obando-Ceron, Pablo Samuel Castro, Aaron C. Courville, and Glen Berseth. Mitigating plasticity loss in continual reinforcement learning by reducing churn. In *ICML*, 2025.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Zihao Wu, Hongyao Tang, Yi Ma, Jiashun Liu, Yan Zheng, and Jianye Hao. The rank and gradient lost in non-stationarity: Sample weight decay for mitigating plasticity loss in reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5DpzzTPnJZ.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. MuJoCo Playground, 2025. URL https://arxiv.org/abs/2502.08844.

Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. mjlab: A lightweight framework for gpu-accelerated robot learning, 2026. URL https://arxiv.org/abs/2601.22074.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Appendix Contents

A The Use of Large Language Models **14**

B Impact Statement **14**

C Wall-Clock Time **14**

C.1 Update 14

C.2 SWD Sampling 14

C.3 A800 End-to-End Wall Time 15

C.4 GPU-Parallel Learning Curves by Wall Time 16

D Experimental Details **19**

D.1 SWD Implementations 19

D.2 Hyperparameters 21

D.3 Network Architecture 23

E Environment Details **24**

F Full Results **29**

F.1 Gym–MuJoCo 29

F.2 DMC 30

F.3 MyoSuite 31

F.4 HumanoidBench 32

F.5 MuJoCo Playground 33

F.6 IsaacLab 34

F.7 MJLab 35

F.8 ManiSkill 36

F.9 Ablation Learning Curves 37

F.10 Mechanism Analysis: SWD $\times$ Capacity 38

F.11 Mechanism Analysis: Normalization $\times$ Capacity 39

F.12 Sim-to-Real Details 40

A THE USE OF LARGE LANGUAGE MODELS

LLMs were used only for language polishing and grammar correction. All ideas, method design, analysis, figures, and experimental results were produced by the authors.

B IMPACT STATEMENT

This work studies reinforcement-learning algorithms for simulated control. It does not introduce new deployment systems or human-subject data. Potential societal effects are therefore indirect and mainly relate to future uses of improved control algorithms.

C WALL-CLOCK TIME

C.1 UPDATE

We measure the wall-clock cost of one complete SAC update after reimplementing the FlashSAC backbone in JAX with Flax NNX. This benchmark isolates learner computation from environment stepping, replay-buffer sampling, logging, and evaluation. We use synthetic batches on an NVIDIA GeForce RTX 5060 Ti with observation dimension 128, action dimension 12, and batch size 2048. Each result is measured after JIT or `torch.compile` warmup, using 20 warmup repetitions and 80 measured repetitions per round over three alternating rounds. We report the median over the three round-level means. For PyTorch AMP, we use the FlashSAC implementation with CUDA autocast to `float16` and `GradScaler`.

Table 2: **SAC update micro-benchmark.** Lower is better.

Implementation	Full Update
JAX/NNX bfloat16	7.895 ms
JAX/NNX float32	13.244 ms
PyTorch FlashSAC, `torch.compile`, AMP on	13.922 ms
PyTorch FlashSAC, `torch.compile`, AMP off	12.293 ms

The JAX/NNX bfloat16 update takes 7.895 ms, which is $1.68\times$ faster than JAX/NNX float32, $1.76\times$ faster than compiled PyTorch FlashSAC with AMP, and $1.56\times$ faster than compiled PyTorch FlashSAC without AMP under the same synthetic-batch setting. In the compiled PyTorch implementation, AMP accelerates the critic update in isolation, but the complete update is slightly faster without AMP because the full step also includes the actor update, temperature update, normalization, and scaler/autocast overhead. These measurements suggest that the main computational benefit of the JAX/NNX implementation comes from the compiled update path, while bfloat16 further improves update throughput.

C.2 SWD SAMPLING

We also measure the wall-clock overhead introduced by SWD replay sampling. The benchmark uses the same NVIDIA GeForce RTX 5060 Ti and the same synthetic transition shape as above, with a replay capacity of 10^6, batch size 2048, observation dimension 128, and action dimension 12. We compare uniform replay, exact SWD sampling, and the bucketed approximate SWD sampler with 2000 buckets. The SWD decay horizon is $80{,}000$ and $w_{\min} = 0.1$. Each result measures only `JaxBuffer.sample` after JIT warmup, using 80 warmup repetitions and 300 measured repetitions per round over nine alternating rounds. We report the median over the nine round-level means.

Table 3: **Replay sampling micro-benchmark.** Lower is better.

Sampler	CPU Sample	GPU Sample
Uniform replay	0.710 ms	0.267 ms
Exact SWD	3.263 ms	1.751 ms
Approximate SWD, 2000 buckets	0.959 ms	1.187 ms

On CPU, exact SWD is $4.59\times$ slower than uniform replay because it constructs and normalizes a probability vector over the full million-transition buffer at each sample call. The bucketed approximation avoids this full scan and reduces sampling time from 3.263 ms to 0.959 ms, although it remains $1.35\times$ slower than uniform replay. On GPU, exact SWD costs 1.751 ms per sample and is $6.56\times$ slower than uniform replay, while the bucketed approximation reduces this cost to 1.187 ms. In absolute terms, however, the exact GPU sampling overhead is still small relative to learner computation: exact SWD sampling plus a full JAX/NNX bfloat16 update takes about 9.646 ms, which remains faster than compiled PyTorch FlashSAC with AMP (13.922 ms) and without AMP (12.293 ms). These results support using the approximate sampler for CPU-scale replay buffers and exact SWD for GPU-parallel experiments, where preserving the exact sampling distribution is affordable and does not erase the throughput advantage of the JAX/NNX bfloat16 implementation.

C.3 A800 End-to-End Wall Time

We measure end-to-end wall-clock learning on NVIDIA A800 GPUs. This includes environment stepping, replay sampling, learner updates, logging, and evaluation. We report four completed IsaacLab environments and average **WarpSAC** w Norm OFF over five seeds.

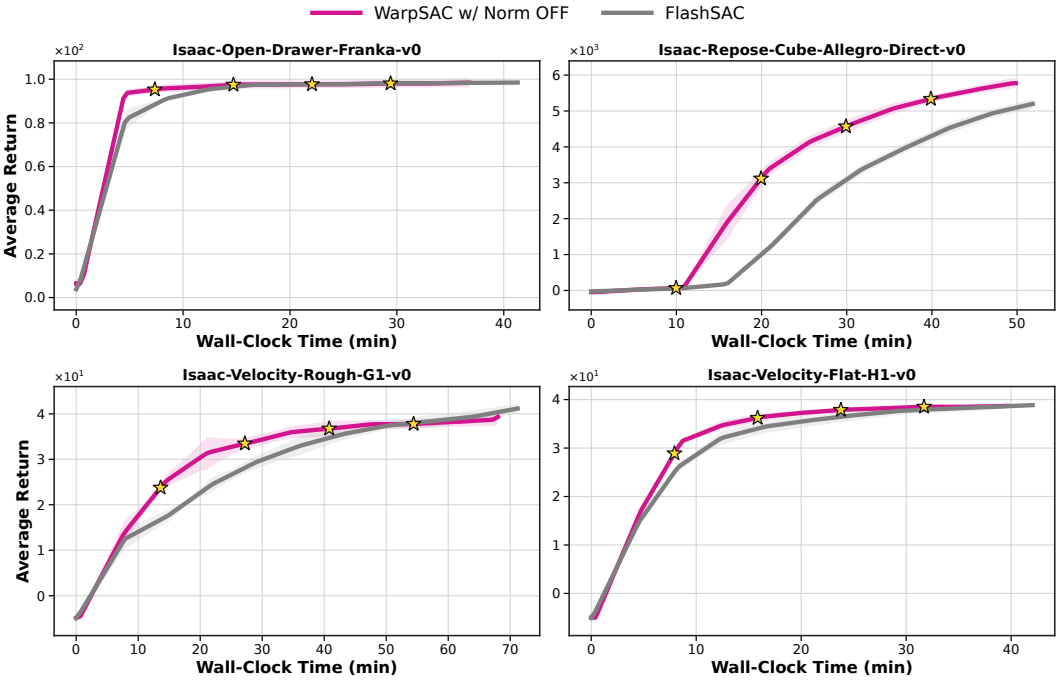

Figure 4: Actual A800 return–wall-time curves on four completed IsaacLab environments. Curves report average return versus elapsed wall-clock time.

Figure 4 shows that **WarpSAC** w Norm OFF reaches the 50.0M-step endpoint at a similar or slightly faster wall-clock speed than FlashSAC on all four selected environments. The final wall-clock speedups are $1.11\times$ on Isaac-Open-Drawer-Franka-v0, $1.03\times$ on Isaac-Repose-Cube-Allegro-Direct-v0, $1.04\times$ on Isaac-Velocity-Rough-G1-v0, and $1.05\times$ on Isaac-Velocity-Flat-H1-v0. Final returns are comparable on Isaac-Open-Drawer-Franka-v0, Isaac-Velocity-Rough-G1-v0, and Isaac-

Velocity-Flat-H1-v0, while **WarpSAC** w Norm OFF obtains a higher final return on Isaac-Repose-Cube-Allegro-Direct-v0.

These real-machine measurements indicate that the JAX/NNX implementation does not introduce a meaningful end-to-end slowdown relative to FlashSAC in GPU-parallel IsaacLab training. We therefore use the measured **WarpSAC** wall-time traces as a practical speed proxy only for domains whose FlashSAC logs contain sample-efficiency curves but not full wall-clock traces. In those cases, the compute-efficiency figures below plot FlashSAC on the wall-time axis of the corresponding **WarpSAC** Norm ON, num-Q $= 2$ run, which keeps the speed assumption matched to the closest two-critic baseline and avoids attributing additional speed differences to FlashSAC.

C.4 GPU-Parallel Learning Curves by Wall Time

Figures 5–8 report GPU-parallel learning curves against wall-clock time for MuJoCo Playground, IsaacLab, MJLab, and ManiSkill. For MJLab, every method is plotted against its own logged wall-clock trace, including FlashSAC. For the other three domains, the FlashSAC sample-efficiency curves use the wall-time axis of the corresponding **WarpSAC** Norm ON, num-Q $= 2$ run, following the speed-proxy convention motivated above.

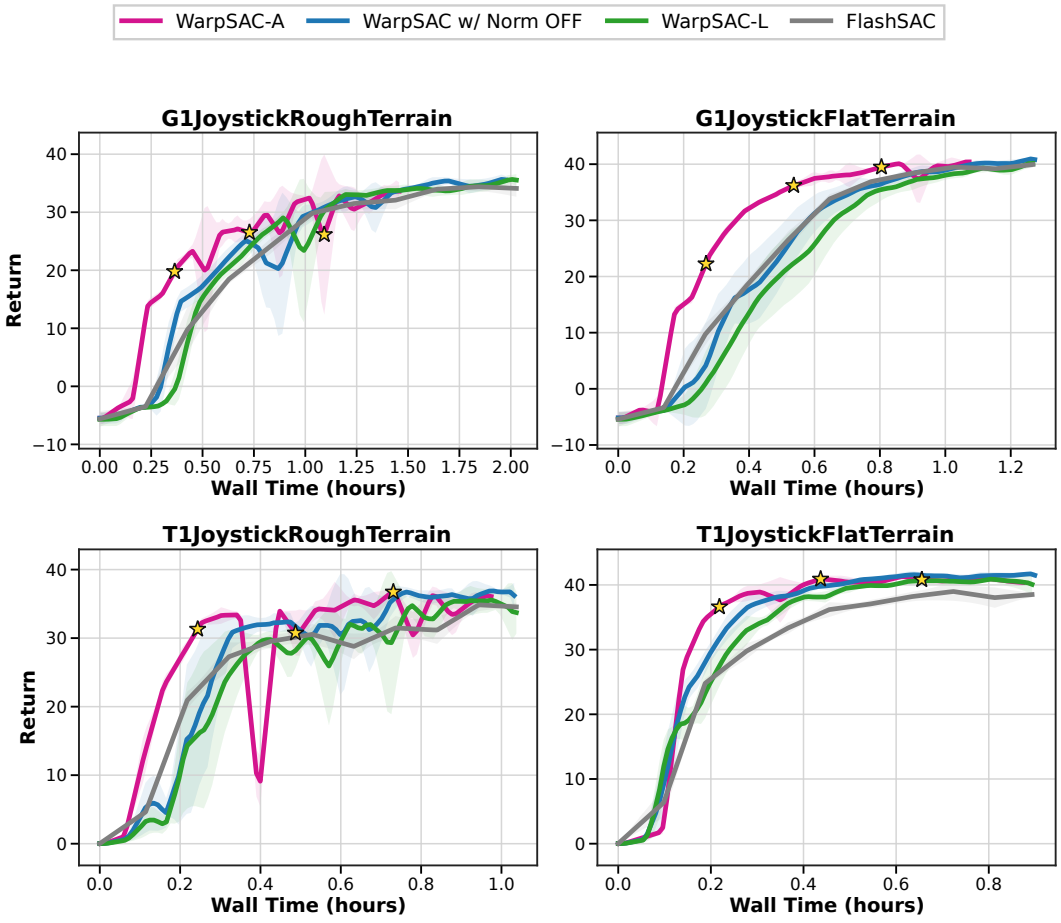

Figure 5: **MuJoCo Playground learning curves by wall-clock time.** Curves report mean return with standard deviation across five seeds. FlashSAC is plotted using the same wall-time axis as the corresponding **WarpSAC** Norm ON, num-Q $= 2$ run.

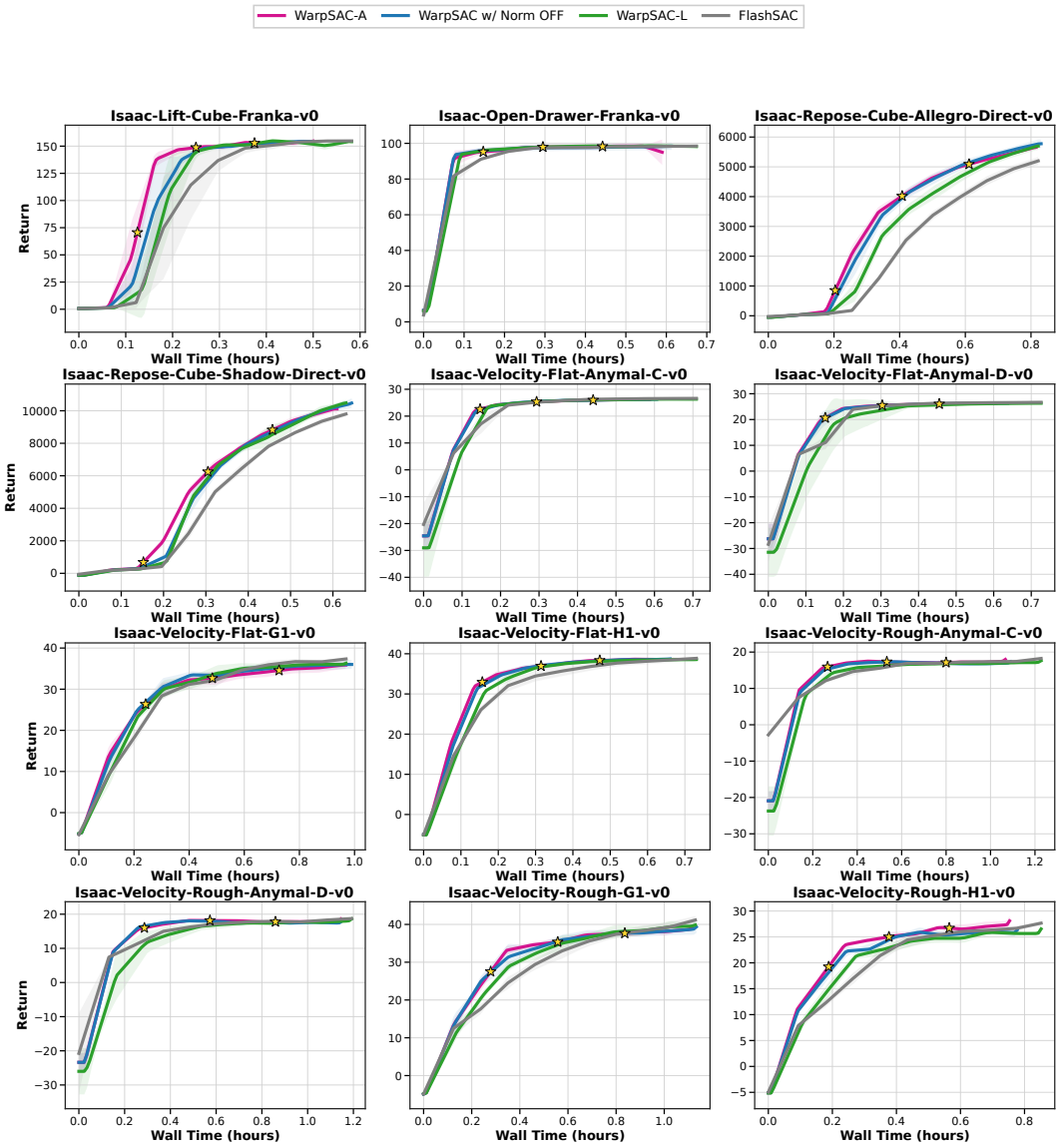

Figure 6: **IsaacLab learning curves by wall-clock time.** Curves report mean return with standard deviation across five seeds for the evaluated GPU-parallel IsaacLab tasks. FlashSAC is plotted using the same wall-time axis as the corresponding **WarpSAC** Norm ON, num-Q $= 2$ run.

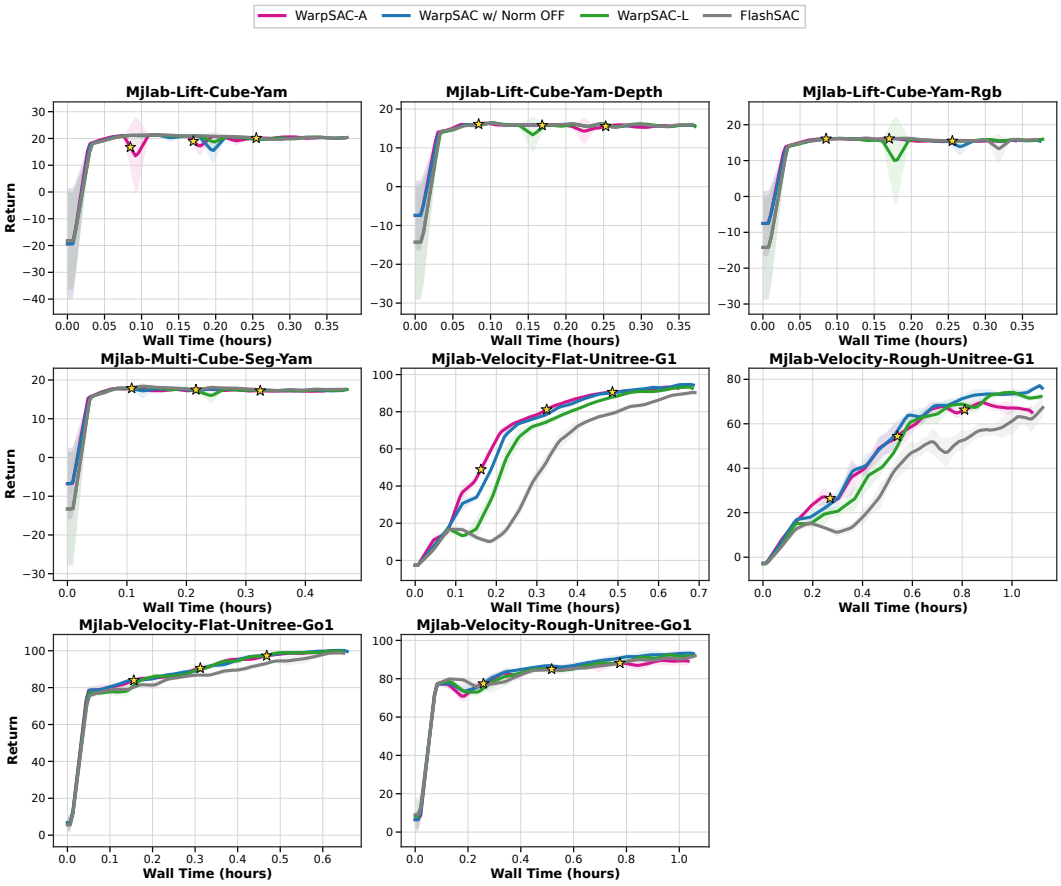

Figure 7: **MJLab learning curves by wall-clock time.** Curves report mean return with standard deviation across five seeds for Unitree locomotion and Yam manipulation tasks. All methods, including FlashSAC, use their own logged wall-clock traces.

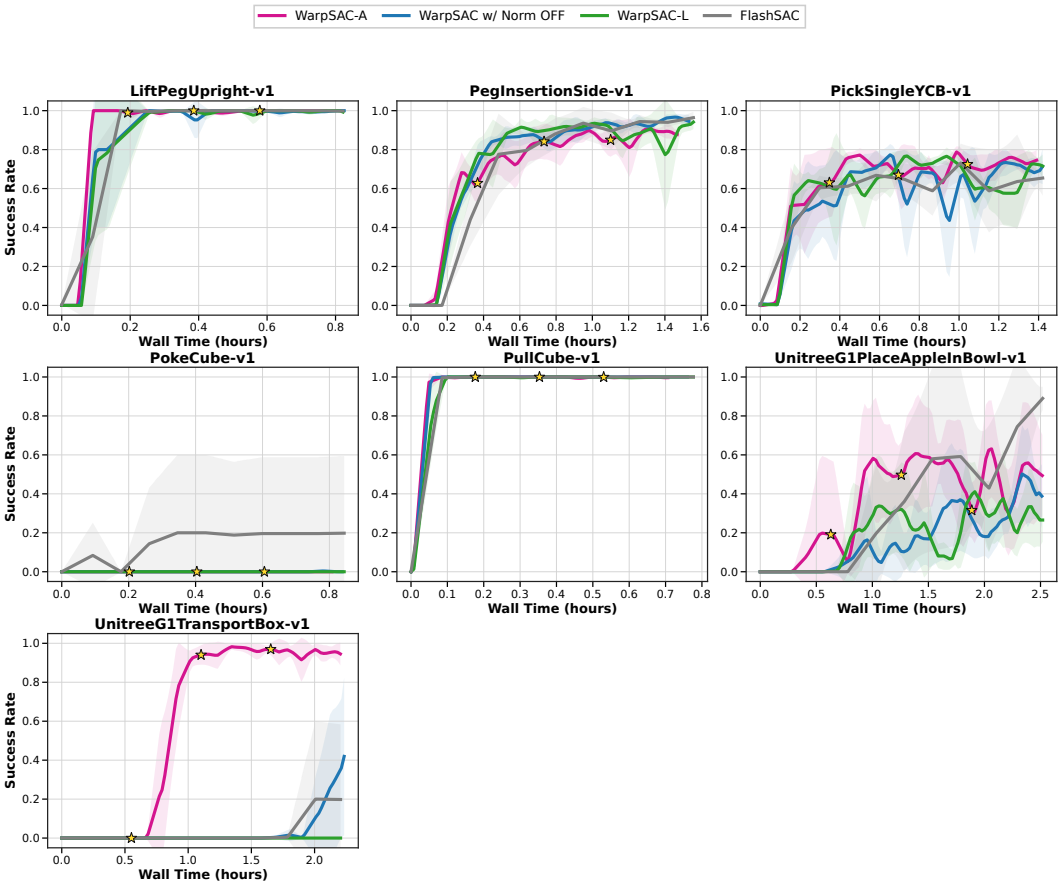

Figure 8: **ManiSkill learning curves by wall-clock time.** Curves report mean success-once rate with standard deviation across five seeds. FlashSAC is plotted using the same wall-time axis as the corresponding **WarpSAC** Norm ON, num-Q = 2 run.

D Experimental Details

D.1 SWD Implementations

SWD is implemented in the replay buffer. Each transition receives an insertion timestamp, and sampling weights are computed from transition age. Setting the decay horizon to zero recovers uniform replay, so FlashSAC and SWD variants share the same buffer implementation.

We use bucketed approximate sampling only for CPU-scale replay buffers: DMC hard tasks, MuJoCo, MyoSuite, and HumanoidBench. These runs store replay on the host with capacity 10^6. GPU-parallel domains, including MuJoCo Playground, MJLab, ManiSkill, and IsaacLab, use exact SWD because replay tensors are stored on GPU and categorical sampling is CUDA-accelerated through JAX.

The wall-clock micro-benchmark in Appendix C.2 supports this choice. With a 10^6-transition replay buffer and batch size 2048, bucketed CPU sampling reduces exact SWD sampling from 3.263 ms to 0.959 ms. On GPU, exact SWD sampling costs 1.751 ms, and exact SWD sampling plus a full JAX/NNX bfloat16 update remains faster than compiled PyTorch FlashSAC under the same synthetic-batch setting.

For a transition inserted at time t_i and sampled at replay time t, the age is $A_t(i) = t - t_i$. In the main experiments we use a recent-sample bias,

$$w_t(i) = \max\left(w_{\min}, 1 - \frac{A_t(i)}{T_{\text{decay}}}\right),$$

where $w_{\min} = 0.1$ in all reported runs. The resulting sampling probability is

$$p_t(i) = \frac{w_t(i)}{\sum_j w_t(j)}.$$

The implementation also supports the opposite bias direction through a negative decay horizon, but this option is not used in the reported experiments.

Exact Numpy implementation. The following code is the core Numpy implementation used for exact age-biased sampling. It first constructs a probability distribution over all valid replay entries and then samples minibatch indices with `np.random.choice`.

Listing 1: Exact Numpy implementation of Sample Weight Decay.

```
def _sample_indices(self, batch_size: int) -> np.ndarray:
   if self.linear_decay_step == 0:
      return np.random.randint(0, self.size, size=batch_size)

   if self.use_approximate_sampling:
      return self._sample_indices_with_approximate_bias(batch_size)

   return self._sample_indices_with_bias(batch_size)


def _sample_indices_with_bias(self, batch_size: int) -> np.ndarray:
   valid_timestamps = self.timestamps[:self.size]
   age = self.current_time - valid_timestamps

   if self.linear_decay_step > 0:
      weights = np.maximum(
         self.min_weight,
         1.0 - age / self.abs_linear_decay_step,
      )
   else:
      weights = np.minimum(
         1.0,
         self.min_weight + age / self.abs_linear_decay_step,
      )

   if weights.sum() <= 0:
      weights = np.ones_like(weights, dtype=np.float32)

   probabilities = weights / weights.sum()
   return np.random.choice(self.size, size=batch_size, p=probabilities)
```

This exact sampler is faithful to the SWD definition, but it normalizes over all valid replay entries at each sampling call.

Approximate bucketed sampling. For CPU replay, we approximate SWD with fixed logical buckets. The sampler computes one midpoint weight per bucket, samples a bucket from the bucket-level distribution, and then samples uniformly within that bucket. GPU-parallel runs do not use this approximation.

Listing 2: Approximate bucketed SWD sampling used to reduce wall-clock overhead.

```
def _sample_indices_with_approximate_bias(self, batch_size: int) -> np.ndarray:
   bucket_size = max(
      (self.size + self.num_buckets - 1) // self.num_buckets,
      1,
   )

   logical_starts = np.arange(0, self.size, bucket_size)
   logical_ends = np.minimum(logical_starts + bucket_size, self.size)
   logical_midpoints = (logical_starts + logical_ends - 1) // 2

   if self.full:
      bucket_midpoints = (self.ptr + logical_midpoints) % self.max_size
   else:
      bucket_midpoints = logical_midpoints

   bucket_timestamps = self.timestamps[bucket_midpoints]
   bucket_ages = self.current_time - bucket_timestamps

   if self.linear_decay_step > 0:
      bucket_weights = np.maximum(
```

```
            self.min_weight,
            1.0 - bucket_ages / self.abs_linear_decay_step,
        )
    else:
        bucket_weights = np.minimum(
            1.0,
            self.min_weight + bucket_ages / self.abs_linear_decay_step,
        )

    if bucket_weights.sum() <= 0:
        bucket_weights = np.ones_like(bucket_weights, dtype=np.float32)

    bucket_probabilities = bucket_weights / bucket_weights.sum()
    sampled_buckets = np.random.choice(
        len(logical_starts),
        size=batch_size,
        p=bucket_probabilities,
    )

    sampled_starts = logical_starts[sampled_buckets]
    sampled_ends = logical_ends[sampled_buckets]
    offsets = (
        np.random.random(size=batch_size)
        * (sampled_ends - sampled_starts)
    ).astype(np.int64)
    logical_indices = sampled_starts + offsets

    if self.full:
        return (self.ptr + logical_indices) % self.max_size
    return logical_indices
```

The approximation is piecewise constant in replay age. We use 2000 buckets by default, making sampling cost depend mainly on the number of buckets and the minibatch size rather than replay capacity.

For circular buffers, logical indices are mapped through the current write pointer before indexing physical storage.

D.2 HYPERPARAMETERS

Tables 4 and 5 list the shared training settings. Unless stated otherwise, all variants within a benchmark family use the same environment interface, optimizer, replay buffer, evaluation protocol, and network size. The controlled changes are replay decay horizon, parameter projection normalization, and critic ensemble size. We use five seeds, $\{1, 2, 3, 4, 5\}$.

Table 4: **Training profiles used across benchmark families.** CPU-scale domains use a single environment and CPU replay, while GPU-parallel domains use 1024 vectorized environments and GPU replay. Sim2Real retains the MJLab profile while doubling the training horizon to 100,001,792 environment steps.

Hyperparameter	**CPU-scale**	**Playground**	**MJLab**	**ManiSkill**	**IsaacLab**	**Sim2Real**
Benchmark families	DMC hard tasks, MuJoCo, MyoSuite, Humanoid-Bench	MuJoCo Playground	MJLab	ManiSkill	IsaacLab	Unitree-G1-Flat
Number of environments	1	1024	1024	1024	1024	1024
Total env. steps	1.0×10^6	50,000,896	50,000,896	50,000,896	50,000,896	100,001,792
Learning starts	10,000	100,000	100,000	100,000	100,000	100,000
Replay buffer size	1.0×10^6	1.0×10^7	1.0×10^7	1.0×10^7	1.0×10^7	1.0×10^7
Replay device	CPU	GPU	GPU	GPU	GPU	GPU
Batch size	512	2048	2048	2048	2048	2048
Grad. steps per interaction step	1	2	2	2	2	2
Discount factor γ	0.99	0.97	0.99	0.90	0.99	0.99
n-step return	1	1	3	1	3	3
SWD decay horizon	80,000	2,000	2,000	2,000	2,000	2,000
Compute type	float32	bfloat16	bfloat16	bfloat16	bfloat16	bfloat16
Evaluation episodes	50	50	50	50	1024	50
Evaluation environments	1	50	50	50	1024	50

Table 5: **Shared optimization and SAC hyperparameters.** These values are fixed unless a controlled ablation explicitly changes the corresponding component.

	Hyperparameter	Value
Optimization	Actor learning rate	3×10^{-4} with cosine decay to 1.5×10^{-4}
	Critic learning rate	3×10^{-4} with cosine decay to 1.5×10^{-4}
	Temperature learning rate	3×10^{-4} with cosine decay to 1.5×10^{-4}
Entropy	Initial entropy temperature	0.01
	Target entropy	$\frac{1}{2}\lvert\mathcal{A}\rvert \log(2\pi e \cdot 0.15^2)$
Updates	Target network update rate τ	0.01
	Target update frequency	Every critic update
	Policy update frequency	Every 2 critic updates
Common	Reward normalization	Enabled for all experiments
	Action repeat	1
	Linear-layer bias	Disabled
Critic	Critic ensemble size	2 for FlashSAC, **WarpSAC** Norm ON, and **WarpSAC** Norm OFF; 1 for Single-Q variants
	Critic output heads	101 categorical heads with cross-entropy distributional critic loss
Replay	SWD minimum sampling weight	0.1
	Uniform replay baseline	Implemented by setting the SWD decay horizon to 0

For the normalization ablations, **Norm ON** applies parameter projection to both actor and critic after initialization and after each gradient update. **Norm OFF** disables this projection while keeping all other training settings fixed. The scale ablations in Section 5 vary only the number of FlashSAC blocks in the actor and critic; the hidden dimensions, optimizer settings, replay settings, and environment budgets remain unchanged.

D.3 Network Architecture

All experiments use the same FlashSAC-style actor–critic backbone. The actor predicts a tanh-squashed Gaussian policy. The critic encodes the observation–action pair and predicts a distributional value head. Standard variants use two critics; Single-Q variants use one critic with the same per-critic architecture.

Table 6: **Actor and critic architecture.** The number of residual blocks is 2 in the main experiments and is varied in the capacity ablations.

	Component	Architecture
Actor	Input	Flattened actor observation; asymmetric environments use the actor observation stream for the policy.
	Encoder	Batch-normalized input projection to 128 hidden units, followed by B_π FlashSAC residual blocks and final RMSNorm.
	Residual block	Linear $128 \to 512$, BatchNorm, ReLU, Linear $512 \to 128$, BatchNorm, ReLU, residual connection.
	Output	Two orthogonally initialized linear heads predict action mean and log standard deviation. The policy samples from a tanh-squashed diagonal Gaussian and rescales actions to $[-1, 1]$.
	Log-standard-deviation range	$[-10, 2]$ after tanh-based squashing.
Critic	Input	Concatenated critic observation and action; asymmetric environments use the critic observation stream for value learning.
	Encoder	Batch-normalized input projection to 256 hidden units, followed by B_Q FlashSAC residual blocks and final RMSNorm.
	Residual block	Linear $256 \to 1024$, BatchNorm, ReLU, Linear $1024 \to 256$, BatchNorm, ReLU, residual connection.
	Output	Orthogonally initialized linear projection from 256 hidden units to 101 categorical value logits.
Variants	Main setting	$B_\pi = B_Q = 2$, with two critics for clipped double-Q learning.
	Capacity ablation	$B_\pi = B_Q \in \{1, 2, 3\}$ depending on the experiment.
	Single-Q variant	Same actor and per-critic network, but with one critic ensemble member and no clipped minimum over two critics.
	Parameter projection normalization	When enabled, all linear kernels are projected by column-wise norm normalization. BatchNorm affine scale and bias are jointly normalized, and RMSNorm scales are normalized.

E Environment Details

This section lists the environments used in the experiments. The benchmark suite contains IsaacLab, MuJoCo Playground, MJLab, ManiSkill, Gym–MuJoCo, DMC hard tasks, HumanoidBench, and MyoSuite. It does not include Genesis environments.

E.1 IsaacLab

We use IsaacLab v2.1.0 (Mittal et al., 2025). The suite includes 12 gripper manipulation, dexterous manipulation, quadruped locomotion, and humanoid locomotion tasks. Scores are normalized using near-asymptotic FlashSAC references.

Table 7: **IsaacLab environments.** Normalized scores correspond to near-asymptotic performance after extended training.

Task	Observation dim	Action dim	Normalize Score
`Isaac-Repose-Cube-Shadow-Direct-v0`	157	20	10000
`Isaac-Repose-Cube-Allegro-Direct-v0`	124	16	6000
`Isaac-Velocity-Flat-G1-v0`	123	37	40
`Isaac-Velocity-Rough-G1-v0`	310	37	40
`Isaac-Velocity-Flat-H1-v0`	69	19	40
`Isaac-Velocity-Rough-H1-v0`	256	19	40
`Isaac-Lift-Cube-Franka-v0`	36	8	160
`Isaac-Open-Drawer-Franka-v0`	31	8	100
`Isaac-Velocity-Flat-Anymal-C-v0`	48	12	30
`Isaac-Velocity-Rough-Anymal-C-v0`	235	12	30
`Isaac-Velocity-Flat-Anymal-D-v0`	48	12	30
`Isaac-Velocity-Rough-Anymal-D-v0`	235	12	30

E.2 MuJoCo Playground

We evaluate four humanoid locomotion tasks from MuJoCo Playground v0.0.5 (Zakka et al., 2025). Normalized scores are scaled to 40. Privileged critic observation dimensions are shown in parentheses when available.

Table 8: **MuJoCo Playground environments.** We evaluate four humanoid locomotion tasks from MuJoCo Playground. Normalized scores are scaled to 40, corresponding to asymptotic performance achieved after extended training. If the environment supports asymmetric observation, the privileged observation size is given in parentheses.

Task	Observation dim	Action dim	Normalize Score
`G1JoystickRoughTerrain`	103 (216)	29	40
`G1JoystickFlatTerrain`	103 (216)	29	40
`T1JoystickRoughTerrain`	85 (180)	23	40
`T1JoystickFlatTerrain`	85 (180)	23	40

E.3 MJLab

We use MJLab v1.2.0 (Zakka et al., 2026), an Isaac Lab-style robot-learning framework powered by the MuJoCo-Warp GPU simulator. We evaluate eight GPU-parallel tasks: four Unitree velocity-tracking locomotion tasks and four Yam manipulation tasks. Normalized scores are scaled to 100 for the locomotion tasks and 20 for the Yam manipulation tasks. Privileged critic observation dimensions are shown in parentheses when available.

Table 9: **MJLab environments.** We evaluate Unitree velocity-tracking and Yam manipulation tasks from MJLab. Observation dimensions report the actor input, with the effective critic input shown in parentheses. Normalized scores are scaled to 100 for locomotion tasks and 20 for manipulation tasks.

Task	Observation dim	Action dim	Normalize Score
`Mjlab-Velocity-Flat-Unitree-G1`	99 (210)	29	100
`Mjlab-Velocity-Rough-Unitree-G1`	286 (584)	29	100
`Mjlab-Velocity-Flat-Unitree-Go1`	48 (120)	12	100
`Mjlab-Velocity-Rough-Unitree-Go1`	235 (494)	12	100
`Mjlab-Lift-Cube-Yam`	29 (58)	7	20
`Mjlab-Lift-Cube-Yam-Depth`	26 (55)	7	20
`Mjlab-Lift-Cube-Yam-Rgb`	26 (55)	7	20
`Mjlab-Multi-Cube-Seg-Yam`	26 (52)	7	20

E.4 ManiSkill

We use ManiSkill (Tao et al., 2024), a large-scale benchmark built on the SAPIEN physics engine. We evaluate six rigid-body manipulation tasks with a gripper-based robotic arm. For reproducibility, we use the environment snapshot corresponding to commit hash `aad75f2`. Normalized scores correspond to the maximum success rate, where 100 denotes 100% success.

Table 10: **ManiSkill environments.** We evaluate 6 ManiSkill gripper-based manipulation environments. Normalized scores correspond to the maximum success rate, where 100 denotes a 100% success rate.

Task	Observation dim	Action dim	Normalize Score
`PickSingleYCB-v1`	45	8	100
`PegInsertionSide-v1`	43	8	100
`LiftPegUpright-v1`	32	8	100
`PokeCube-v1`	54	8	100
`PullCube-v1`	35	8	100
`RollBall-v1`	44	8	100

E.5 Gym–MuJoCo

We evaluate five Gym–MuJoCo v4 locomotion tasks (Todorov et al., 2012). Scores are normalized using random-policy lower references and TD7 5M-step upper references (Fujimoto & Gu, 2023).

Table 11: **Gym–MuJoCo environments.** Normalized scores are computed relative to random-policy and TD7 5M-step reference scores.

Task	Observation dim	Action dim	Random Score	Normalize Score
`HalfCheetah-v4`	17	6	-289.415	18165
`Hopper-v4`	11	3	18.791	4075
`Walker2d-v4`	17	6	2.791	7397
`Ant-v4`	27	8	-70.288	10133
`Humanoid-v4`	376	17	104.361	8584

E.6 DeepMind Control Suite Hard Tasks

We report only DMC hard tasks: humanoid and dog domains from the DeepMind Control Suite (Tassa et al., 2018). Low-dimensional tasks such as Cartpole and Pendulum are not included. Normalized scores use the DMC maximum of 1000.

Table 12: **DMC hard environments.** We report only humanoid and dog tasks, with normalized scores set to the DMC maximum.

Task	Observation dim	Action dim	Normalize Score
humanoid-stand	67	21	1000
humanoid-walk	67	21	1000
humanoid-run	67	21	1000
dog-stand	223	38	1000
dog-walk	223	38	1000
dog-run	223	38	1000
dog-trot	223	38	1000

E.7 HUMANOIDBENCH

We evaluate HumanoidBench (Sferrazza et al., 2024), a high-dimensional whole-body-control benchmark based on the Unitree H1 humanoid robot. In our experiments, HumanoidBench is installed from a compatibility fork. We evaluate 14 locomotion and whole-body-control tasks without hand control. Normalized scores follow the benchmark success thresholds.

Table 13: **HumanoidBench environments.** Scores are normalized using benchmark task-success thresholds.

Task	Observation dim	Action dim	Random Score	Normalize Score
h1-walk-v0	51	19	2.38	700
h1-stand-v0	51	19	10.55	800
h1-run-v0	51	19	2.02	700
h1-reach-v0	57	19	260.30	12000
h1-maze-v0	51	19	106.44	1200
h1-hurdle-v0	51	19	2.21	700
h1-crawl-v0	51	19	272.66	700
h1-sit_simple-v0	51	19	9.40	750
h1-sit_hard-v0	64	19	2.45	750
h1-balance_simple-v0	64	19	9.40	800
h1-balance_hard-v0	77	19	9.04	800
h1-stair-v0	51	19	3.11	700
h1-slide-v0	51	19	3.19	700
h1-pole-v0	51	19	20.09	700

E.8 MYOSUITE

We evaluate 10 MyoSuite dexterous manipulation tasks (Caggiano et al., 2022). Tasks marked `hard` use randomized goals. Normalized scores correspond to success rate, with 100 denoting 100% success.

Table 14: **MyoSuite environments.** Normalized scores correspond to 100% success.

Task	Observation dim	Action dim	Normalize Score
myo-reach	115	39	100
myo-reach-hard	115	39	100
myo-pose	108	39	100
myo-pose-hard	108	39	100
myo-obj-hold	91	39	100
myo-obj-hold-hard	91	39	100
myo-key-turn	93	39	100
myo-key-turn-hard	93	39	100
myo-pen-twirl	83	39	100
myo-pen-twirl-hard	83	39	100

E.9 SIM2REAL

For sim-to-real locomotion, we use the `Unitree-G1-Flat` implementation from Unitree Robotics. The task exposes asymmetric observations: the policy receives a 98-dimensional actor observation, while the critic receives a 211-dimensional privileged observation. The 29-dimensional action controls the Unitree G1 joints.

Table 15: **Sim2Real environment.** The `Unitree-G1-Flat` locomotion task uses asymmetric actor and critic observations for sim-to-real policy training.

Task	Actor obs. dim	Critic obs. dim	Action dim
Unitree-G1-Flat	98	211	29

F FULL RESULTS

F.1 GYM–MUJOCO

Figure 9 reports the complete Gym–MuJoCo learning curves. These tasks provide a standard CPU-scale locomotion benchmark for comparing final performance and sample efficiency under single-environment replay.

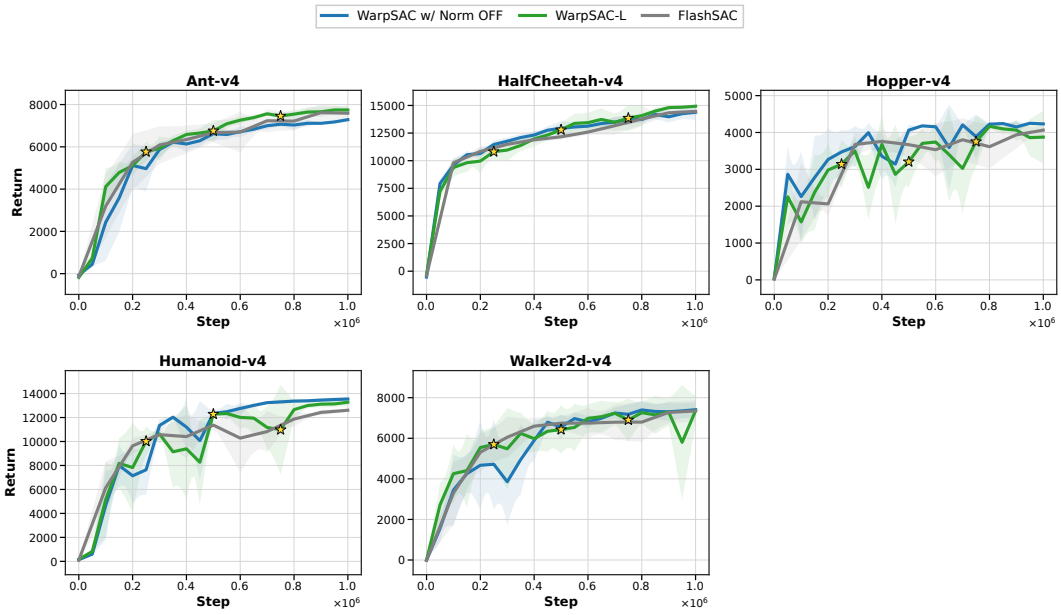

Figure 9: **Full Gym–MuJoCo learning curves.** Curves report mean performance with standard deviation across five seeds for all evaluated Gym–MuJoCo tasks.

F.2 DMC

Figure 10 reports the full DMC hard-task results on humanoid and dog domains. These tasks test whether replay weighting remains useful in CPU-scale control settings with diverse dynamics and reward scales.

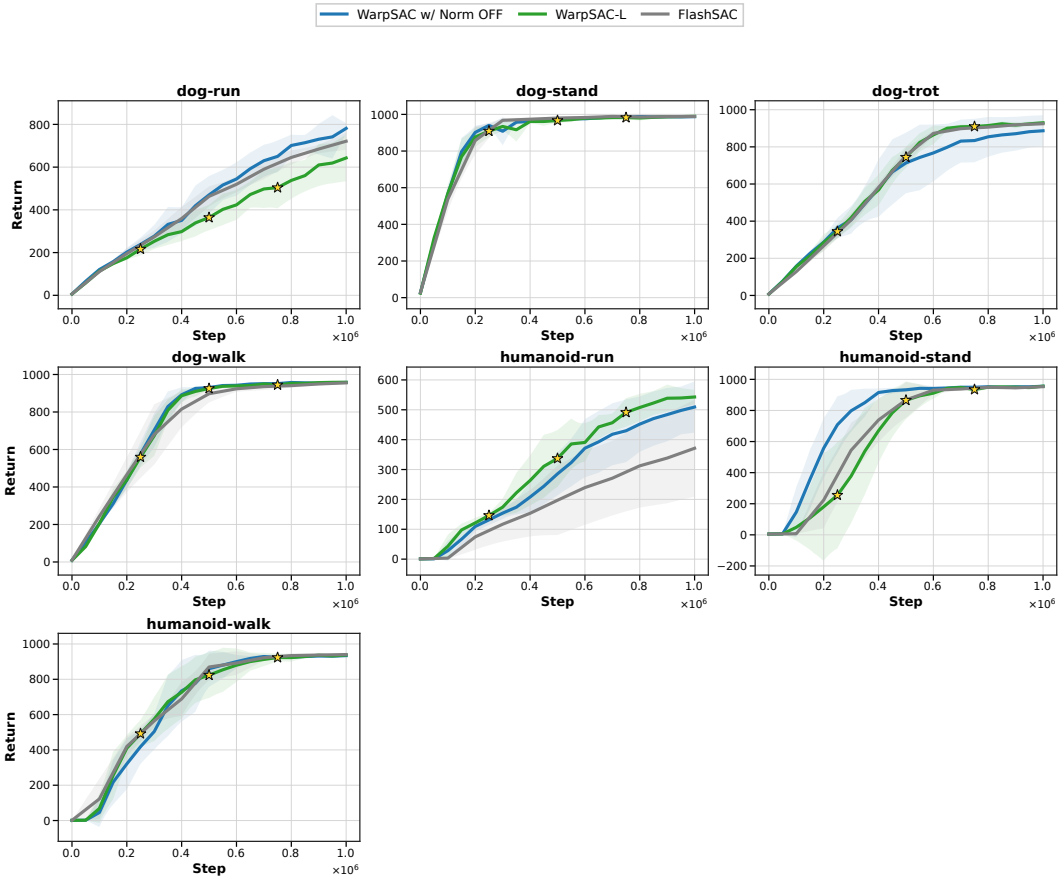

Figure 10: **Full DMC hard-task learning curves.** Curves report mean performance with standard deviation across seeds on humanoid and dog tasks from the DeepMind Control Suite.

F.3 MyoSuite

Figure 11 reports the dexterous manipulation results on MyoSuite. These environments use success-rate-style returns, so the curves emphasize how quickly each method reaches reliable task completion.

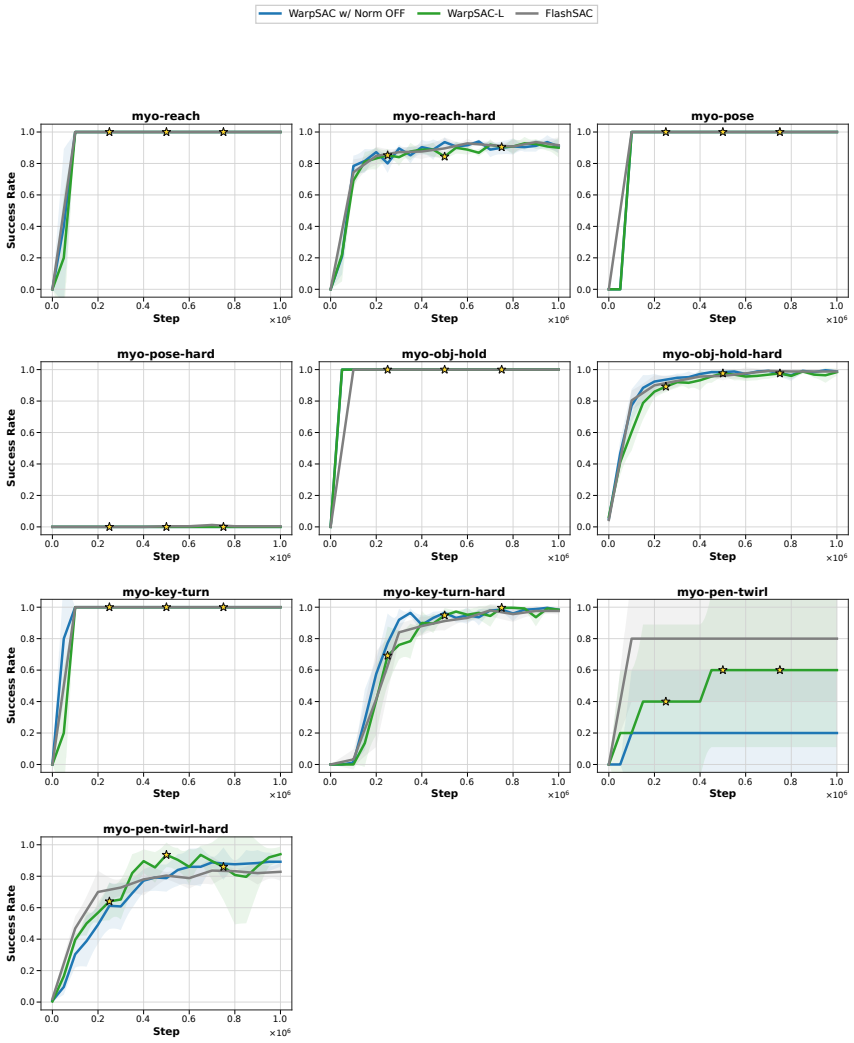

Figure 11: **Full MyoSuite learning curves.** Curves report mean success-oriented performance with standard deviation across five seeds for the evaluated MyoSuite manipulation tasks.

F.4 HUMANOIDBENCH

Figure 12 gives the full HumanoidBench results. These tasks are high-dimensional whole-body-control problems and provide a stress test for stability under sparse or delayed progress.

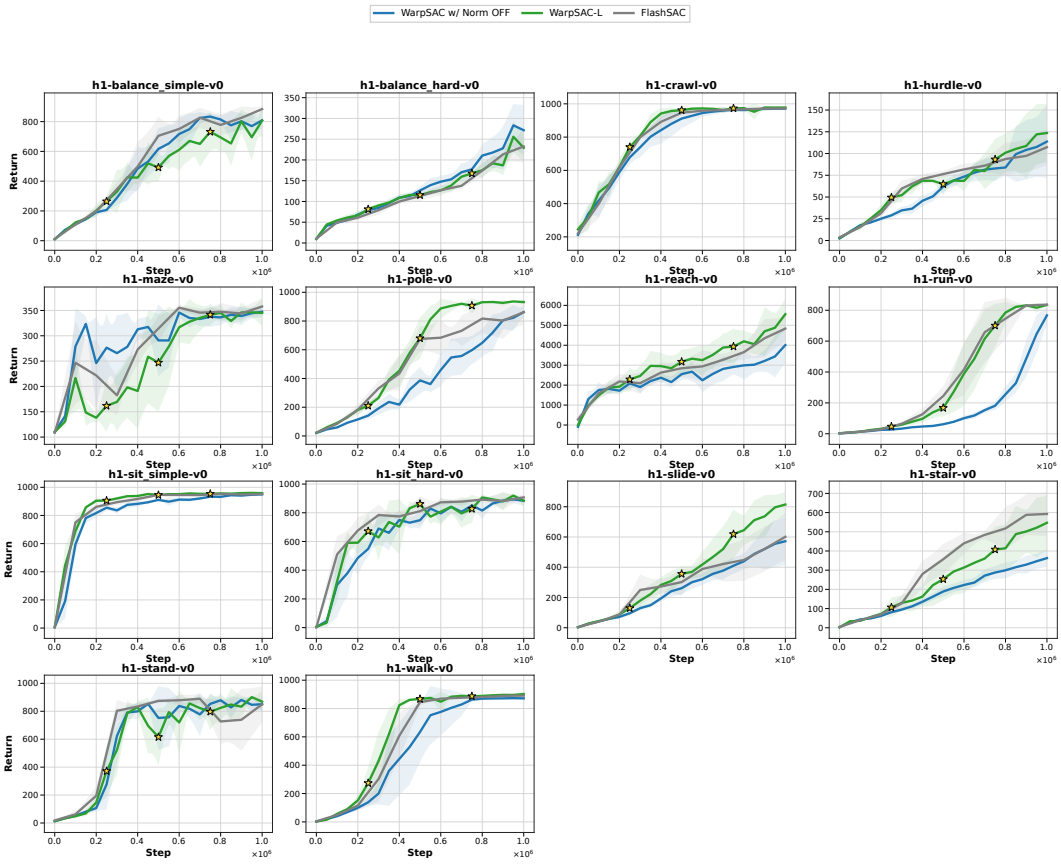

Figure 12: **Full HumanoidBench learning curves.** Curves report mean performance with standard deviation across five seeds for the evaluated H1 whole-body-control tasks.

F.5 MuJoCo Playground

Figure 13 reports the GPU-parallel MuJoCo Playground results. The large number of vectorized environments shifts the comparison toward how effectively each method exploits broad replay coverage.

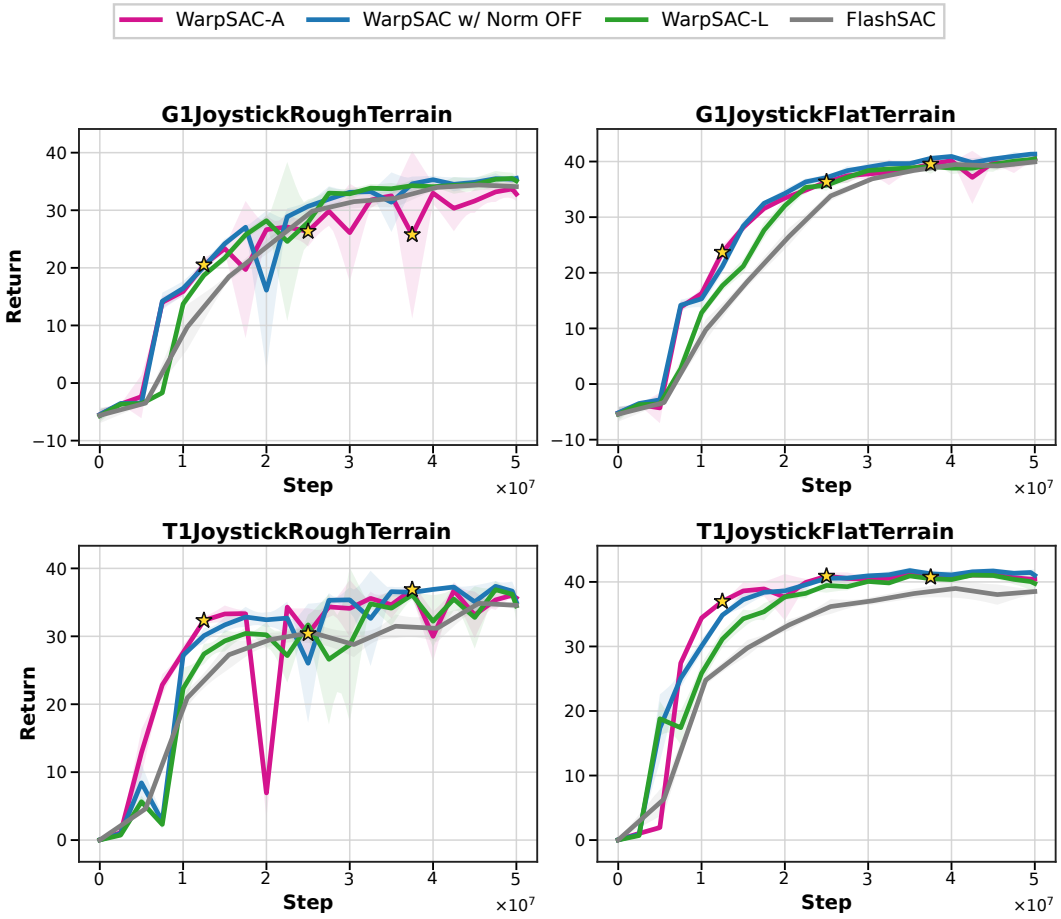

Figure 13: **Full MuJoCo Playground learning curves.** Curves report mean performance with standard deviation across five seeds for the evaluated GPU-parallel humanoid locomotion tasks.

F.6 ISAACLAB

Figure 14 reports the full IsaacLab results across manipulation and locomotion tasks. This domain evaluates performance under highly parallel simulation with diverse robot embodiments and reward scales.

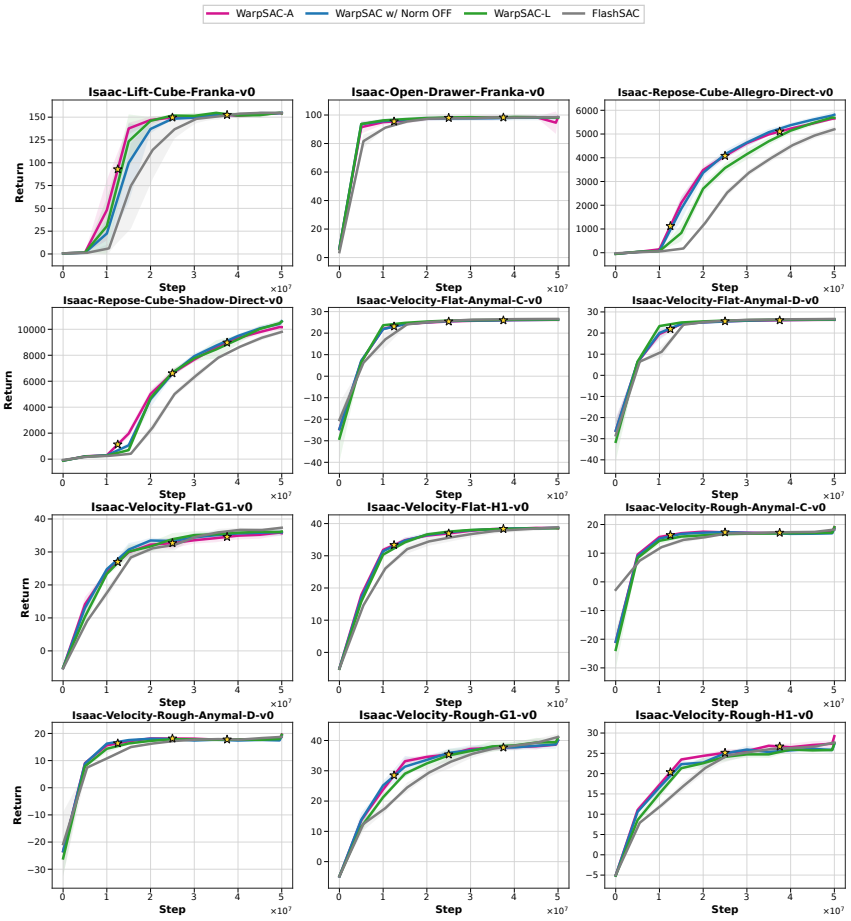

Figure 14: **Full IsaacLab learning curves.** Curves report mean performance with standard deviation across five seeds for the evaluated IsaacLab manipulation and locomotion tasks.

F.7 MJLAB

Figure 15 reports the complete MJLab results, covering Unitree G1 and Go1 velocity-tracking tasks on flat and rough terrain as well as Yam manipulation tasks with state, depth, RGB, and segmentation observations. These tasks provide an additional GPU-parallel backend spanning locomotion and manipulation, allowing us to test whether the observed data-regime effects extend beyond IsaacLab and MuJoCo Playground.

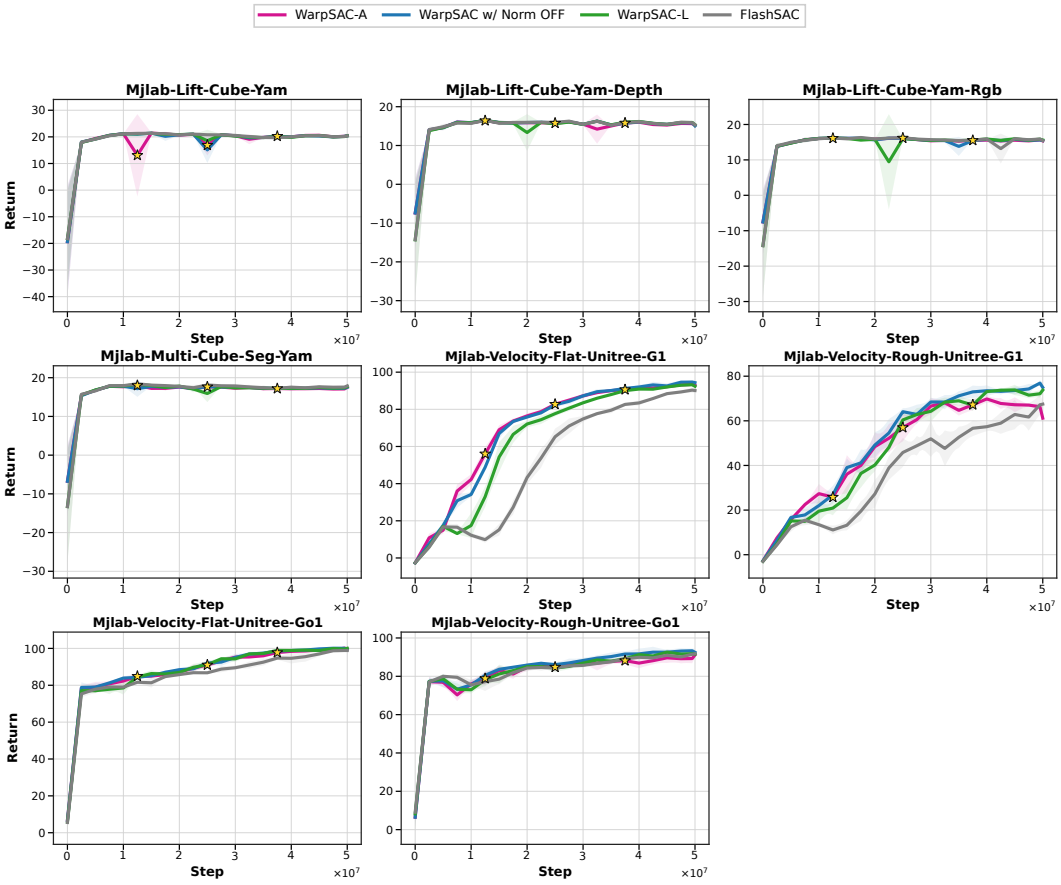

Figure 15: **Full MJLab learning curves.** Curves report mean episodic return with standard deviation across five seeds for Unitree locomotion and Yam manipulation tasks.

F.8 MANISKILL

Figure 16 reports the full ManiSkill results. These GPU-parallel manipulation tasks test whether the methods remain effective when success-rate objectives are collected from many simultaneous environments.

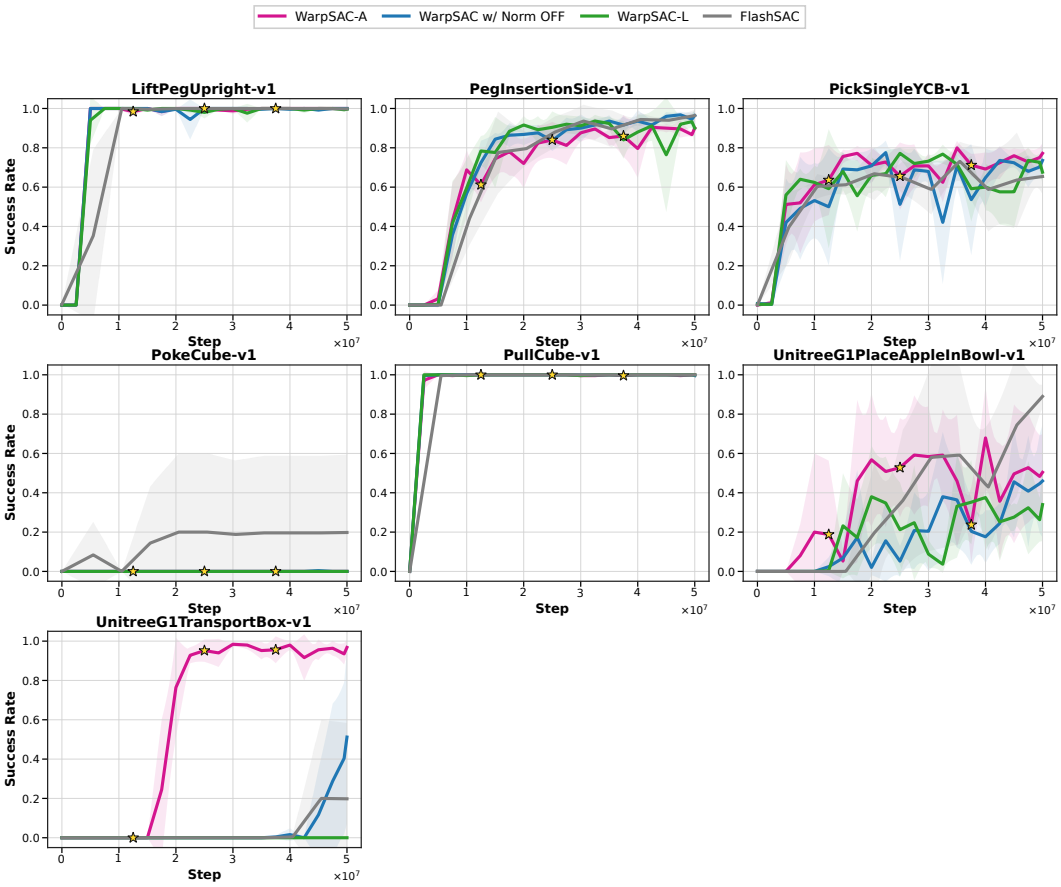

Figure 16: **Full ManiSkill learning curves.** Curves report mean success-oriented performance with standard deviation across five seeds for the evaluated ManiSkill manipulation tasks.

F.9 Ablation Learning Curves

The endpoint and AUC bar charts in the main paper summarize the controlled ablations. Figures 17 and 18 provide the corresponding learning curves, exposing how the capacity, replay-weighting, and normalization choices affect learning speed throughout training.

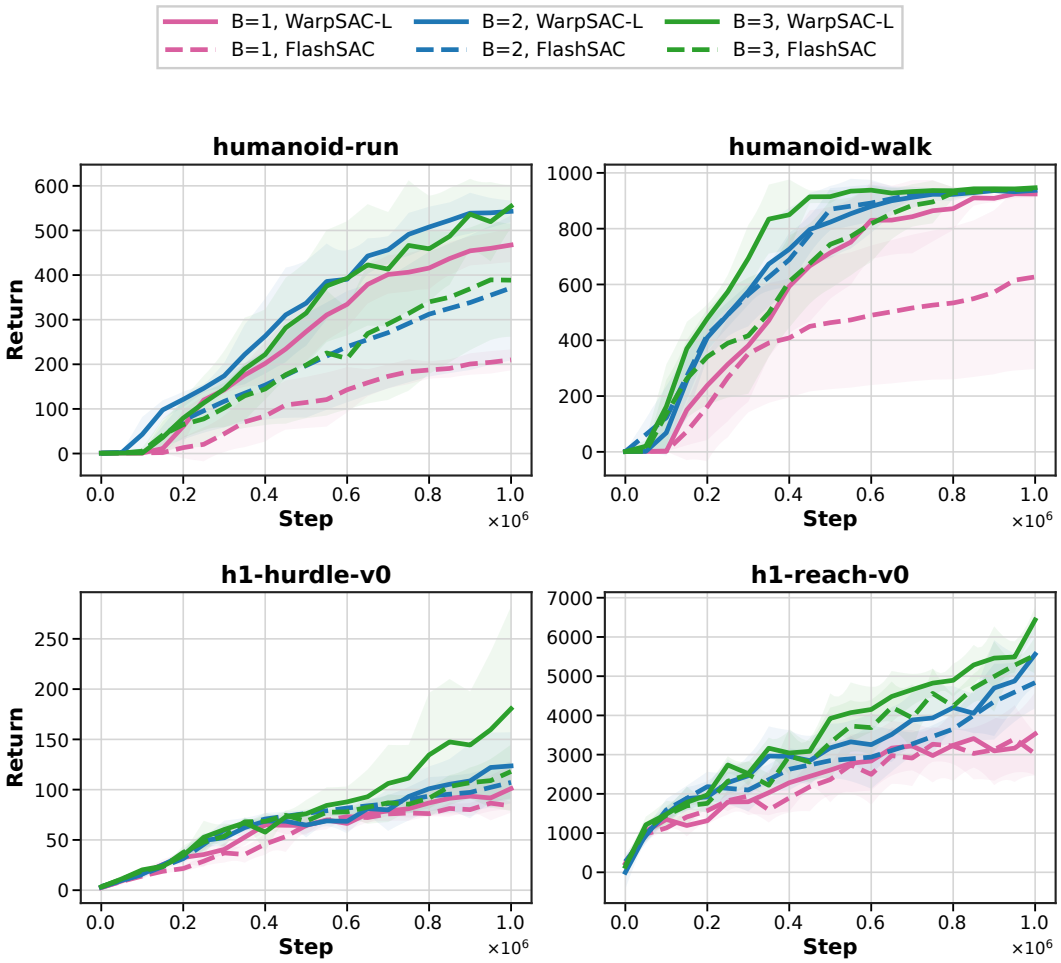

Figure 17: **Capacity and SWD ablation learning curves.** Curves compare SWD and uniform replay using one, two, and three residual blocks on humanoid-run, humanoid-walk, h1-hurdle-v0, and h1-reach-v0. Shaded regions show standard deviation across five seeds.

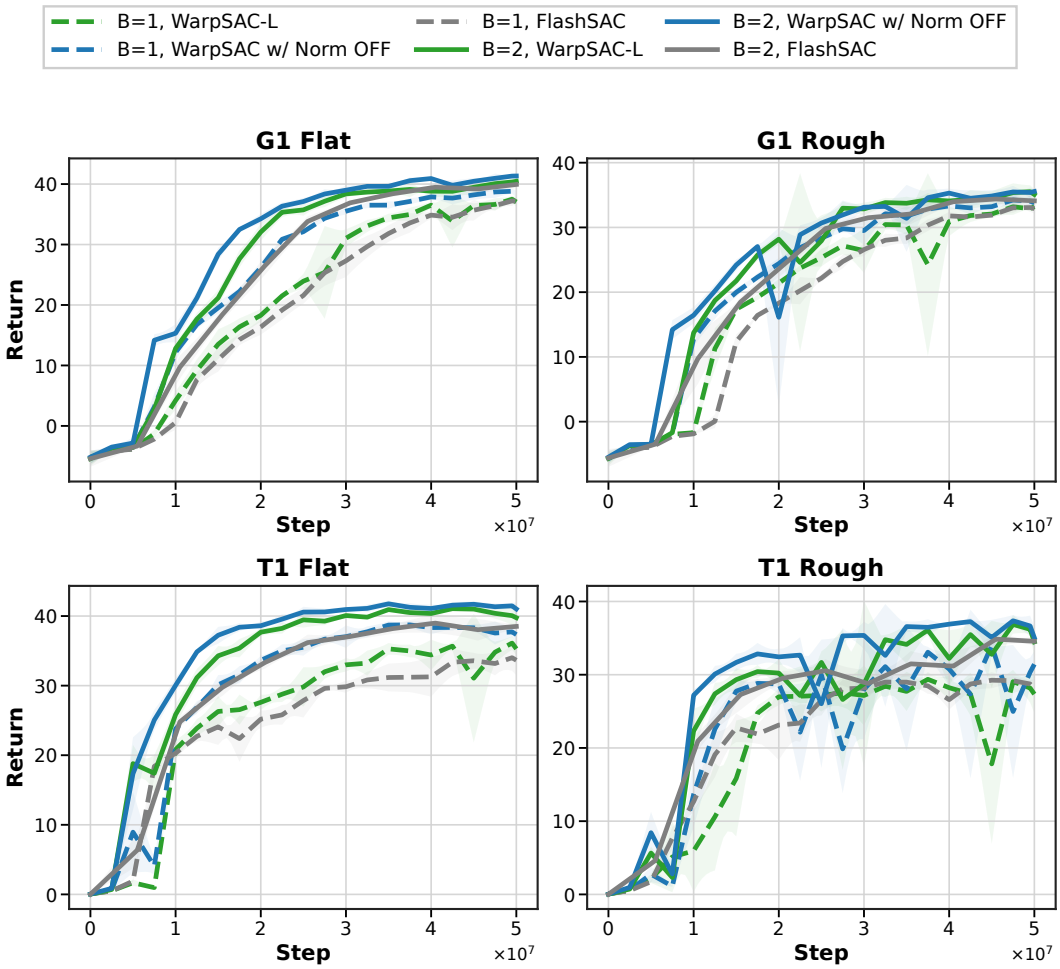

Figure 18: **MuJoCo Playground normalization and capacity ablation learning curves.** Curves compare Norm ON, Norm OFF, and No SWD using one and two residual blocks on the four evaluated Playground tasks. Shaded regions show standard deviation across five seeds.

F.10 Mechanism Analysis: SWD × Network Capacity in CPU-Scale

To isolate how replay weighting interacts with model capacity (Q4), Figure 19 varies the number of FlashSAC blocks on four CPU-scale locomotion tasks: humanoid-run, humanoid-walk, h1-hurdle-v0, and h1-reach-v0. SWD is most beneficial in the low-capacity regime: with a single block the relative gain over uniform replay exceeds $2\times$ on humanoid-run (from 209.93 to 467.39) and reaches $\sim$47% on humanoid-walk (from 627.07 to 924.39), $\sim$21% on h1-hurdle-v0 (from 83.80 to 101.09), and $\sim$17% on h1-reach-v0 (from 3018.97 to 3529.44). These gains suggest that age-biased replay can improve update efficiency by emphasizing more policy-relevant samples when the function approximator is capacity constrained.

As capacity increases, the gap narrows on saturated tasks such as humanoid-walk, while remaining visible on harder HumanoidBench tasks (h1-hurdle-v0, h1-reach-v0). This trend shows that SWD is not merely a substitute for additional parameters; it changes *how* available capacity is used. When the network is small, prioritizing recent samples provides a clear optimization advantage; when the network is large enough to fit broader replay distributions, the marginal benefit becomes task-dependent, though SWD remains competitive or better across all tested capacities.

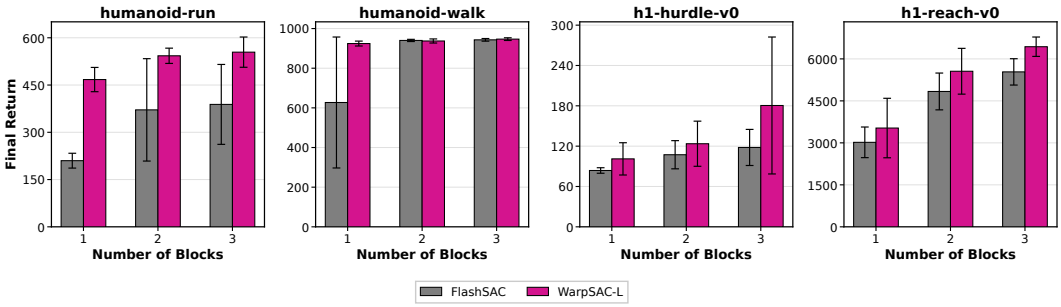

Figure 19: **Network-capacity comparison of WarpSAC-L and FlashSAC in CPU-scale training.** Final return of **WarpSAC**-L and FlashSAC as the number of residual blocks is varied on humanoid-run, humanoid-walk, h1-hurdle-v0, and h1-reach-v0. Both methods use Norm ON and clipped double-Q and differ only in replay weighting, isolating how the **WarpSAC** replay mechanism interacts with network capacity. With a single block, **WarpSAC**-L improves final return over FlashSAC across all the four tasks; as blocks increase, the gap narrows on saturated tasks but remains visible on harder HumanoidBench tasks.

F.11 Mechanism Analysis: Normalization × Network Capacity in GPU-Parallel

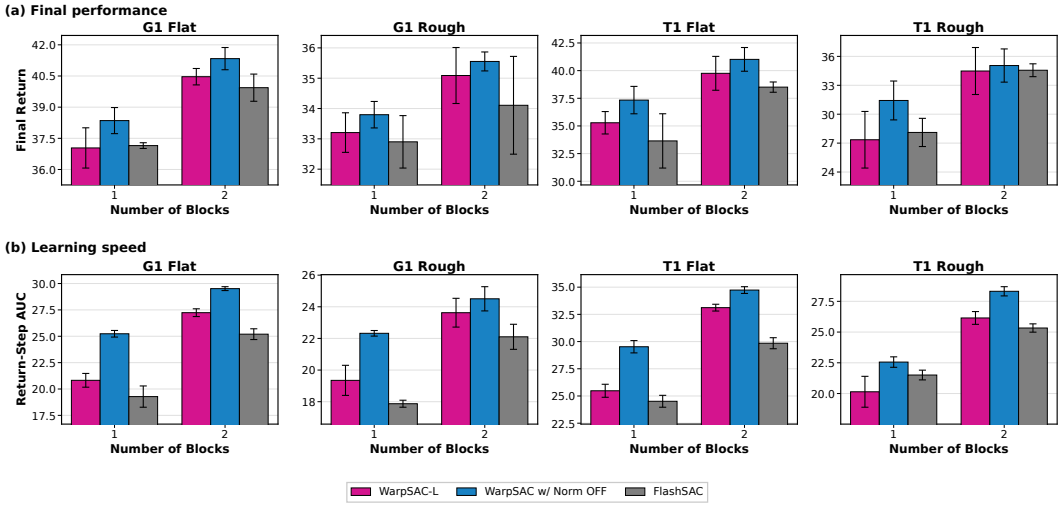

Figure 20: **Network-capacity comparison of WarpSAC variants and FlashSAC in GPU-parallel training.** Final return and return–step AUC (normalized by the common training horizon, reflecting average return throughout training) of **WarpSAC**-L, **WarpSAC** w/ Norm OFF, and FlashSAC as the number of residual blocks is varied on MuJoCo Playground tasks. At one block, **WarpSAC** w/ Norm OFF improves both final return and AUC markedly over **WarpSAC**-L and FlashSAC on G1 Flat and T1 Rough; with two blocks the gap narrows, but **WarpSAC** w/ Norm OFF remains competitive or better across all four tasks.

We study the interaction between SWD, parameter projection normalization, and model capacity in MuJoCo Playground (Figure 20), a massively parallel setting where replay data are generated rapidly and the bottleneck is fitting high-value behavior rather than obtaining sufficient exploration.

The results show that parameter projection normalization can sharply reduce performance when capacity is limited. With one FlashSAC block, disabling normalization improves performance markedly on G1 Flat and T1 Rough. With two blocks, the gap becomes smaller, but Norm OFF remains competitive or better across the Playground tasks. This supports our hypothesis that normalization is useful when data are scarce, but can restrict expressive freedom in large-parallel environments where state–action coverage is already broad.

Importantly, SWD alone is not always sufficient when normalization strongly constrains the function class: the best-performing configurations combine SWD with reduced normalization, confirming that replay-side exploitation and network expressivity must be balanced jointly in GPU-scale RL (Q4).

F.12 Sim-to-Real Details

Off-policy RL is often viewed as brittle for sim-to-real transfer, especially on high-dimensional humanoid systems where unstable value learning can quickly produce unsafe locomotion. We therefore include a sim-to-real case study on `Unitree-G1-Flat`, a 29-DoF Unitree G1 locomotion task with proprioceptive policy inputs from Unitree Robotics' pipeline. The task uses an asymmetric actor–critic formulation: the deployed policy receives a 98-dimensional actor observation, while the critic is trained with a 211-dimensional privileged observation. We keep the environment, reward design, observation interface, and sim-to-real adaptation pipeline fixed across methods, and compare **WarpSAC**, FlashSAC, and PPO under the same Unitree-G1-Flat training profile.

For PPO, we start from the `rsl_rl`-style on-policy pipeline commonly used in Unitree sim-to-real locomotion. The unmodified `rsl_rl` implementation does not include the mixed-precision and compilation path used by our scalable off-policy learners. To make the wall-clock comparison less dependent on an avoidably slow baseline implementation, we add bfloat16 autocasting and `torch.compile` to the PPO training path. Thus, PPO is a strengthened baseline rather than the vanilla `rsl_rl` code path.

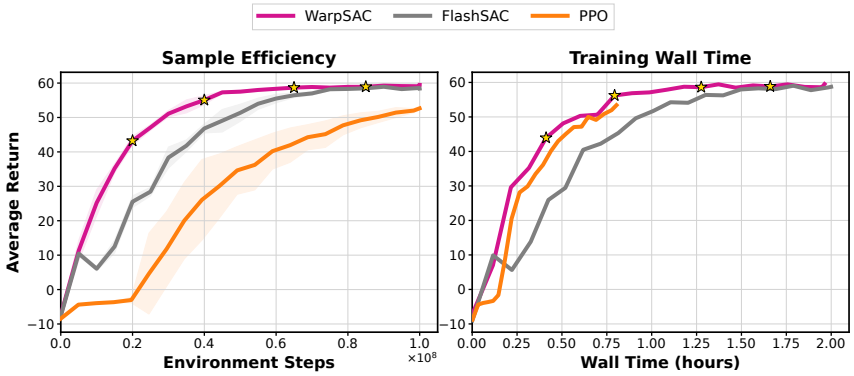

Figure 21: **Unitree G1 Sim-to-Real learning curves.** The left panel compares sample efficiency with standard-deviation shading. The right panel compares training wall time.

Figure 21 reports both sample-efficiency and wall-time learning curves, and Figure 1 also shows the screenshots of the gaits. On an end-to-end A800 run that includes simulation, replay, learner updates, logging, and evaluation, **WarpSAC** reaches the target performance in roughly **35 minutes**, whereas FlashSAC takes about **55 minutes** under the same sim-to-real setup. These results suggest that the replay-side exploitation benefits of **WarpSAC** transfer beyond simulator-only benchmarks, making scalable off-policy learning practical for high-dimensional humanoid sim-to-real locomotion when paired with a stable adaptation pipeline.