FLASHSAC: Fast and Stable Off-Policy Reinforcement Learning for High-Dimensional Robot Control

Author Names Omitted for Anonymous Review. Paper-ID 191

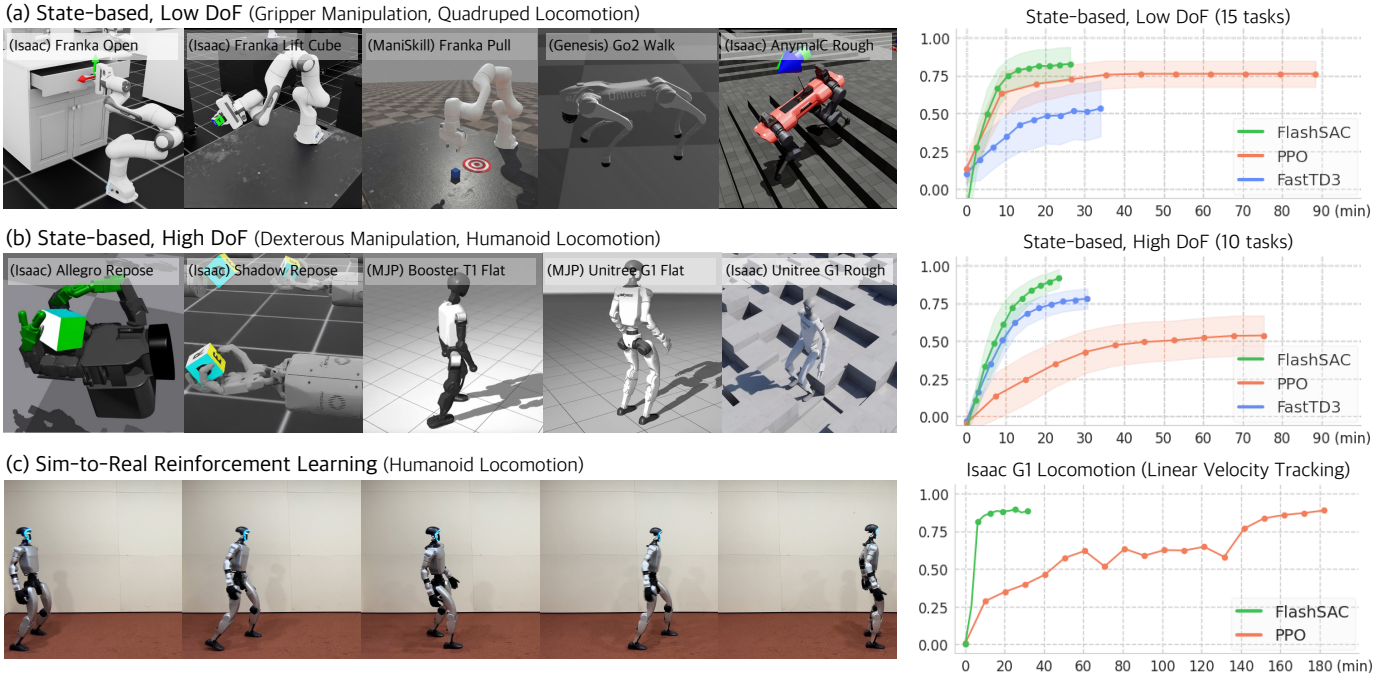

Fig. 1: **Results Overview.** Tasks grouped by state–action dimensionality, with representative examples shown for each category. **(a) State-based, Low DoF:** Gripper manipulation and quadruped locomotion tasks from IsaacLab, ManiSkill, and Genesis. In low-dimensional settings, FLASHSAC achieves performance comparable to PPO. **(b) State-based, High DoF:** Dexterous manipulation and humanoid locomotion tasks from IsaacLab and MuJoCo Playground. In high-dimensional settings, FLASHSAC substantially outperforms PPO in both asymptotic return and wall-clock efficiency. **(c) Sim-to-Real:** Humanoid locomotion on the Unitree G1 platform. FLASHSAC enables sim-to-real transfer within minutes, whereas PPO requires hours of training.

***Abstract*—Simulation-based reinforcement learning (RL) is central for robotic control when expert demonstrations are unavailable. However, scaling RL to high-dimensional robots remains challenging. On-policy methods such as PPO are reliable but require large amounts of simulation because they discard past data. Off-policy methods can reuse experience and are more sample-efficient, but they often become unstable in high-dimensional control due to critic errors that are amplified during bootstrapped updates. We introduce FLASHSAC, a fast and stable off-policy RL algorithm for high-dimensional robotic control. FLASHSAC improves training stability in two ways: (1) it explicitly bounds weight, feature, and gradient norms to limit critic error amplification, and (2) it increases data coverage through large-scale parallel simulation, a high-capacity replay buffer, and strong exploration. These design choices preserve the sample efficiency of off-policy learning while improving training stability. Across 50+ state-based and vision-based tasks in 10 simulators, FLASHSAC consistently surpasses PPO and strong off-policy baselines in both final performance and wall-clock efficiency, with larger gains on higher-dimensional tasks. In sim-to-real humanoid walking, FLASHSAC reduces training time from hours to minutes while maintaining stable real-world deployment. Our results show that stabilizing off-policy learning enables scalable sim-to-real RL for high-dimensional robotic systems.**

I. INTRODUCTION

The long-standing goal of robot learning is to develop agents that can generalize across a wide range of tasks in the real world. While large-scale imitation learning from real-world data has recently produced impressive progress in robotic control [36, 80], reinforcement learning (RL) from simulation remains a core paradigm when expert demonstrations are unavailable, incomplete, or insufficient [37, 3, 93].

To date, sim-to-real RL has been most successful in relatively constrained domains such as quadruped locomotion [30, 66] and gripper-based manipulation [2]. These tasks are characterized by low-dimensional state–action spaces and extremely high-throughput simulators [78, 54, 4, 91]. In this regime, on-policy methods such as Proximal Policy Optimization (PPO)

[70, 71] have proven effective: PPO is stable, easy to tune, and its data inefficiency is acceptable when fresh on-policy data can be collected cheaply.

However, the robotics community is rapidly moving beyond this regime. Emerging applications, such as humanoid locomotion [73], dexterous manipulation [12, 87], and vision-based control [59, 34], involve high-dimensional state–action spaces that dramatically increase the data requirements for learning reliable policies. At the same time, simulation becomes slower due to complex contact dynamics [46, 88], while large neural backbones further increase rollout cost [36, 80]. As a result, the high data demands and rising simulation costs render on-policy RL increasingly impractical for high-dimensional robotic tasks.

Off-policy RL offers a principled alternative. By reusing experience stored in a replay buffer, off-policy algorithms can achieve significantly higher data efficiency and improved wall-clock performance [55, 45, 23]. Despite these advantages, off-policy methods remain underused in robotics due to persistent training instability [18, 43].

This instability primarily stems from training critic in off-policy RL. The critic Q_θ is trained by minimizing a bootstrapped Bellman error,

$$\mathcal{L}_Q = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - (r + \gamma Q_\theta(s', a'))\right)^2\right], \quad (1)$$

where transitions, (s, a, r, s'), are sampled from a replay buffer $\mathcal{D}$ and $a' \sim \pi(\cdot \mid s')$. In the off-policy setting, this objective is difficult to optimize for two reasons [77, 84]. First, target values are evaluated at next state–action pairs (s', a') that may be poorly represented, or absent, from the replay buffer, leading to approximation and extrapolation errors [18]. Second, because the target depends on the critic's own predictions, these errors can be recursively amplified through bootstrapped updates before being corrected by new data. Both effects are exacerbated in high-dimensional tasks, where function approximation errors are more frequent and harder to rectify.

We introduce FLASHSAC, a fast and stable off-policy RL algorithm designed for high-dimensional robotic control. FLASHSAC addresses critics' training instability through two complementary mechanisms. First, it reduces approximation and extrapolation error by scaling data volume and diversity via large-scale parallel simulation, a high-capacity replay buffer, and strong exploration. Second, it limits Bellman error amplification by explicitly controlling critic update dynamics. By bounding weight, feature, and gradient norms throughout training, FLASHSAC prevents small estimation errors from being magnified through repeated bootstrapped updates. Together, these design choices enable stable and data-efficient RL for modern high-dimensional robotic tasks.

We evaluate FLASHSAC on over 50 tasks across 10 simulators, spanning high-dimensional state-based control, vision-based control, and sim-to-real humanoid locomotion. Across this diverse benchmark suite, FLASHSAC consistently outperforms PPO and strong off-policy baselines in both final performance and wall-clock efficiency. While performance is comparable to PPO in low-dimensional tasks, the advantage of FLASHSAC increases with task dimensionality. In sim-to-real humanoid walking, FLASHSAC reduces training time from 3 hours with PPO to 20 minutes while maintaining stable real-world deployment. These results demonstrate that stabilizing off-policy learning enables scalable and efficient training for high-dimensional robotic control, positioning FLASHSAC as a practical foundation for sim-to-real robot learning.

II. Related Work

A. On-Policy Reinforcement Learning

On-policy RL has been the dominant paradigm for simulation-based robot learning when environment interaction is cheap and can be massively parallelized. Among on-policy methods, PPO [70] is particularly popular due to its stability, ease of implementation, and robustness to hyperparameter choices. Combined with modern high-throughput simulators [52, 78, 54, 4], PPO has enabled successful sim-to-real transfer in relatively constrained domains such as quadruped locomotion [30, 66, 71] and rigid-body manipulation [2].

Despite these successes, on-policy methods fundamentally rely on collecting fresh data from the current policy and discarding data generated by earlier policies [77]. As task dimensionality increases, the state–action space grows rapidly, making sufficient coverage via on-policy data collection increasingly expensive and often impractical in high-dimensional robotic tasks.

In principle, on-policy algorithms can reuse past experience through importance sampling to correct for policy mismatch [74, 51]. In practice, however, importance weights in high-dimensional continuous action spaces exhibit extremely high variance, leading to unstable updates. As a result, importance-sampling-based data reuse is rarely effective in large-scale robotic learning, and on-policy methods remain limited in data efficiency.

B. Off-Policy Reinforcement Learning

Off-policy RL addresses these limitations by decoupling data collection from policy optimization. Instead of discarding experience after each policy update, off-policy algorithms store transitions collected under previous policies in a replay buffer and reuse them for learning [77, 55]. This reuse enables broader coverage of the state–action space and significantly improves data efficiency, making off-policy RL attractive for high-dimensional robotic control. Off-policy methods are broadly categorized into model-based and model-free approaches.

Off-Policy Model-Based RL: Model-based RL improves sample efficiency by learning an explicit or latent model of environment dynamics and using it for planning or imagined rollouts [76, 32]. Recent approaches such as DreamerV3 [24] and TD-MPC2 [25] have demonstrated strong performance in vision-based domains by planning in learned latent spaces.

However, these methods typically incur substantial computational overhead. Learning accurate dynamics models and performing repeated planning or rollout procedures significantly increases training cost per environment step [50]. In simulation settings where wall-clock efficiency is critical, this overhead

often limits scalability, making model-based methods less practical for large-scale robotic training.

Off-Policy Model-Free RL: Model-free off-policy algorithms, such as DDPG [45], TD3 [18], and SAC [23], learn policies and value functions directly from replayed experience without explicit dynamics models. Their simplicity and ability to reuse data make them attractive for robotic control.

Despite these advantages, off-policy methods remain underused in robotics due to persistent training instability [9, 43]. As discussed in § I, this instability arises from optimizing the bootstrapped Bellman objective in Equation 1 using function approximation and off-policy data [77, 84]. Target values are evaluated at next state–action pairs that may be poorly represented or absent in the replay buffer, leading to approximation and extrapolation errors. Because these targets depend on the value function's own predictions, such errors can be amplified through repeated bootstrapped updates. These effects are exacerbated in high-dimensional robotic tasks, where large state–action spaces increase extrapolation error and deep neural networks further magnify small estimation errors [10, 40].

Prior work has explored two main strategies to improve stability. One line of work emphasizes data scaling, using parallel simulation and large replay buffers to reduce approximation error and improve wall-clock efficiency. Methods such as FastTD3 [72] achieve strong performance in humanoid locomotion, but typically rely on relatively small networks (on the order of 0.2M parameters), as increasing model capacity tends to exacerbate instability.

A second line of work stabilizes off-policy learning by explicitly constraining value-function sensitivity, for example by bounding feature, weight, or gradient norms [8, 21, 60, 40, 41, 49, 61, 62]. These constraints limit error amplification under distribution shift and repeated bootstrapped updates, enabling training with larger networks. However, strong regularization can restrict representation capacity and slow learning in data-rich simulation regimes.

FLASHSAC builds on insights from both directions. It combines large-scale data collection for wall-clock efficiency with explicit control of update dynamics to prevent error amplification when training large, deep value functions. This integration enables fast, stable, and high-performance off-policy learning for high-dimensional robotic control.

III. PRELIMINARY

In this section, we introduce the RL framework and algorithmic foundation upon which FLASHSAC is built.

A. Markov Decision Process (MDP)

We model robotic control as a discounted Markov Decision Process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the continuous action space, $P(s'|s,a)$ denotes the transition dynamics, $r(s,a)$ denotes the reward function, and $\gamma \in [0,1)$ is the discount factor.

At each timestep t, the agent observes $s_t \in \mathcal{S}$, samples an action $a_t \in \mathcal{A}$, receives a reward $r_t = r(s_t, a_t)$, and transitions to the next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. The goal is to learn a policy $\pi(a|s)$ that maximizes the discounted sum of rewards.

B. Soft Actor-Critic (SAC)

FLASHSAC builds upon SAC [23], a widely used off-policy RL algorithm. SAC stores transitions (s, a, r, s') collected under past policies in a replay buffer $\mathcal{D}$, and trains the policy using samples drawn from this buffer.

Beyond maximizing expected return, SAC incorporates an entropy regularization term that encourages exploration. This entropy maximization is particularly important in high-dimensional state–action spaces, where insufficient exploration can lead to poor coverage of the replay buffer and exacerbate approximation and extrapolation errors.

To reduce approximation errors in bootstrapped value learning, SAC commonly employs clipped double Q-learning [18], maintaining two action-value functions $Q_{\phi_1}(s,a)$ and $Q_{\phi_2}(s,a)$. The minimum of the two estimates is used when forming targets, reducing the impact of optimistic value errors.

Concretely, the policy $\pi_\theta(a|s)$ is optimized by minimizing

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\theta}\big(\alpha \log \pi_\theta(a|s) - \min_{i=1,2} Q_{\phi_i}(s,a)\big), \quad (2)$$

where $\alpha > 0$ controls the relative importance of entropy.

Each critic is trained by minimizing a bootstrapped Bellman error using slowly updated target networks $\bar{\phi}_1$ and $\bar{\phi}_2$, which are updated via exponential moving average:

$$\bar{\phi}_j \leftarrow \tau\phi_j + (1-\tau)\bar{\phi}_j,\ j \in \{1,2\} \quad (3)$$

where $\tau \in (0,1)$ is the target update rate.

For $i \in \{1,2\}$, the critic weights ϕ_i are optimized by minimizing the Bellman loss

$$\mathcal{L}_Q(\phi_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[Q_{\phi_i}(s,a) - y\right]^2, \quad (4)$$

where the target value is

$$y = r + \gamma\big(\min_{j=1,2} Q_{\bar{\phi}_j}(s',a') - \alpha \log \pi_\theta(a'|s')\big), a' \sim \pi_\theta(\cdot|s'). \quad (5)$$

IV. FLASHSAC

FLASHSAC is a fast and stable off-policy RL algorithm designed for high-dimensional robotic control. FLASHSAC achieves strong asymptotic performance with fast wall-clock time through three complementary mechanisms: (i) scaling data volume and diversity (§IV-A), (ii) explicitly controlling critic update dynamics (§IV-B), and (iii) scaling model size for faster training (§IV-C).

A. Scaling Data Volume and Diversity

In off-policy RL, critics are trained using bootstrapped targets evaluated at next-state action pairs induced by the current policy. When these target state–action pairs are poorly represented in the replay buffer, approximation and extrapolation errors arise and can be recursively amplified through repeated Bellman updates. This problem becomes more pronounced as state–action dimensionality increases.

FLASHSAC reduces these errors by aggressively scaling both the volume and diversity of data, thereby improving coverage of the state–action space.

1) Massively Parallel Simulation: We collect data using 1024 parallel simulation environments, enabling rapid accumulation of diverse trajectories. While many off-policy RL setups rely on a small number of environments [23, 18], high-throughput data collection is critical for maintaining adequate coverage of the state–action space in high-dimensional tasks.

2) Large-Capacity Replay Buffer: FLASHSAC employs a replay buffer with capacity up to 10M transitions by default, substantially larger than the 1M transitions commonly used in standard off-policy configurations [40, 62]. In high-dimensional robotic tasks, rare but semantically important state–action pairs can be easily overwritten in smaller buffers, leading to catastrophic forgetting and inducing the extrapolation error. A large-capacity replay buffer preserves long-tail experiences and improves the diversity of training data available to the critic throughout learning [16].

3) Exploration: Because off-policy RL decouples data collection from policy optimization, FLASHSAC employs aggressive exploration strategies that target broad data coverage.

Unified Entropy Target: We adopt maximum-entropy RL with automatic temperature tuning [23] to encourage sustained exploration. However, choosing a task-agnostic entropy target is challenging when action dimensionality varies across robotic embodiments. To address this, we parameterize the entropy target using a fixed target action standard deviation σ_{tgt}, which provides a consistent and interpretable exploration scale.

For a Gaussian policy with diagonal covariance, the target entropy is defined as:

$$\bar{\mathcal{H}} = \frac{1}{2}|\mathcal{A}| \log \left(2\pi e \sigma_{\text{tgt}}^2\right), \tag{6}$$

which scales linearly with the action dimension. This ensures comparable exploration behavior across tasks with different robotic embodiments. We use $\sigma_{\text{tgt}} = 0.15$ in all experiments.

Noise Repetition: Temporally correlated action noise is commonly used to improve exploration in sparse-reward settings, with pink noise [15] and Ornstein–Uhlenbeck noise [28] being widely used. However, these methods are ill-suited to massively parallel simulation, as they require per-environment correlated noise processes, incurring substantial computational and memory overhead.

We propose *Noise Repetition*, a lightweight alternative that induces temporal correlation using minimal local state. At each repetition interval, a noise vector $\epsilon \sim \mathcal{N}(0, I)$ is sampled for action selection and held constant for k consecutive steps. The repetition length k is drawn from a Zeta distribution with probability mass function $P(k) \propto k^{-s}$ [13], favoring short repeat intervals while occasionally producing long, temporally correlated action sequences.

B. Constraining Update Dynamics

While scaling data reduces approximation error, it does not fully prevent instability arising from bootstrapped critic updates. In high-dimensional settings, even small estimation errors can be recursively amplified through repeated Bellman backups. Thus, FLASHSAC constrains update dynamics by bounding weight, feature, and gradient norms throughout training.

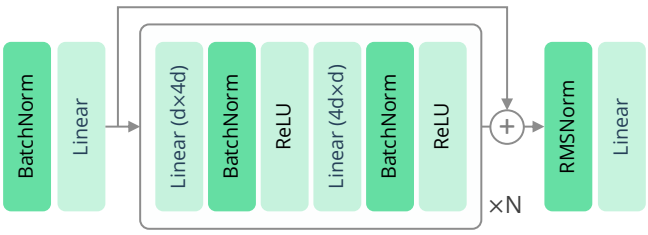

Fig. 2: **FLASHSAC Architecture.** The architecture consists of stacked inverted residual blocks with pre-activation batch normalization and post-RMS normalization.

1) Architecture: FLASHSAC 's architecture is designed to promote consistent gradient propagation while constraining feature norms (Figure 2).

Inverted Residual Block: The backbone consists of a stack of inverted residual blocks inspired by the feedforward block in Transformer [86]. Each block uses an inverted bottleneck design [29], expanding features to a higher-dimension and then projecting them back to the original dimension, using a residual connection [26] to stabilize gradient propagation.

Pre-activation Normalization: Replay data is generated by a mixture of evolving policies, which induce non-stationary input distributions. Without normalization, feature activations can saturate (e.g., dead ReLUs [1]), leading to degraded gradient propagation [14, 48]. To address this, we apply normalization before each nonlinearity to keep feature activations well-scaled prior to nonlinear transformation.

Batch Normalization: We adopt batch normalization [31] rather than layer normalization [5], as it leverages large-batch statistics from diverse replay data [68] and yields a smoother loss landscape with a lower effective condition number [8, 62], reducing variance in gradients and feature norms.

Post RMS Normalization: While batch normalization stabilizes features dimension-wise, it does not explicitly bound per-sample feature norms. Rare or out-of-distribution inputs can still produce large activations that destabilize bootstrapping. Thus, we apply RMSNorm [92] after the final residual block to bound the feature norm before the policy and value heads.

2) Training: Beyond architecture, we stabilize training by aligning normalization statistics, smoothing critic optimization, controlling return scale, and constraining weight norm.

Cross-Batch Value Prediction: When using batch normalization, Bellman updates can suffer from mismatched normalization statistics between current Q-value estimates and target Q-value. Following [8], we concatenate current and next transitions into a single batch and compute both predicted and target Q-values using shared normalization statistics, ensuring consistency across the Bellman update.

Distributional Critic: Following [41, 62], we represent the Q-value as a categorical distribution over n_{atom} atoms uniformly spaced on $[G_{\min}, G_{\max}]$. The network predicts atom probabilities and is trained via cross-entropy loss against the projected Bellman target [7]. This distributional formulation smooths the optimization landscape and reduces sensitivity to noisy targets [62].

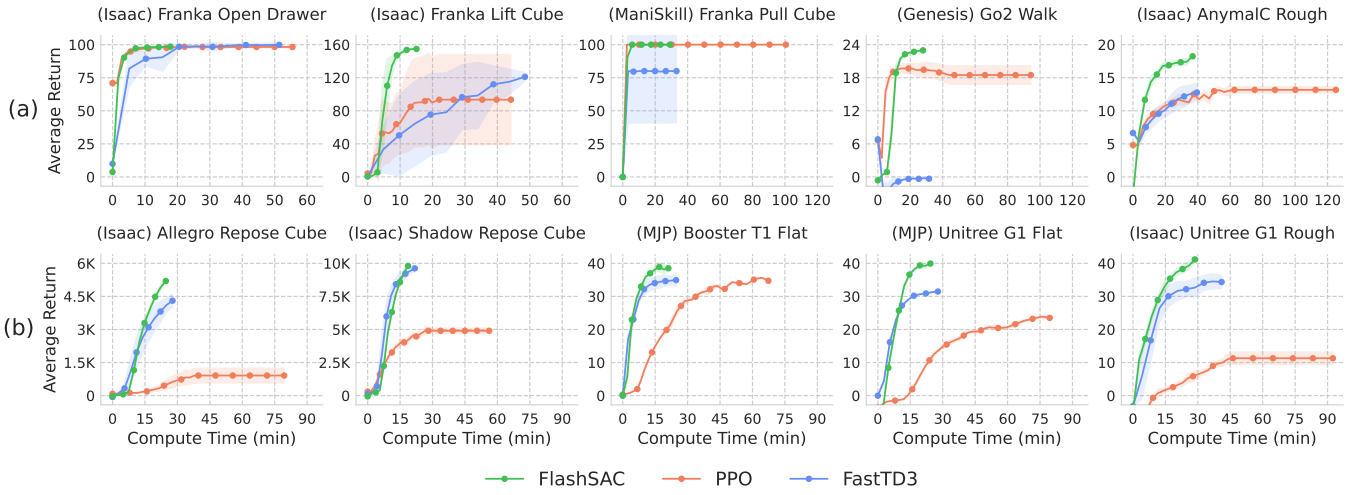

Fig. 3: **Results on State-Based RL.** Learning curves on select tasks from IsaacLab [54], ManiSkill [78], Genesis [4], and MuJoCo Playground [91]. We evaluate performance on **(a)** low-dimensional tasks with gripper manipulation and quadruped locomotion, and **(b)** high-dimensional tasks involving dexterous manipulation and humanoid locomotion. While FLASHSAC is comparable to PPO [71] in low-dimensional tasks, FLASHSAC significantly outperforms PPO in high-dimensional tasks.

Adaptive Reward Scaling: To stabilize gradients and keep returns within the distributional critic's fixed support, we apply adaptive reward scaling. Unlike reward centering [58] or loss scaling [69], we normalize the reward itself, preserving the optimal policy while maintaining a consistent scale. Specifically, we track the running discounted return G_t, its variance $\sigma^2_{t,G}$, and maximum magnitude $G_{t,\max}$, and scaled reward as:

$$\bar{r}_t = \frac{r_t}{\max\left(\sqrt{\sigma^2_{t,G} + \epsilon},\ G_{t,\max}/G_{\max}\right)}. \tag{7}$$

This bounds effective returns while maintaining a consistent scale throughout training.

Weight Normalization: Uncontrolled weight growth increases Q-value variance and exacerbates error amplification under bootstrapping [49]. After each gradient update, we project each weight vector onto the unit-norm sphere [85, 47, 41, 61, 62]. We additionally project normalization parameters (γ, β) to a fixed norm of $\sqrt{d}$, encouraging the network to encode information through direction rather than scale.

C. Scaling Model for Faster Training

Having stabilized off-policy learning, we focus on scaling the model to achieve high performance with short time.

In supervised learning, scaling laws indicate that under a fixed compute budget, larger models trained with larger batches and fewer parameter updates converge faster than smaller models trained with frequent updates [35]. Applying this principle to off-policy RL has been challenging due to critic instability under increased model capacity. By stabilizing critic training through data diversity and constrained updates, FLASHSAC enables this scaling strategy.

1) Large Model, Large Batch, Fewer Update: Most off-policy RL baselines use small MLPs (0.2–0.5M parameters, 2–3 layers) to maintain stability [23, 72]. In contrast, FLASHSAC employs a 2.5M-parameter, 6-layer network for both the actor and critic. This capacity is sufficient for our benchmarks, which require learning a single behavior per task.

We use a batch size of 2048, which nearly saturates GPU utilization in our setup, and set an updates-to-data ratio of $2/1024$ (i.e., 2 gradient update per 1024 newly collected transitions). While such infrequent updates are typically ineffective in off-policy RL, FLASHSAC compensates with large batches, higher learning rates, and increased model capacity, enabling fast convergence with fewer updates.

2) Code Optimization: FLASHSAC is implemented in JAX [17], with both training and inference JIT-compiled to minimize Python overhead. Since many baselines are implemented in PyTorch [63], this choice yields a practical speedup; in our experiments, switching to JAX reduced runtime by up to 20%.

V. EXPERIMENTS

We evaluate FLASHSAC across a diverse suite of robotic control tasks, assessing both asymptotic performance and wall-clock efficiency. All reported wall-clock times are measured using a single RTX 5090 GPU. Our experiments span low- and high-dimensional state-based control, sim-to-real humanoid locomotion, and vision-based control.

A. State-Based RL

Tasks: We benchmark 25 state-based control environments spanning a wide range of state–action dimensionalities:

- *Low-dim (15 tasks):* Gripper-based manipulation (Franka) and quadruped locomotion (AnyMal-C/D, Unitree Go2).
- *High-dim (10 tasks):* Dexterous manipulation (Allegro, Shadow Hand) and humanoid locomotion (Unitree G1, H1, Booster T1).

To ensure robustness across physics engines, tasks are drawn from four widely used simulators: IsaacLab [54], MuJoCo

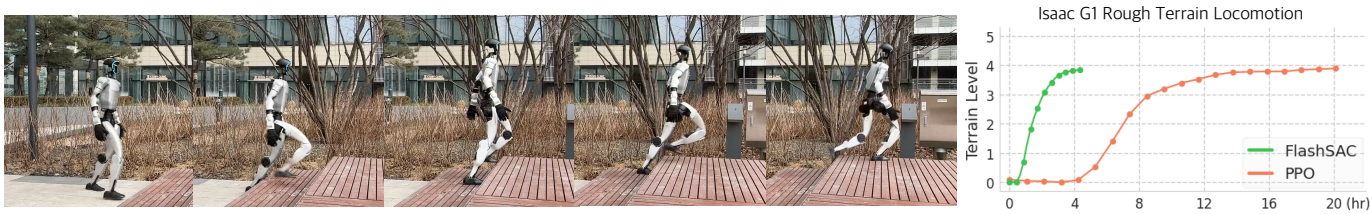

Fig. 4: **Sim-to-real Stair Climbing on Unitree G1.** FLASHSAC achieves stable real-world stair climbing after only 4 hours of training on simulation, whereas PPO requires nearly 20 hours to reach the same capability.

Playground [91], ManiSkill3 [78], and Genesis [4]. A complete task list is provided in § X.

Baselines: We compare FLASHSAC against strong, widely adopted baselines:

- *PPO* [71]: A highly optimized on-policy implementation from RSL-RL, representative of current best practices in sim-to-real robotic RL.
- *FastTD3* [72]: A wall-clock–optimized off-policy method designed for high throughput simulations.

When available, we report published results; otherwise, we reproduce results using official implementations and recommended hyperparameters.

Experimental Setup: Off-policy methods (FLASHSAC and FastTD3) are trained for 50M environment steps. To probe asymptotic performance, PPO is trained for 200M steps, requiring approximately $3\times$ the compute of FLASHSAC . While baseline methods use task-specific hyperparameter tuning, FLASHSAC is evaluated using a single unified configuration across all tasks, varying only the discount factor γ to match simulator defaults (e.g., 0.99 for IsaacLab, 0.97 for Playground).

Results: Figure 3 summarizes performance on representative tasks, with full results in § XII.

In low-dimensional tasks, FLASHSAC achieves performance comparable to PPO (Figure 3.a), consistent with prior work showing that on-policy methods remain effective when state–action spaces are small and simulation throughput is high.

In high-dimensional tasks, FLASHSAC demonstrates a clear and consistent advantage (Figure 3.b). Across dexterous manipulation and humanoid locomotion benchmarks, FLASHSAC converges reliably to higher asymptotic performance while requiring substantially less wall-clock time than PPO.

Compared to FastTD3, FLASHSAC is markedly more stable, converging across all tasks where FastTD3 frequently fails or underperforms (e.g., Go2Walk, Franka Pull Cube). When both methods converge, FLASHSAC achieves higher asymptotic performance, with the largest gains observed in humanoid locomotion, where larger model capacity is particularly beneficial.

Sample-Efficient RL: We further evaluate FLASHSAC on standard single-environment, CPU-based continuous-control benchmarks, comparing against sample-efficient off-policy methods including XQC [62], SimbaV2 [41] and TD-MPC2 [25]. Experiments cover 40 tasks from MuJoCo [81], DeepMind Control Suite [79], MyoSuite [11], and Humanoid-Bench [73]. In this sample-efficient regime, FLASHSAC consistently outperforms these baselines (§ VIII).

B. Sim-to-Real Transfer

Off-policy RL is often regarded as unreliable for sim-to-real transfer, particularly in high-dimensional systems, where training instability can lead to unsafe behaviors [56]. We evaluate whether FLASHSAC enables reliable sim-to-real transfer on a challenging 29-DoF Unitree G1 humanoid performing blind locomotion.

Experimental Setup: We train blind locomotion policies in simulation using a terrain curriculum comprising pyramid stairs, discrete grids, waves, and pits. Terrain difficulty is increased automatically using a game-inspired curriculum [67] once the policy traverses 50% of the environment.

As a baseline, we use PPO with the sim-to-real pipeline of [57]. FLASHSAC adopts the same sim-to-real adaptation techniques for a fair comparison. Both methods use implicit system identification via a context estimator [57] and an asymmetric actor–critic formulation [64], where the critic receives privileged information (e.g., contact states and height maps) during training.

Both methods share identical reward design and coefficients, combining velocity tracking with regularization terms penalizing foot slip, excessive torque, action discontinuities, and orientation instability. FLASHSAC uses the same architecture and hyperparameters as in the state-based experiments V-A. Additional details are provided in § XI.

Results: On flat terrain (Figure 1.(c)), FLASHSAC achieves stable real-world locomotion after approximately 20 minutes of training, whereas PPO requires about 3 hours to reach comparable performance. The learned policy supports omnidirectional locomotion (forward, backward, and lateral) without task-specific re-training or command curricula.

The advantage of FLASHSAC is more pronounced on rough terrain (Figure 4). FLASHSAC successfully climbs unseen, real-world stairs after approximately 4 hours of training. PPO requires nearly 20 hours to achieve a similar capability.

Overall, FLASHSAC reduces the training time for sim-to-real humanoid locomotion by nearly an order of magnitude compared to PPO while maintaining stable and safe behaviors.

C. Vision-based RL

We extend our evaluation to vision-based control, where learning is constrained by high rendering cost and low environment throughput. In contrast to state-based RL, visual environments provide orders-of-magnitude fewer transitions per unit time, making data efficiency critical.

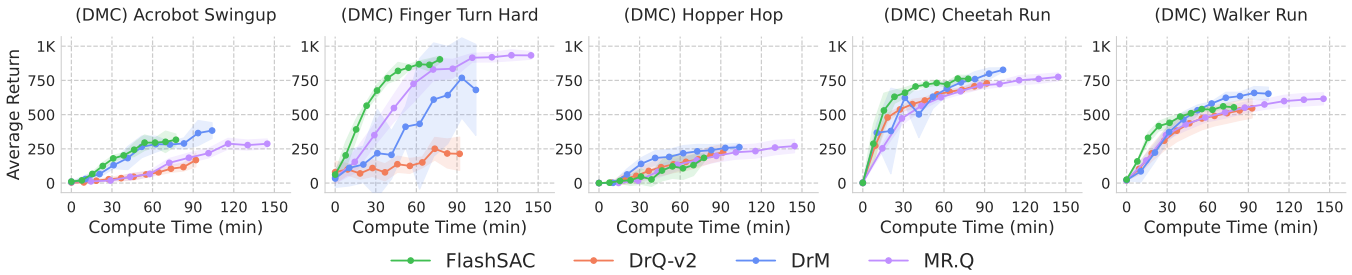

Fig. 5: **Results on Vision-Based RL.** Learning curves on selected tasks from vision-based DMControl Suite [79]. We assess from-scratch learning performance in low-dimensional environments, including pendulum manipulation and bipedal locomotion. FLASHSAC achieves superior convergence rate while matching the final performance of baselines.

Tasks: We evaluate on 8 environments from the DMC [83], spanning manipulation, and mono- bi-pedal locomotion tasks.

Baselines: Given the low data efficiency of visual RL, we focus on off-policy baselines:

- *DrQ-v2* [90]: A DDPG-based [45] method that improves data efficiency through image augmentation [39].
- *DrM* [89]: An extension of DrQ-v2 that augments exploration using a neuron-activity heuristic, encouraging exploration when network units are inactive.
- *MR.Q* [20]: An off-policy method with dynamics modeling objective to improve representation learning.

Experimental Setup: To adapt FLASHSAC to visual observations, we use the lightweight convolutional encoder from DrQ-v2, consisting of 3 convolutional layers followed by a linear bottleneck. All methods are trained for 1M environment steps with action repeat 2 and 3-step returns. Observations are provided as a stack of the three most recent frames ($84\times84\times9$), enabling temporal reasoning without recurrent architectures. Full details are in § X.

Results: Figure 5 shows representative learning curves; full results are provided in § XII.

Across tasks, FLASHSAC matches or exceeds asymptotic performance while converging faster in wall-clock time. MR.Q achieves strong final returns but incurs additional computational cost from model-based learning. DrQ-v2 is efficient but unstable, failing to converge in several environments (e.g., Finger Turn Hard). DrM attains strong performance but depends on environment-specific exploration heuristics, whereas FLASHSAC uses a single unified hyperparameter.

Overall, these results suggest that stabilizing off-policy learning is sufficient for efficient and robust visual control, and is complementary to exploration or model-based extensions.

VI. ANALYSIS

FLASHSAC has demonstrated strong performance throughout the previous sections. Here, we analyze the factors underlying this performance, focusing on differences in data diversity between off-policy and on-policy data, as well as the effects of architectural design choices and key hyperparameters.

Experiments are conducted in 4 IsaacLab environments [54], including cube reorientation with the Allegro and Shadow Hands, and flat and rough terrain locomotion with the G1.

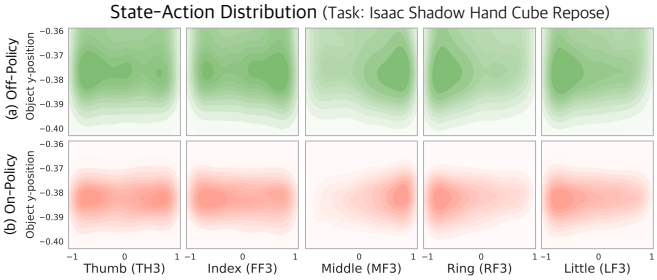

Fig. 6: **On- vs. Off-Policy Data Coverage.** We first train FLASHSAC for 1M steps on the Shadow Hand cube reorientation task. Off-policy samples are taken from the replay buffer, while on-policy samples are collected by rolling out the final policy for an additional 1M steps. We visualize the joint distribution of object y-position and fingertip joint actions using 2D density plots. Off-policy data covers a substantially broader region of the state–action space than on-policy data.

A. On-Policy vs Off-Policy

Setup: We train FLASHSAC for 1M steps on the IsaacLab Shadow Hand task with a replay buffer of size 1M, then collect 1M additional on-policy transitions using the final policy. We compare the state–action coverage of the two datasets.

Results: Figure 6 shows 2D density plots of state–action pairs, with finger actions on the x-axis and object y-position on the y-axis. Off-policy data covers a much broader region of the state–action space, reflecting experience accumulated from multiple behavior policies in the replay buffer. In contrast, on-policy data is tightly concentrated around the final policy distribution. This limited coverage helps explain why on-policy methods are less effective in high-dimensional tasks: achieving similar coverage would require collecting substantially more on-policy data to enable reliable policy learning.

B. Component Ablation

We analyze the contribution of each architectural component in FLASHSAC (§IV-B1) to determine whether the proposed design stabilizes critic training.

Beyond final task performance, we measure parameter, feature, and gradient norms throughout training as indicators of optimization stability. Following [62], we also measure the

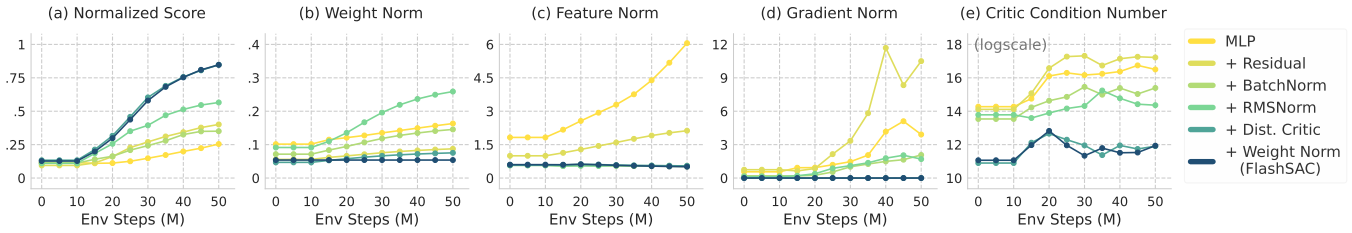

Fig. 7: **Component Ablation Results on IsaacLab.** We start from a standard MLP and incrementally build up to FlashSAC. Each component stabilizes training by constraining weight, feature and gradient norms while minimizing the condition number.

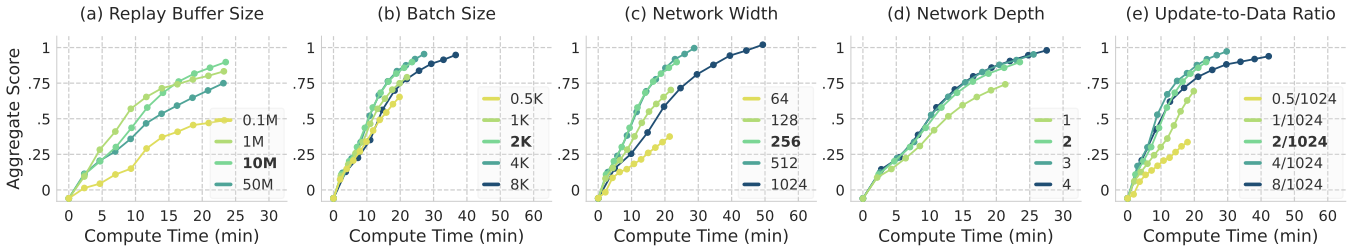

Fig. 8: **Hyperparameter Ablation Results on IsaacLab. (a)** Replay buffer size trades off training stability and efficiency. **(b–e)** Scaling batch size and model capacity and reducing the UTD ratio accelerate convergence.

condition number of the critic loss landscape, where larger values correspond to poorly conditioned updates that can exacerbate critic error amplification.

Setup: We conduct architectural ablations over the following components: *Residual Blocks*, *Batch Normalization*, *Post RMSNorm*, *Distributional Critics*, and *Weight Normalization*. Starting from a standard MLP critic, we incrementally add each component until reaching the full FlashSAC architecture.

Results: Figure 7 summarizes the results. As components are added, parameter, feature, and gradient norms stabilize throughout training, with no uncontrolled growth, indicating well-behaved critic updates and reduced error amplification. The critic's condition number also decreases monotonically, reaching its lowest value with the full FlashSAC architecture.

These gains in optimization stability directly translate to improved task performance (Figure 7.(a)), highlighting the importance of controlling critic update dynamics in off-policy RL. While weight normalization alone yields modest gains, it improves robustness in sample-limited regimes and is therefore retained in the final design.

C. Hyperparameter Ablation

We study how scaling key hyperparameters (§ IV-C) affects the compute efficiency of FlashSAC.

Setup: We perform univariate ablations over five hyperparameters: batch size, replay buffer size, network width, network depth, and update-to-data (UTD) ratio. Each hyperparameter is varied while all others are fixed at their default values.

Results: Figure 8 shows learning curves plotted against wall-clock time. Figure 8.(a) examines the effect of replay buffer size. Increasing the buffer size improves performance up to 10M transitions by stabilizing training. However, overly large buffers (e.g., 50M) slow learning because recent high-quality samples are drawn less frequently. Nevertheless, a 50M buffer can achieve slightly higher asymptotic performance given sufficiently long training.

More generally, Figures 8.(b)–(e) exhibit trends consistent with established scaling laws [35]: increasing batch size and model capacity, along with reducing the update-to-data (UTD) ratio, accelerate convergence. Most existing off-policy RL methods rely on small network architectures for training stability (e.g., width 128 with inverted bottlenecks and block depth 1, corresponding to four layers), which in turn limits convergence speed. In contrast, the scaling mechanisms of FlashSAC enable the use of higher-capacity models, resulting in substantially faster convergence.

VII. Lessons and Opportunities

As the robotics community moves toward high dimensional [22], perception-rich [59, 34], and contact-intensive tasks [88], the scalability of on-policy RL becomes increasingly constrained. In this regime, off-policy RL is an appealing alternative, but its adoption in the community has been limited by instability in critic learning, arising from function approximation error and bootstrapped updates. FlashSAC addresses this challenge by combining large and diverse replay data with explicit constraints on critic updates, enabling stable training, strong asymptotic performance, and substantially faster training than on-policy methods.

Stabilized off-policy learning opens new opportunities for robot learning. Improved data efficiency makes it feasible to train larger policies, incorporate vision and other rich sensory inputs [75], and use slower but more realistic simulators [65]. Off-policy methods also naturally support learning from a mixture of demonstrations and self-collected experience [6]. While this work focuses on state-based control, extending these critic-stabilization principles to vision- and tactile-based learning is a promising direction for future research.

References

[1] Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. In *Conference on lifelong learning agents*, pages 620–636. PMLR, 2023.

[2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[4] Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.

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

[6] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[7] 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.

[8] 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. *International Conference on Learning Representations (ICLR)*, 2024.

[9] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Is high variance unavoidable in rl? a case study in continuous control. *arXiv preprint arXiv:2110.11222*, 2021.

[10] 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.

[11] Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. Myosuite–a contact-rich simulation suite for musculoskeletal motor control. *arXiv preprint arXiv:2205.13600*, 2022.

[12] Yuanpei Chen, Yaodong Yang, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen Marcus McAleer, Hao Dong, and Song-Chun Zhu. Towards human-level bimanual dexterous manipulation with reinforcement learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=D29JbExncTP.

[13] Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended {\epsilon}-greedy exploration. *arXiv preprint arXiv:2006.01782*, 2020.

[14] Shibhansh Dohare, J Fernando Hernandez-Garcia, Parash Rahman, Richard S Sutton, and A Rupam Mahmood. Maintaining plasticity in deep continual learning. *arXiv preprint arXiv:2306.13812*, 2023.

[15] Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[16] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International conference on machine learning*, pages 3061–3071. PMLR, 2020.

[17] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *SysML conference 2018*, 2019.

[18] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[19] Scott Fujimoto, Wei-Di Chang, Edward J Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For sale: State-action representation learning for deep reinforcement learning. *arXiv preprint arXiv:2306.02451*, 2023.

[20] Scott Fujimoto, Pierluca D'Oro, Amy Zhang, Yuandong Tian, and Michael Rabbat. Towards general-purpose model-free reinforcement learning. *arXiv preprint arXiv:2501.16142*, 2025.

[21] Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.

[22] Zhaoyuan Gu, Junheng Li, Wenlan Shen, Wenhao Yu, Zhaoming Xie, Stephen McCrory, Xianyi Cheng, Abdulaziz Shamsah, Robert Griffin, C Karen Liu, et al. Humanoid locomotion and manipulation: Current progress and challenges in control, planning, and learning. *arXiv preprint arXiv:2501.02116*, 2025.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[24] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[25] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision*

and pattern recognition, pages 770–778, 2016.

[27] David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024.

[28] Jakob Hollenstein, Sayantan Auddy, Matteo Saveriano, Erwan Renaudo, and Justus Piater. Action noise in off-policy deep reinforcement learning: Impact on exploration and performance. *arXiv preprint arXiv:2206.03787*, 2022.

[29] Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[30] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[32] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

[33] Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *IEEE Robotics and Automation Letters*, 7(2):4630–4637, April 2022. ISSN 2377-3774. doi: 10.1109/lra.2022.3151396. URL http://dx.doi.org/10.1109/LRA.2022.3151396.

[34] Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Jim Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16923–16930. IEEE, 2025.

[35] 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.

[36] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2025.

[37] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[38] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[39] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33:19884–19895, 2020.

[40] 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. *arXiv preprint arXiv:2410.09754*, 2024.

[41] Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. *arXiv preprint arXiv:2502.15280*, 2025.

[42] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47), October 2020. ISSN 2470-9476. doi: 10.1126/scirobotics.abc5986. URL http://dx.doi.org/10.1126/scirobotics.abc5986.

[43] Qiyang Li, Aviral Kumar, Ilya Kostrikov, and Sergey Levine. Efficient deep reinforcement learning requires regulating overfitting. *arXiv preprint arXiv:2304.10466*, 2023.

[44] Qiayuan Liao, Takara E Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C Karen Liu. Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion. *arXiv preprint arXiv:2508.08241*, 2025.

[45] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[46] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 432–448. PMLR, 2021.

[47] Ilya Loshchilov, Cheng-Ping Hsieh, Simeng Sun, and Boris Ginsburg. ngpt: Normalized transformer with representation learning on the hypersphere. *arXiv preprint arXiv:2410.01131*, 2024.

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

[49] Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado P van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:106440–106473, 2024.

[50] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118, 2023.

[51] A Rupam Mahmood, Hado P Van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. *Advances in neural information processing systems*, 27, 2014.

[52] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[53] Mayank Mittal, Nikita Rudin, Victor Klemm, Arthur Allshire, and Marco Hutter. Symmetry considerations for learning task symmetric robot policies. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7433–7439. IEEE, 2024.

[54] Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

[55] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[56] J.W. Mock and University of Wyoming. Department of Electrical Engineering. *A Comparison of PPO, TD3, and SAC Reinforcement Algorithms for Quadruped Walking Gait Generation and Transfer Learning to a Physical Robot*. University of Wyoming, 2023. ISBN 9798379561789. URL https://books.google.co.kr/books?id=waUG0AEACAAJ.

[57] I Nahrendra, Byeongho Yu, and Hyun Myung. Dreamwaq: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning. *arXiv preprint arXiv:2301.10602*, 2023.

[58] Abhishek Naik, Yi Wan, Manan Tomar, and Richard S Sutton. Reward centering. *arXiv preprint arXiv:2405.09999*, 2024.

[59] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems*, 2024.

[60] Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. *arXiv preprint arXiv:2405.16158*, 2024.

[61] Daniel Palenicek, Florian Vogt, Joe Watson, and Jan Peters. Scaling off-policy reinforcement learning with batch and weight normalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[62] Daniel Palenicek, Florian Vogt, Joe Watson, Ingmar Posner, and Jan Peters. XQC: Well-conditioned optimization accelerates deep reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2026.

[63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[64] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[65] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.

[66] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on robot learning*, pages 91–100. PMLR, 2022.

[67] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on robot learning*, pages 91–100. PMLR, 2022.

[68] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[69] Tom Schaul, Georg Ostrovski, Iurii Kemaev, and Diana Borsa. Return-based scaling: Yet another normalisation trick for deep rl. *arXiv preprint arXiv:2105.05347*, 2021.

[70] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[71] Clemens Schwarke, Mayank Mittal, Nikita Rudin, David Hoeller, and Marco Hutter. Rsl-rl: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025.

[72] Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.

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

[74] Christian Robert Shelton. Importance sampling for reinforcement learning with multiple objectives. *PhD thesis, Massachusetts Institute of Technology*, 2001.

[75] Entong Su, Chengzhe Jia, Yuzhe Qin, Wenxuan Zhou, Annabella Macaluso, Binghao Huang, and Xiaolong Wang. Sim2real manipulation on unknown objects with tactile-based reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9234–9241. IEEE, 2024.

[76] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[77] Richard S Sutton, Andrew G Barto, et al. *Reinforce-*

ment learning: An introduction, volume 1. MIT press Cambridge, 1998.

[78] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[79] 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.

[80] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[81] 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*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[82] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

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

[84] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[85] Twan Van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[87] Jun Wang, Ying Yuan, Haichuan Che, Haozhi Qi, Yi Ma, Jitendra Malik, and Xiaolong Wang. Lessons from learning to spin" pens". *arXiv preprint arXiv:2407.18902*, 2024.

[88] Yuran Wang, Ruihai Wu, Yue Chen, Jiarui Wang, Jiaqi Liang, Ziyu Zhu, Haoran Geng, Jitendra Malik, Pieter Abbeel, and Hao Dong. Dexgarmentlab: Dexterous garment manipulation environment with generalizable policy. *arXiv preprint arXiv:2505.11032*, 2025.

[89] Guowei Xu, Ruijie Zheng, Yongyuan Liang, Xiyao Wang, Zhecheng Yuan, Tianying Ji, Yu Luo, Xiaoyu Liu, Jiaxin Yuan, Pu Hua, et al. Drm: Mastering visual reinforcement learning through dormant ratio minimization. *arXiv preprint arXiv:2310.19668*, 2023.

[90] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[91] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

[92] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[93] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.

Supplementary Material

VIII. Extended Experiments

We additionally evaluate FlashSAC in benchmarks that use single environment and CPU-based simulators.

Tasks: We benchmark 44 state-based control environments spanning a wide range of simulators and benchmarks:

- *MuJoCo (5 tasks):* Contact-rich locomotion tasks with complex multi-body dynamics.
- *DeepMind Control Suite (10 tasks):* Task spanning both low-dimensional sparse-reward environments and high-dimensional challenges like humanoid.
- *Humanoid Bench (14 tasks):* High-dimensional whole-body locomotion tasks based on the Unitree H1 robot.
- *MyoSuite (15 tasks):* Dexterous manipulation tasks using accurate musculoskeletal simulation from elbow to hand.

Detailed description and complete task list are provided in § X.

Baselines: We compare FlashSAC against on-policy algorithms (PPO), model-based algorithms (TD-MPC2), and model-free algorithms that focus on stabilizing the optimization process (XQC, SimbaV2) or learning model dynamics (MR.Q).

- *PPO* [71]: A highly optimized on-policy implementation from RSL-RL.
- *XQC* [62]: A lightweight off-policy method that stabilizes optimization by employing BatchNorm, distributional critic and weight normalization.
- *SimbaV2* [41]: A scalable off-policy method using hyperspherical normalization, distributional critic and weight normalization.
- *TD-MPC2* [25]: A model-based algorithm that performs local trajectory optimization in a learned latent world model.
- *MR.Q* [20]: An off-policy method with dynamics modeling objective to improve representation learning.

We reproduce the reuslts for PPO and XQC for all benchmarks. For TD-MPC2 and MR.Q, we report published results wherever feasible.

Experimental Setup: All off-policy methods are trained for 1M environment steps, while PPO is trained for 4M steps, requiring $2\times$ compute compared to FlashSAC . All methods are evaluated with a unified set of hyperparameters across all tasks.

Results: Figure 9 visualizes the aggregate performance for each benchmark, with full results in § XII.

In single-environment settings, PPO exhibits exacerbated data-inefficiency, resulting in suboptimal asymptotic performance. This highlights the limitations of on-policy methods when the flow of data is restricted. In contrast, FlashSAC demonstrates a clear and consistent advantage, achieving remarkable performance while requiring substantially less wall-clock time and fewer samples.

Algorithms that leverage model-based dynamics (TD-MPC2, MR.Q) show improved sample-efficiency and final performance. However, this comes at the cost of increased compute per update step, requiring $2-3\times$ more time to reach the same level of performance.

Finally, compared to XQC and SimbaV2, which similarly aim to stabilize off-policy learning, FlashSAC demonstrates more robust performance and higher efficiency. While SimbaV2 constrains feature norms via hyperspherical layers, it induces a sharper loss landscape [62]; conversely, XQC employs BatchNorm but does not explicitly bound feature magnitude. FlashSAC achieves the best of both worlds by integrating BatchNorm with post-RMSNorm, maintaining favorable optimization geometry without sacrificing bound constraints. Furthermore, FlashSAC incorporates enhanced exploration mechanisms, allowing it to escape local optima and achieve higher asymptotic performance.

IX. Measuring Wall-Clock Time

We adopt a unified protocol for measuring wall-clock time across all environments and simulators. All experiments are executed on identical hardware (AMD Ryzen 9 9950X3D CPU with an RTX 5090 GPU).

We decompose the total runtime into two components: (i) *environment interaction time*, which consists of environment stepping, physics simulation, and replay buffer operations; and (ii) *algorithm update time*, which consists of policy inference and gradient-based parameter updates.

Environment interaction time is measured independently for each environment. For a fixed environment, this cost is determined by the underlying simulator and is independent of the learning algorithm.

Algorithm update time is measured separately using representative benchmark environments. Specifically, we profile update time on MuJoCo Humanoid-v4 for state-based experiments and on DMC Walker-run for vision-based experiments. This update cost is reused across all environments, as it is largely determined by the algorithm's architecture and optimization procedure rather than the environment itself.

The total wall-clock time for a given environment is estimated by summing the measured environment interaction time and the corresponding algorithm update time. While differences in observation and action dimensionality across environments may lead to small variations in update cost, these effects are negligible in practice and do not affect the overall comparison.

X. Environment Details

We evaluate FlashSAC across a diverse set of simulation environments, grouped by simulator. For each environment, we report the observation space, action space, and the score used for normalization.

A. IsaacLab

We use IsaacLab v2.1.0 [54], a GPU-accelerated simulation platform built on NVIDIA Isaac Sim. Our evaluation includes 12 tasks spanning gripper manipulation, dexterous hand manipulation, quadruped locomotion, and humanoid locomotion. Scores are normalized per environment using a near-asymptotic performance reference obtained by running FlashSAC for an extended training duration. Full environment specifications are provided in Table I.

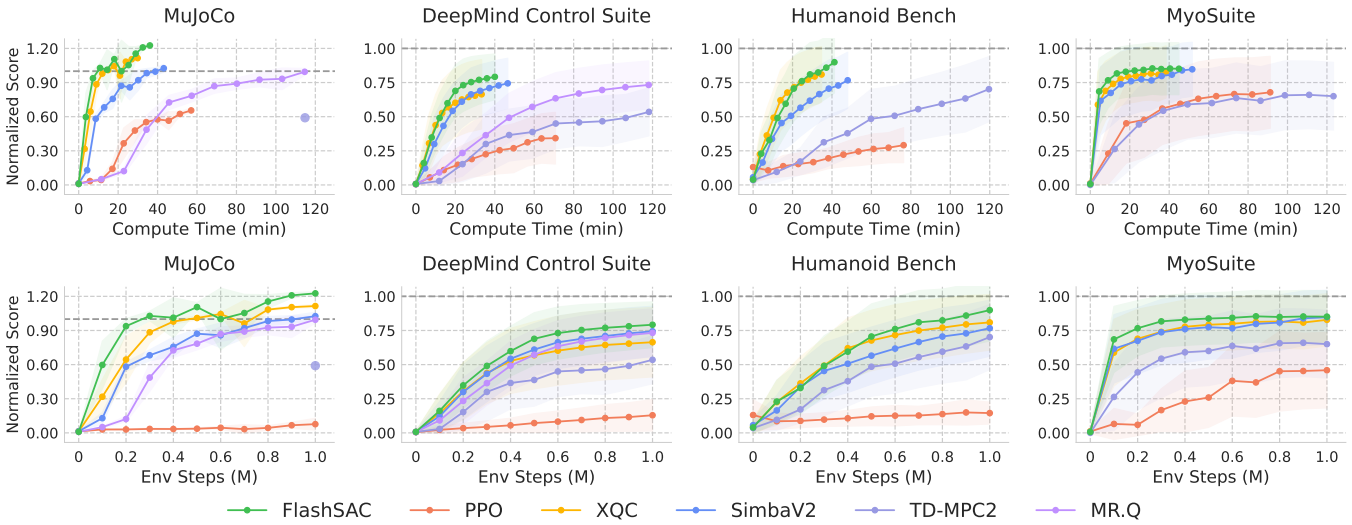

Fig. 9: **Results on State-Based RL (CPU-based Simulators).** Aggregate learning curves on MuJoCo [82, 81], DMC [79], HumanoidBench [73] and MyoSuite [11], with compute-efficiency (wall-clock time) on top row and sample-efficiency (environment steps) on bottom row. Across all benchmarks, FLASHSAC shows superior learning curve, converging to higher asymptotic performance in a relatively shorter period of learning time.

B. Mujoco Playground

We evaluate four humanoid locomotion tasks from the MuJoCo Playground suite (v0.0.5) [91], which emphasizes robust whole-body control under complex contact dynamics. For consistency across tasks, normalized scores are uniformly scaled to 40, corresponding to the asymptotic performance observed in converged training runs. Task details are listed in Table II.

C. ManiSkill

We employ ManiSkill [78], a large-scale benchmark built on the SAPIEN physics engine that supports diverse object geometries and physical interactions. We evaluate six rigid-body manipulation tasks using a gripper-based robotic arm. To ensure strict reproducibility, we use the environment snapshot corresponding to commit hash `aad75f2`. The full task list is shown in Table III.

D. Genesis

We benchmark three environments from the Genesis simulator (v0.3.13) [4], a general-purpose physics engine. We adapt the source code to comply with the standard Gymnasium API. Additional environment details are provided in Table IV.

E. Gym - MuJoCo

We use the Gym [82] continuous control benchmark simulated with MuJoCo [81]. Our evaluation focuses on five locomotion tasks (version `v4`) involving multi-body dynamics and contact-rich interactions. To enable comparisons across tasks with different reward scales, we normalize scores using TD7 baselines [19], with reference points defined by the random policy score and the score achieved after 5M training steps (approximating asymptotic performance). The task list is given in Table V.

F. DeepMind Control Suite

The DeepMind Control Suite (DMC) [79] provides a broad collection of continuous control tasks. We evaluate 10 tasks, ranging from low-dimensional sparse-reward environments (e.g., Cartpole, Pendulum) used to assess exploration, to high-dimensional tasks commonly referred to as `DMC-Hard`, such as Humanoid and Dog. Observation and action space details are reported in Table VI.

G. HumanoidBench

HumanoidBench [73] is a high-dimensional benchmark for whole-body control based on the Unitree H1 humanoid robot. We evaluate 14 locomotion tasks that require stable gait generation and balance. All scores are normalized using the success thresholds defined by the benchmark authors. Task specifications and dimensionalities are summarized in Table VII.

H. MyoSuite

MyoSuite [11] models human motor control using physiologically accurate musculoskeletal simulations. We evaluate 10 dexterous manipulation tasks involving the elbow, wrist, and hand. Following the benchmark's taxonomy, tasks are labeled `easy` when the target goal is fixed and `hard` when the goal is randomized. The complete set of evaluated tasks is listed in Table VIII.

TABLE I: **IsaacLab environments.** We evaluate 12 tasks from IsaacLab spanning gripper manipulation, dexterous manipulation, quadruped locomotion, and humanoid locomotion. Normalized scores are defined per environment and correspond to near-asymptotic performance achieved 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

TABLE II: **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 support 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

TABLE III: **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

TABLE IV: **Genesis environments.** We evaluate 3 reinforcement learning tasks from Genesis. Normalized scores are defined per environment and correspond to near-asymptotic performance achieved after extended training. If the environment support asymmetric observation, the privileged observation size is given in parentheses.

Task	Observation dim	Action dim	Normalize Score
`go2-walk_easy`	45	12	25
`go2-walk`	45 (60)	12	25
`panda-grasp`	14	6	2.5

TABLE V: **MuJoCo Environments.** We evaluate five standard MuJoCo environments. Normalized scores are computed relative to the performance of TD7 [19] after 5M training steps.

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	120.423	10281

TABLE VI: **DMC Environments.** We evaluate 10 DeepMind Control Suite tasks, including humanoid and dog embodiments and sparse-reward settings. Normalized scores correspond to the theoretical maximum.

Task	Observation dim	Action dim	Normalize Score
cartpole-balance_sparse	5	1	1000
cartpole-swingup_sparse	5	1	1000
pendulum-swingup	3	1	1000
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

TABLE VII: **HumanoidBench Environments.** We evaluate 14 humanoid locomotion tasks without hand control from HumanoidBench. Normalized scores correspond to task success.

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

TABLE VIII: **Myosuite Environments.** We evaluate 10 Myosuite environments. Normalized scores correspond to the maximum success rate, where 100 denotes a 100% success rate.

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

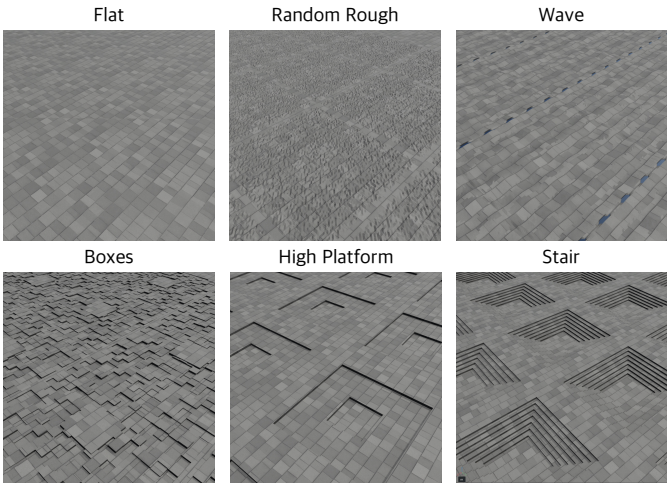

Fig. 10: **Terrain configuration.** Blind locomotion policies are trained on flat and diverse rough terrains, including pyramid stairs, random grids, random uniform terrain, wave terrain, and pit terrain. A terrain curriculum progressively increases difficulty across 10 levels as the policy successfully traverses the environment, enabling stable and adaptive locomotion without exteroceptive perception.

XI. Sim-to-Real Details

1) Simulation: We employed NVIDIA IsaacLab [54] as the simulation platform to train the FLASHSAC controller, with training environments built upon the Legged Gym framework [67]. The agents were trained in 4096 parallel environments with domain randomization, and training was completed in approximately 4 hours using a single NVIDIA A100 GPU. The resulting policy networks were directly deployed on the physical robot without additional fine-tuning.

2) Low-Level Control: The policy network outputs target joint positions at a frequency of 50 Hz, which are subsequently passed to a low-level PD controller running at 200 Hz. Within the PD controller, these target joint positions are translated into torque commands using proportional (K_p) and derivative (K_d) gains. To achieve natural and stable motion, we adopt a heuristic PD gain design following [44], with the specific parameters listed in Table IX.

The PD controller and sensor data measurement routines were implemented using a custom Pybind-based interface to connect our Python RL policy loop with the C++ implementation of Unitree SDK to ensure real-time low-level routine. This interface enables the transmission of target joint positions to the Unitree SDK through governed by a ROS2 node.

3) Terrain Configuration and Curriculum: We train blind locomotion policies on both flat and rough terrains composed of five terrain types as visualized in Figure 10:

- Pyramid stairs (maximum step height of 23cm)
- Random grids (maximum height of 15cm)
- Random uniform terrain (height noise range of -2 to 4cm)
- Wave terrain (maximum amplitude of 20cm)
- Pit terrain (maximum depth of 30cm)

Traversing these terrains without exteroceptive perception requires a high degree of stability and strong in-context adaptation. To facilitate the progressive acquisition of such challenging locomotion skills and to accelerate training, we employ terrain curriculum learning, a widely used technique in legged locomotion [67, 27]. The maximum curriculum level is set to 10, and terrain difficulty is automatically increased once the policy successfully traverses 50% of the environment. We observe that progressing beyond level 5 is difficult without a perception module and leads to aggressive motions of policies.

4) Observation space: The observation

$$\mathbf{o}_t = \begin{bmatrix} \boldsymbol{\omega}_t & \mathbf{g}_t & \mathbf{c}_t & \boldsymbol{q}_t & \dot{\boldsymbol{q}}_t & \mathbf{a}_{t-1} \end{bmatrix}^\top, \tag{8}$$

includes base angular velocity $\boldsymbol{\omega}_t$, joint position $\boldsymbol{q}_t$, joint velocity $\dot{\boldsymbol{q}}_t$, projected gravity $\boldsymbol{g}_t$, previous action $\boldsymbol{a}_{t-1}$, and velocity command $\boldsymbol{c}_t$, which are accessible on hardware during deployment. In addition, we incorporate context estimator network (CENet) [57] into actors of PPO and FLASHSAC to take $\boldsymbol{x}_t = [\boldsymbol{v}_t \ \boldsymbol{o}_t \ \boldsymbol{z}_t]$ as an input. This estimator encodes the proprioceptive observation history $\boldsymbol{o}_{t-H:t-1}$ and outputs predicted base linear velocity $\boldsymbol{v}_t$ and its latents $\boldsymbol{z}_t$, which are further enhanced by auxiliary loss of estimating base linear velocity during training. This history-based encoding enables implicit system identification for sim-to-real transfer and, together with auxiliary velocity estimation, ensures reliable state estimation for stable blind locomotion in real-world settings [42, 38, 33]. Furthermore, the training batch of these observations are augmented via symmetry augmentation [53], where we found it enchances sample efficiency and produces natural behavior. And we adopt asymmetric actor-critic framework [64], where the critic networks of both PPO and FLASHSAC also take a privileged information, comprising ground-truth base linear velocity, foot contact state and height map.

5) Reward: In Table XI, we summarize the reward configurations used for both algorithms. We adopt a shared reward structure comprising task rewards for base velocity tracking and regularization terms penalizing foot slip, joint torques, action rate, and orientation instability. However, due to the differing learning dynamics between our method and PPO, different reward weights are required for stable real-world deployment [72]. A notable distinction lies in the termination penalty. PPO requires substantial training time to achieve stable locomotion behavior without termination-based shaping, whereas FLASHSAC does not. Consequently, we apply only a minimal alive reward for FLASHSAC to avoid premature termination.

TABLE IX: **Joint Information of Unitree G1 humanoid.** Joint list of Unitree G1 29-DoF with default angle, stiffness K_p, and damping K_d, where we adopt heuristic parameters from [44].

Joint name	**Default angle**	K_p	K_d
left_hip_pitch_joint	-0.2	40.16	2.559
left_hip_roll_joint	0	99.08	6.311
left_hip_yaw_joint	0	40.16	2.559
left_knee_joint	0.42	99.08	6.311
left_ankle_pitch_joint	-0.23	28.49	1.815
left_ankle_roll_joint	0	28.49	1.815
right_hip_pitch_joint	-0.2	40.16	2.559
right_hip_roll_joint	0	99.08	6.311
right_hip_yaw_joint	0	40.16	2.559
right_knee_joint	0.42	99.08	6.311
right_ankle_pitch_joint	-0.23	28.49	1.815
right_ankle_roll_joint	0	28.49	1.815
waist_yaw_joint	0	40.16	2.559
waist_roll_joint	0	28.49	1.815
waist_pitch_joint	0	28.49	1.815
left_shoulder_pitch_joint	0.35	14.25	0.907
left_shoulder_roll_joint	0.18	14.25	0.907
left_shoulder_yaw_joint	0	14.25	0.907
left_elbow_joint	0.87	14.25	0.907
left_wrist_roll_joint	0	14.25	0.907
left_wrist_pitch_joint	0	16.78	1.068
left_wrist_yaw_joint	0	16.78	1.068
right_shoulder_pitch_joint	0.35	14.25	0.907
right_shoulder_roll_joint	-0.18	14.25	0.907
right_shoulder_yaw_joint	0	14.25	0.907
right_elbow_joint	0.87	14.25	0.907
right_wrist_roll_joint	0	14.25	0.907
right_wrist_pitch_joint	0	16.78	1.068
right_wrist_yaw_joint	0	16.78	1.068

TABLE X: **Observation Space for Sim-to-Real Experiments.** Our observation space combines proprioceptive information of the robot state and the joint states. where base linear velocity $\boldsymbol{v}_t$ and exteroceptive height map $\boldsymbol{h}_t$ are privileged observations for critic. $\mathcal{U}$ indicates uniform distributions used to augment the measurements and make the system robust against sensor noise.

Observation	**Notation**	**Dimension**	**Augmentation**	**Unit**
Base linear velocity	$\boldsymbol{v}_t$	3	$\mathcal{U}[-0.1, 0.1]$	m/s
Base angular velocity	$\boldsymbol{\omega}_t$	3	$\mathcal{U}[-0.2, 0.2]$	rad/s
Projected gravity	$\boldsymbol{g}_t$	3	$\mathcal{U}[-0.05, 0.05]$	m/s^2
Joint position	$\boldsymbol{q}_t$	29	$\mathcal{U}[-0.01, 0.01]$	rad
Joint velocity	$\dot{\boldsymbol{q}}_t$	29	$\mathcal{U}[-1.5, 1.5]$	rad/s
Last joint action	$\boldsymbol{a}_t$	29	-	rad
Command velocity	$\boldsymbol{c_t}$	3	-	m/s, rad/s
Height map	$\boldsymbol{h}_t$	17×11	-	m

TABLE XI: **Reward Configurations for Sim-to-Real Experiments.** Both methods share the same reward structure, including task rewards for tracking velocities, style rewards for its gait style and regularization terms penalizing excessive joint torques, action rate, and orientation instability. Different reward weights are applied to account for differing learning dynamics and ensure stable real-world deployment [72].

Reward	Expression	Weight (FlashSAC)	Weight (PPO)
Task			
Track linear velocity	$\exp\big(-\lVert \boldsymbol{v}_{xy}^{\mathrm{cmd}} - \boldsymbol{v}_{xy}^{\mathrm{yaw}} \rVert^2 / \sigma^2\big)$	2.0 ($\sigma = 0.25$)	1.5 ($\sigma = 0.5$)
Track angular velocity	$\exp\big(-(\boldsymbol{\omega}_z^{\mathrm{cmd}} - \boldsymbol{\omega}_z)^2 / \sigma^2\big)$	1.5 ($\sigma = 0.25$)	1.5 ($\sigma = 0.5$)
Orthogonal velocity	$\exp\big(-1.5 \lVert \boldsymbol{v}_{\perp} \rVert^2\big)$	1.0	1.0
Style			
Feet air time	$r_{\mathrm{airtime}}(t_{\mathrm{swing}},\, t_{\mathrm{stance}},\, \lVert \boldsymbol{c} \rVert)$	1.0	0.25
Feet slide	$\sum_{i=1}^{2} \lVert \boldsymbol{v}_{xy}^{\mathrm{ft}_i} \rVert \cdot \mathbf{1}[c^{\mathrm{ft}_i}]$	-0.25	-0.25
Feet yaw drag	$\sum_{i=1}^{2} \lvert \boldsymbol{\omega}_z^{\mathrm{ft}_i} \rvert \cdot \mathbf{1}[c^{\mathrm{ft}_i}]$	-0.5	-0.25
Feet force	$\mathrm{clamp}\big(\lVert \boldsymbol{f}_z^{\mathrm{ft}} \rVert - 700,\, 0,\, 400\big)$	$-3{\times}10^{-3}$	$-3{\times}10^{-3}$
Feet lateral distance	$\mathrm{clamp}\big(0.3 - d_{\mathrm{lateral}},\, 0,\, \infty\big)$	-5.0	-2.0
Feet stumble	$\mathbf{1}\big[\lVert \boldsymbol{f}_{xy}^{\mathrm{ft}} \rVert > 5 \lvert \boldsymbol{f}_z^{\mathrm{ft}} \rvert\big]$	-2.0	-2.0
Air time variance	$\mathrm{Var}[t_{\mathrm{air}}] + \mathrm{Var}[t_{\mathrm{contact}}]$	-1.0	-1.0
Impact velocity delta	$\sum_{i=1}^{2} \min\big(\Delta \boldsymbol{v}_{z,\mathrm{ft}_i}^2,\, 1\big)$	-5.0	-5.0
Regularization			
Linear velocity (z)	$\boldsymbol{v}_z^2$	-0.25	-0.25
Angular velocity (xy)	$\lVert \boldsymbol{\omega}_{xy} \rVert^2$	-1.0	-0.05
Energy	$\lVert \boldsymbol{\tau} \odot \dot{\boldsymbol{q}} \rVert$	-10^{-3}	-10^{-3}
Joint acceleration	$\lVert \ddot{\boldsymbol{q}} \rVert^2$	$-2.5{\times}10^{-7}$	$-2.5{\times}10^{-7}$
Action rate	$\lVert \boldsymbol{a}_t - \boldsymbol{a}_{t-1} \rVert^2$	-0.5	-0.01
Flat orientation	$\lVert \boldsymbol{g}_{xy}^{\mathrm{base}} \rVert^2$	-5.0	-1.0
Body orientation	$\lVert \boldsymbol{g}_{xy}^{\mathrm{torso}} \rVert^2$	-52.0	-2.0
Joint deviation	$\lVert \boldsymbol{q} - \boldsymbol{q}_0 \rVert_1$	-0.25 (leg) -0.50 (hip) -1.00 (arm)	-0.02 (leg) -0.15 (hip) $-0.2.0$ (arm)
Joint position limits	$\max(\boldsymbol{q} - \boldsymbol{q}_{\mathrm{max}}, 0) + \max(\boldsymbol{q}_{\mathrm{min}} - \boldsymbol{q}, 0)$	-5.0	-2.0
Stand still	$\lVert \boldsymbol{q} - \boldsymbol{q}_0 \rVert^2 \cdot \mathbf{1}\big[\lVert \boldsymbol{c} \rVert < 0.1\big]$	-5.0	-0.25
Safety & Termination			
Undesired contacts	$\sum_i \mathbf{1}\big[\lVert \boldsymbol{f}_i \rVert > 1\big]$ (non-ankle)	-5.0	-1.0
Fly	$\mathbf{1}[\texttt{no ankle contact}]$	-1.0	-1.0
Termination penalty	$\mathbf{1}[\texttt{terminated} \wedge \neg\, \texttt{timeout}]$	-	-200
Alive bonus	$\mathbf{1}[\neg\, \texttt{terminated}]$	1.0	-

TABLE XII: **Notation for Reward Terms.** Symbol definitions of the reward terms in Table XI.

Symbol	Description
$\boldsymbol{c} = [\boldsymbol{v}_{xy}^{\mathrm{cmd}}\ \boldsymbol{\omega}_z^{\mathrm{cmd}}]$	Commanded xy linear velocity and yaw angular velocity from the command velocity $\boldsymbol{c}$.
$\boldsymbol{v}_{xy}^{\mathrm{yaw}}, \boldsymbol{\omega}_z$	Robot xy linear velocity projected into the yaw-aligned body frame and yaw angular velocity.
$\boldsymbol{v}_{\perp}$	Velocity component orthogonal to the commanded direction; for stop commands ($\lVert \boldsymbol{c} \rVert < 0.1$), equals $\boldsymbol{v}_{xy}^{\mathrm{yaw}}$
$\boldsymbol{q}_0$	Default joint positions.
$(\boldsymbol{q}_{\mathrm{min}}, \boldsymbol{q}_{\mathrm{max}})$	Soft joint position limit for joint j.
$\boldsymbol{g}_{xy}^{\mathrm{base}}$, $\boldsymbol{g}_{xy}^{\mathrm{torso}}$	xy components of the gravity vector projected into the base and torso body frames; $\mathbf{0}$ when perfectly upright.
$\boldsymbol{f}_i$	Net contact force on body i in world frame.
$\boldsymbol{f}_{xy}^{\mathrm{ft}}$, $\boldsymbol{f}_z^{\mathrm{ft}}$	Horizontal and vertical components of foot contact force, respectively.
c^{ft_i}	Binary contact state of foot i, true when $\lVert \boldsymbol{f}^{\mathrm{ft}_i} \rVert > 1\mathrm{N}$.
$\boldsymbol{v}_{xy}^{\mathrm{ft}_i}, \boldsymbol{\omega}_z^{\mathrm{ft}_i}$	xy linear and yaw angular velocity of foot i in world frame, respectively
d_{lateral}	Lateral (y-axis) distance between left and right feet in the body frame; threshold set to 0.3 m.
$\Delta \boldsymbol{v}_{z,\mathrm{ft}_i}$	Frame-to-frame change in vertical velocity of foot i; clamped at $\Delta v_{\max}$=1.0 m/s.
t_{swing}, t_{stance}	Current air (swing) time and ground contact (stance) time per foot.
t_{air}, t_{contact}	Last completed air and contact durations per foot, used for variance computation.
r_{airtime}	Composite feet air-time reward with in-place handling; blends swing-target (0.4 sec), stance-target ($0.5/\lVert \boldsymbol{c} \rVert$, clipped to $[0.1, 0.5]$ sec), and in-place mode ($\lVert \boldsymbol{c} \rVert < 0.25$).
$\mathbf{1}[\cdot]$	Indicator function, returning 1 when the condition is true and 0 otherwise.

TABLE XIII: **Hyperparameters (GPU-Accelerated Simulators).** FLASHSAC hyperparameters used in benchmarks that support massive parallel environments (IsaacLab, MuJoCo Playground, ManiSkill, and Genesis).

	Hyperparameter	Notation	Value
Common	Parallel environments	-	1024
	Replay buffer capacity	-	10M
	Batch size	-	2048
	Update-to-data (UTD) ratio	-	2/2048
	TD steps (n-step)	n	1
Actor	Number of blocks	-	2
	Hidden dimension	d_{actor}	128
	Update delay	-	2
Critic	Number of blocks		2
	Hidden dimension	d_{critic}	256
	Target critic momentum	τ	0.01
	Number of critics	-	2
	Value prediction type	-	Categorical
	Categorical Support	$[G_{\min}, G_{\max}]$	[-5,5]
	Number of bins	n_{atoms}	101
Temperature	Entropy target	σ_{tgt}	0.15
	Initial value	-	0.01
Optimizer	Optimizer		Adam
	Optimizer momentum	(β_1, β_2)	(0.9, 0.999)
	Learning rate scheduler	-	Cosine Decay
	Learning rate init	η	3e-4
	Learning rate end	-	1.5e-4
Noise Repeat	Zeta distribution exponent	s	2
	Maximum repeat limit	-	16

TABLE XIV: **Hyperparameters (Non-Accelerated Simulators).** FLASHSAC hyperparameters used in benchmarks with no parallel environments (MuJoCo, DMC, Humanoid Bench and MyoSuite). We list only the values that differ from the settings provided in table XIII.

	Hyperparameter	Notation	Value
Common	Parallel environments	-	1
	Replay buffer capacity	-	1M
	Batch size	-	256
	Update-to-data (UTD) ratio	-	1
	TD Steps (n-step)	n	1

TABLE XV: **Hyperparameters (Vision-Based RL).** FLASHSAC hyperparameters used in vision-based tasks. We list only the values that differ from the settings provided in table XIII.

	Hyperparameter	Notation	Value
Common	Parallel environments	-	1
	Replay buffer capacity	-	1M
	Batch size	-	256
	Update-to-data (UTD) ratio	-	0.5
	TD Steps (n-step)	n	3
	Action repeat	-	2
	Frame stack	-	3
Encoder	Number of layers	-	4
	Number of channels	-	32
	Output feature dim	-	50

XII. Complete Results

We report the learning curves for each task across all algorithms. The results are plotted against wall-clock time and environment steps to illustrate compute and sample efficiency, respectively.

A. *IsaacLab (State-based RL, GPU Simulator)*

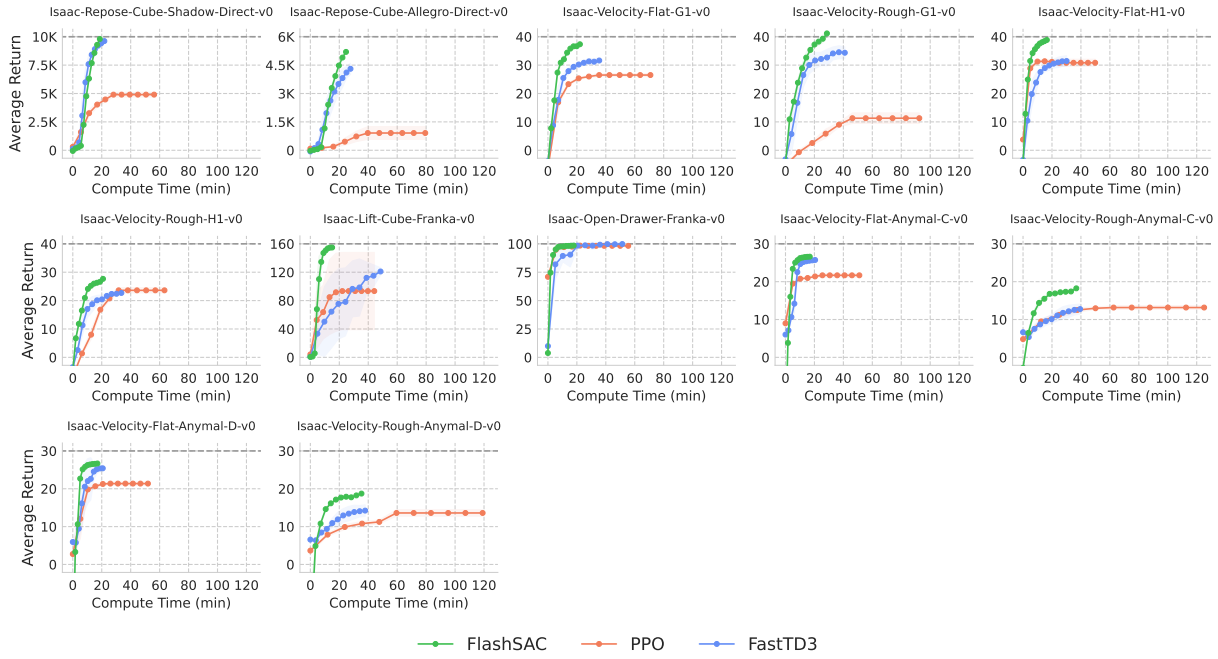

Fig. 11: **IsaacLab Learning Curves (Compute Efficiency).** Average episode returns in IsaacLab, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 50M environment steps except for PPO (200M).

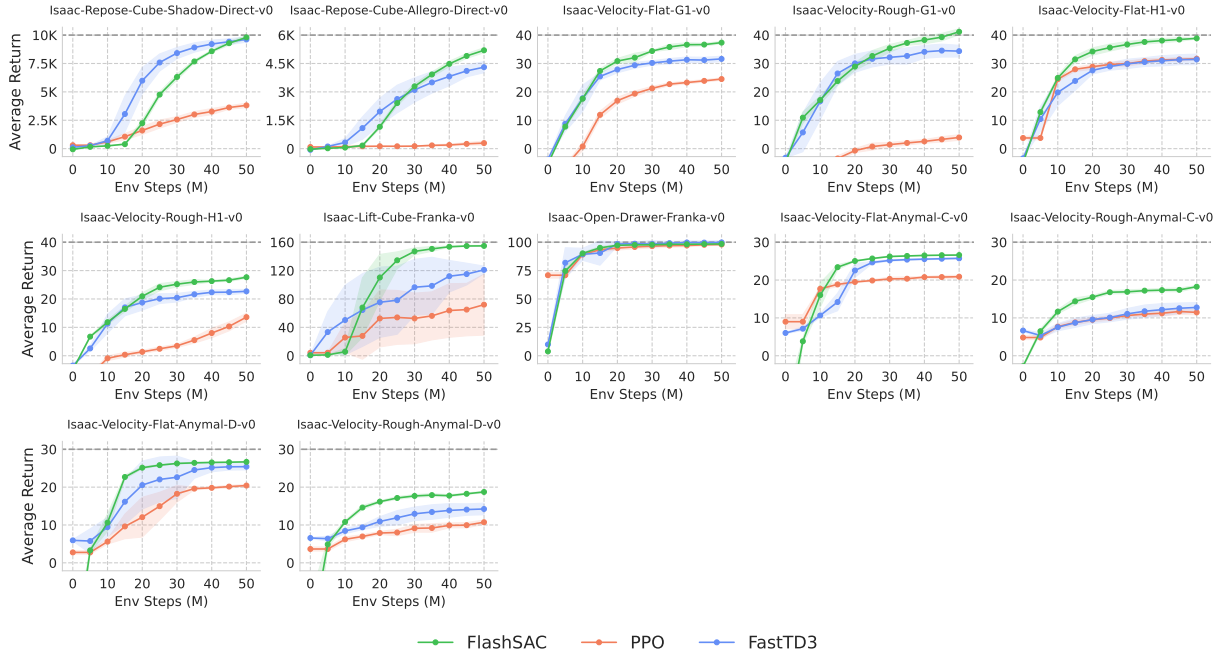

Fig. 12: **IsaacLab Learning Curves (Sample Efficiency).** Average episode returns in IsaacLab environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

B. Mujoco Playground (State-based RL, GPU Simulator)

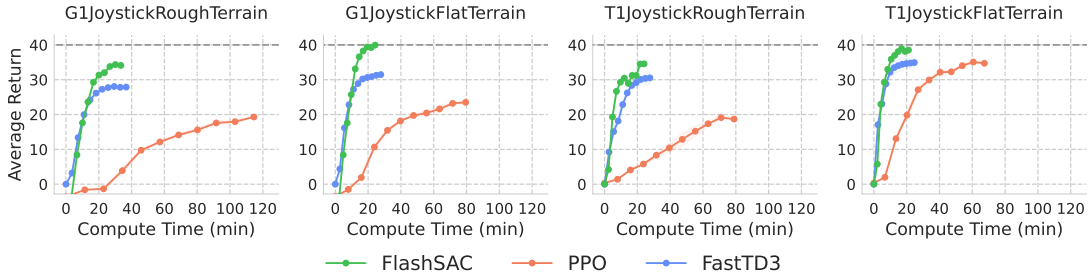

Fig. 13: **Mujoco Playground Learning Curves (Compute Efficiency).** Average episode returns in Mujoco Playground environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 50M environment steps except for PPO (200M).

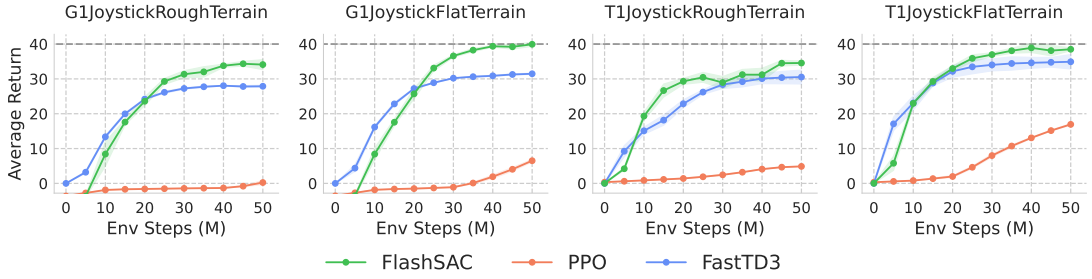

Fig. 14: **Mujoco Playground Learning Curves (Sample Efficiency).** Average episode returns in Mujoco Playground environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

C. ManiSkill (State-based RL, GPU Simulator)

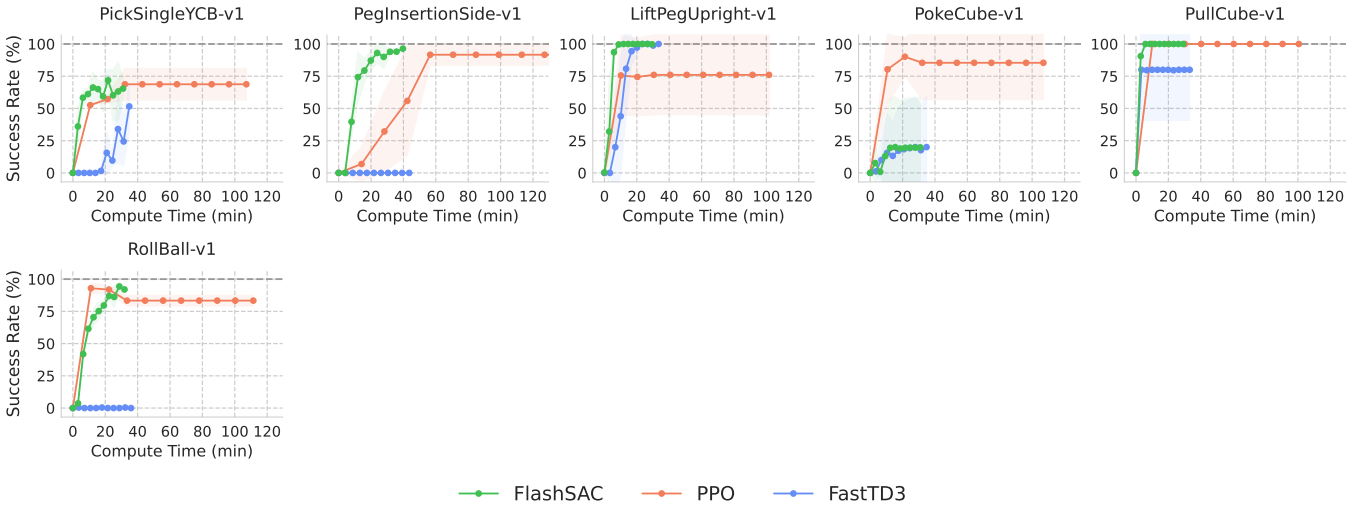

Fig. 15: **ManiSkill Learning Curves (Compute Efficiency).** Average episode returns in ManiSkill environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 50M environment steps except for PPO (200M).

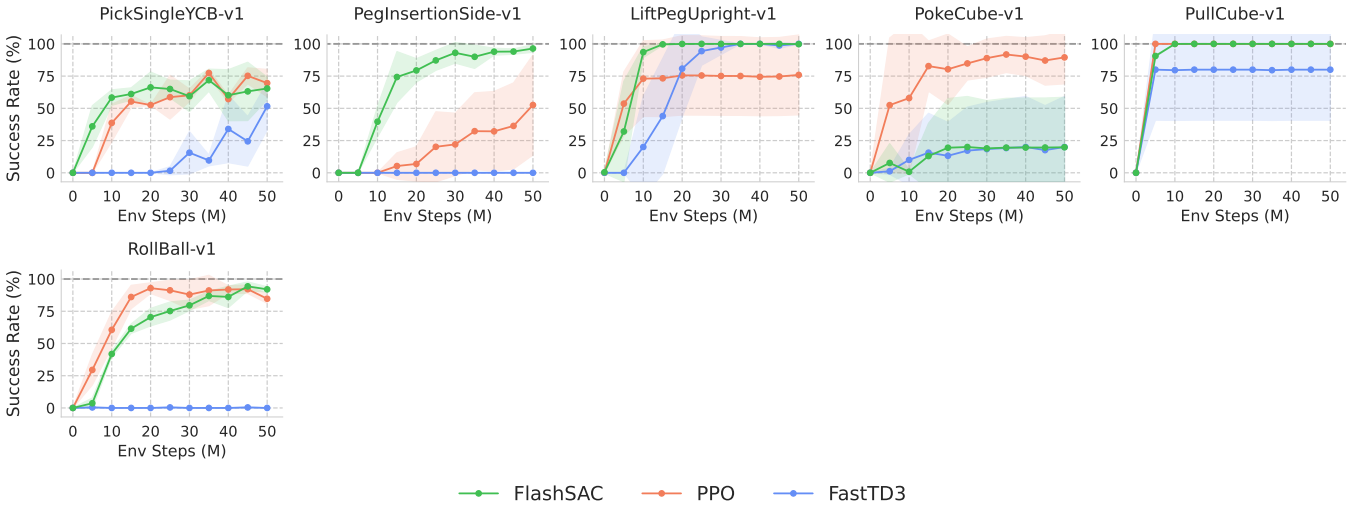

Fig. 16: **ManiSkill Learning Curves (Sample Efficiency).** Average episode returns in ManiSkill environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

D. Genesis (State-based RL, GPU Simulator)

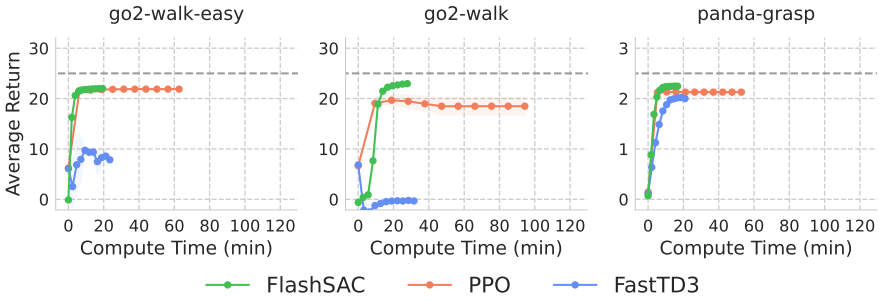

Fig. 17: **Genesis Learning Curves (Compute Efficiency).** Average episode returns in Genesis environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 50M environment steps except for PPO (200M).

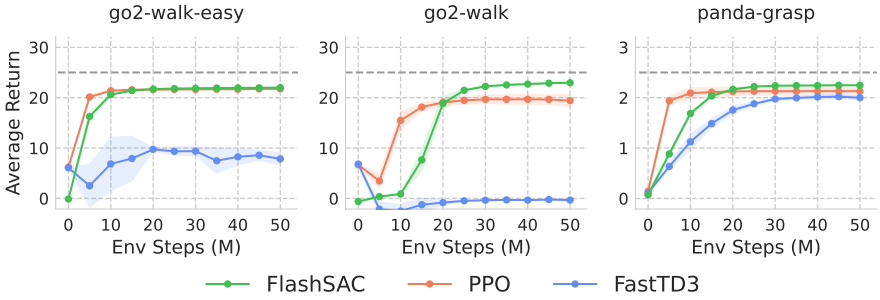

Fig. 18: **Genesis Learning Curves (Sample Efficiency).** Average episode returns in Genesis environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

E. Mujoco (State-based RL, CPU Simulator)

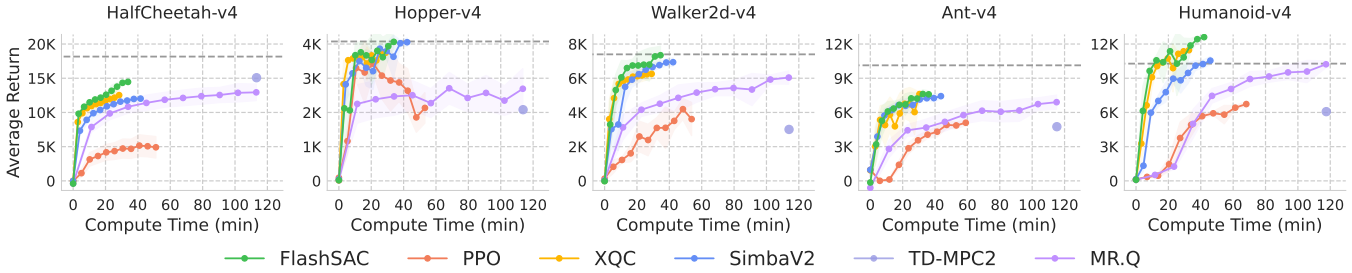

Fig. 19: **MuJoCo Learning Curves (Compute Efficiency).** Average episode returns in MuJoCo environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 1M environment steps, except for PPO (4M).

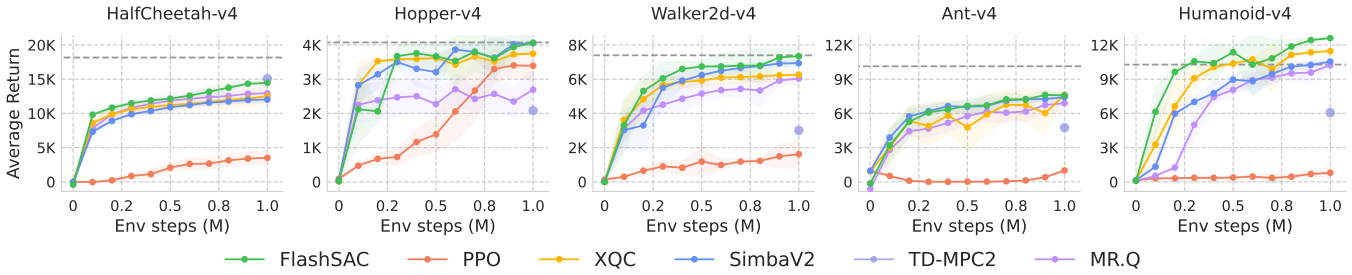

Fig. 20: **MuJoCo Learning Curves (Sample Efficiency).** Average episode returns in MuJoCo environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

F. DeepMind Control Suite (State-based RL, CPU Simulator)

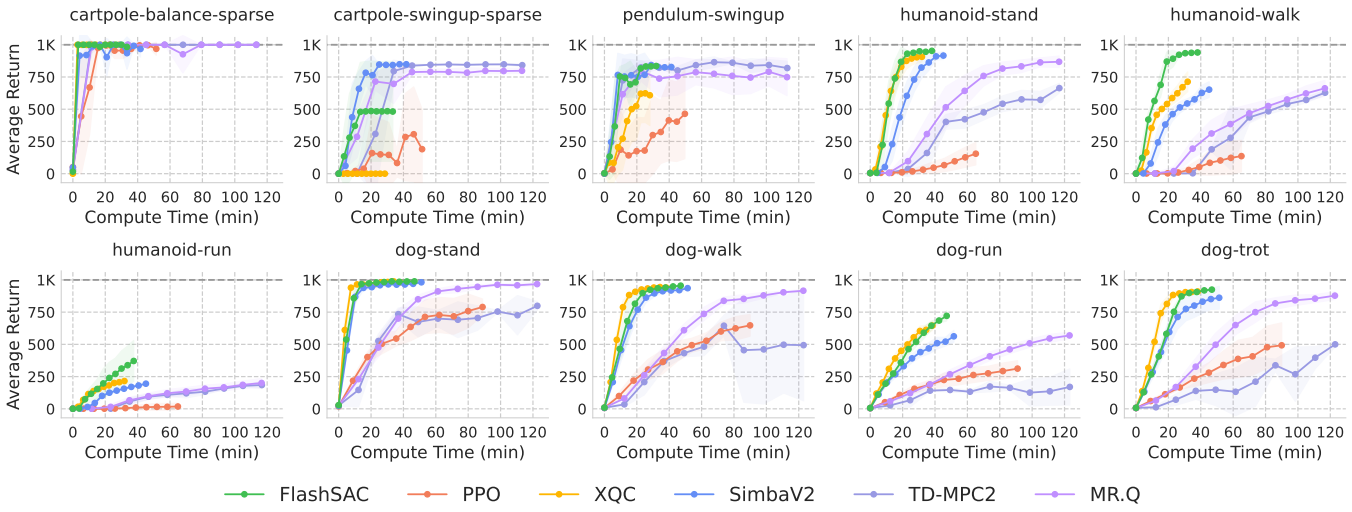

Fig. 21: **DMC Learning Curves (Compute Efficiency).** Average episode returns in DMC environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 1M environment steps, except for PPO (4M).

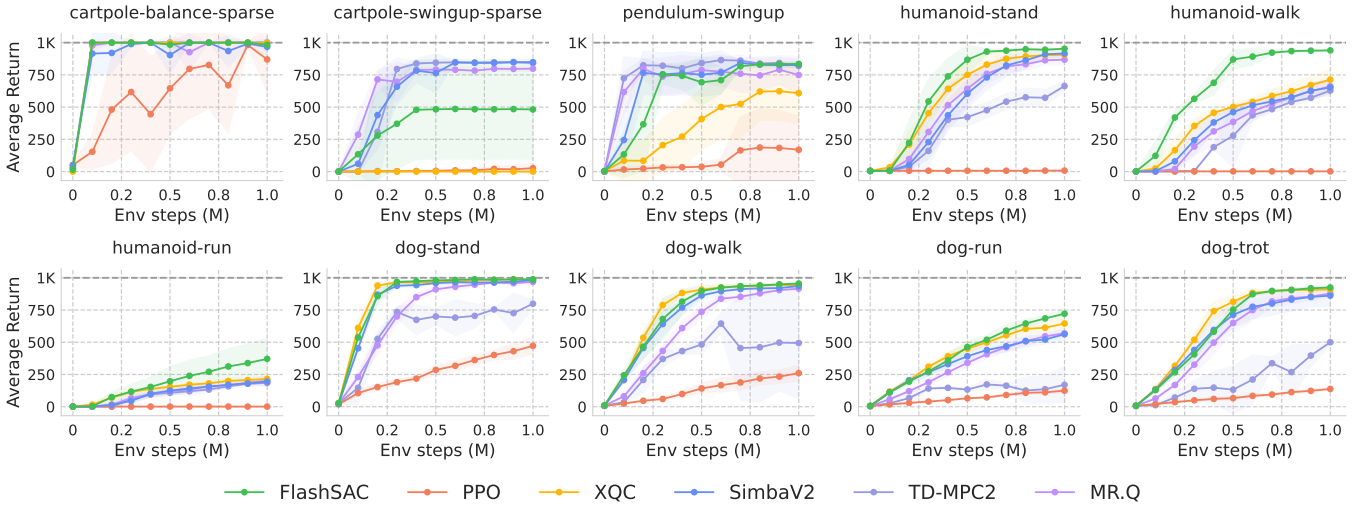

Fig. 22: **DMC Learning Curves (Sample Efficiency).** Average episode returns in DMC environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

G. Humanoid Bench (State-based RL, CPU Simulator)

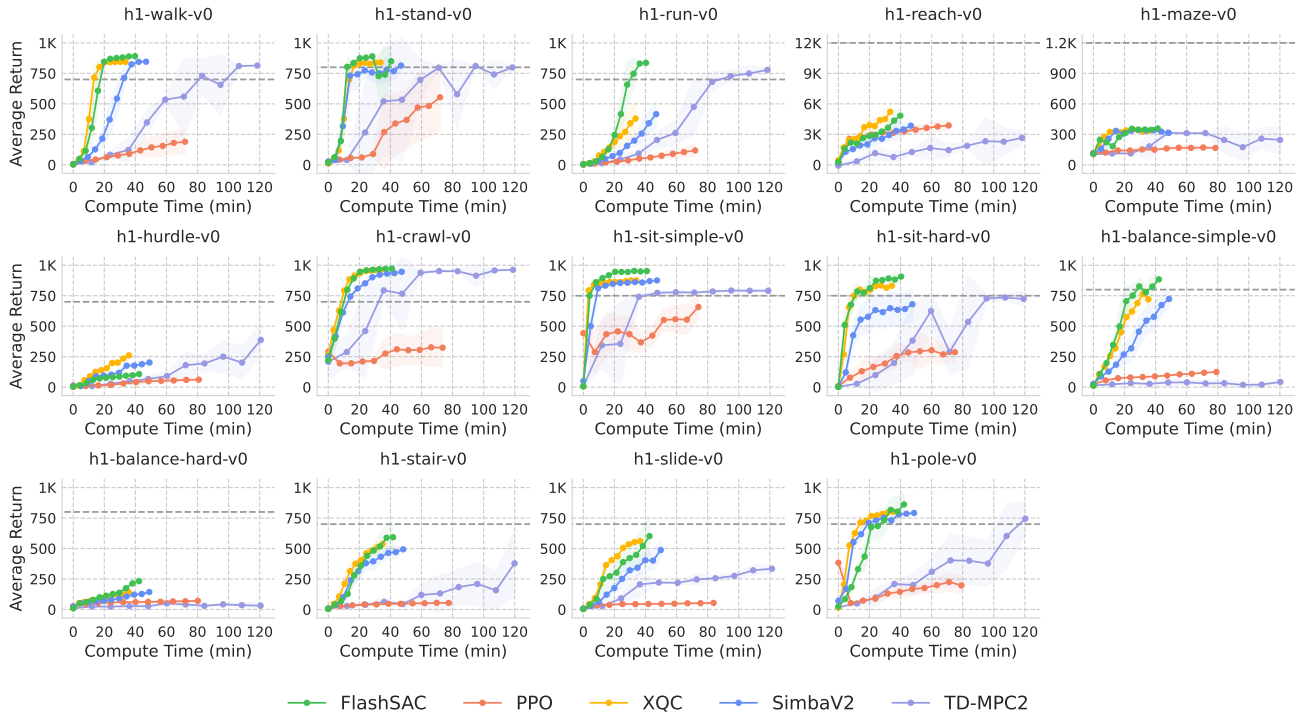

Fig. 23: **Humanoid Bench Learning Curves (Compute Efficiency).** Average episode returns plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 1M environment steps, except for PPO (4M).

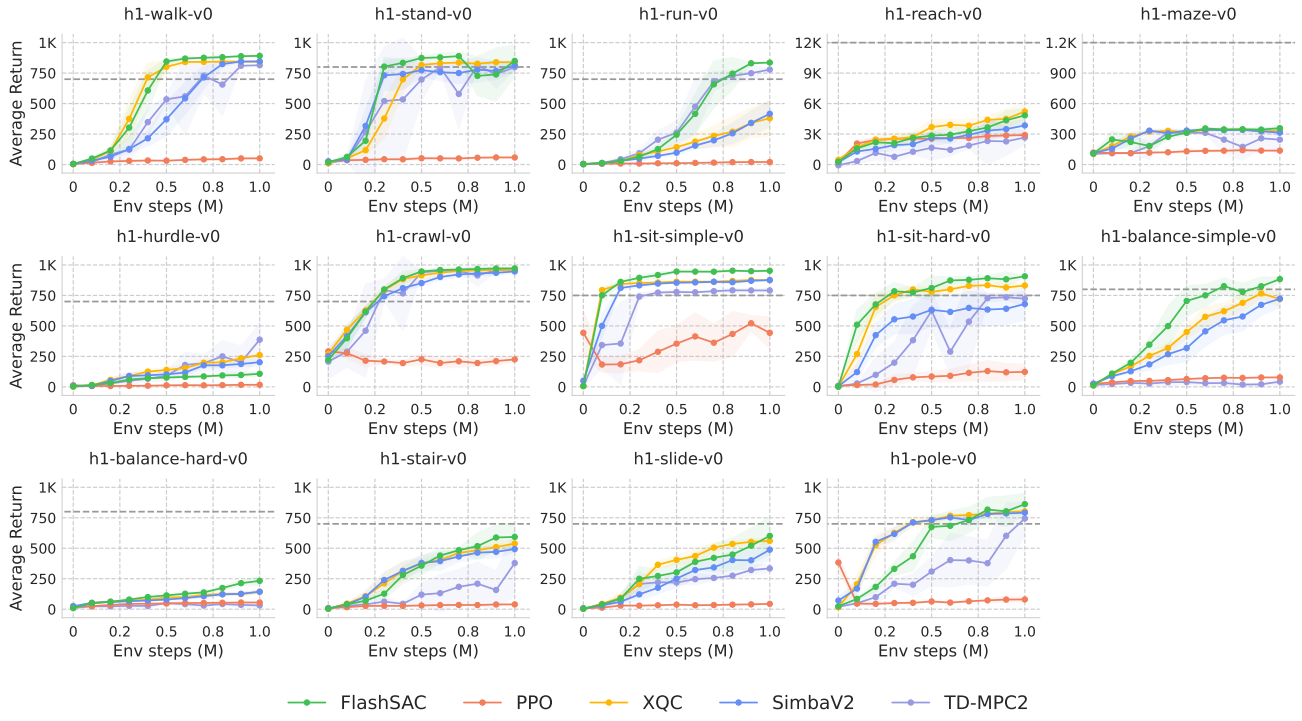

Fig. 24: **Humanoid Bench Learning Curves (Sample Efficiency).** Average episode returns in Humanoid Bench environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

H. MyoSuite (State-based RL, CPU Simulator)

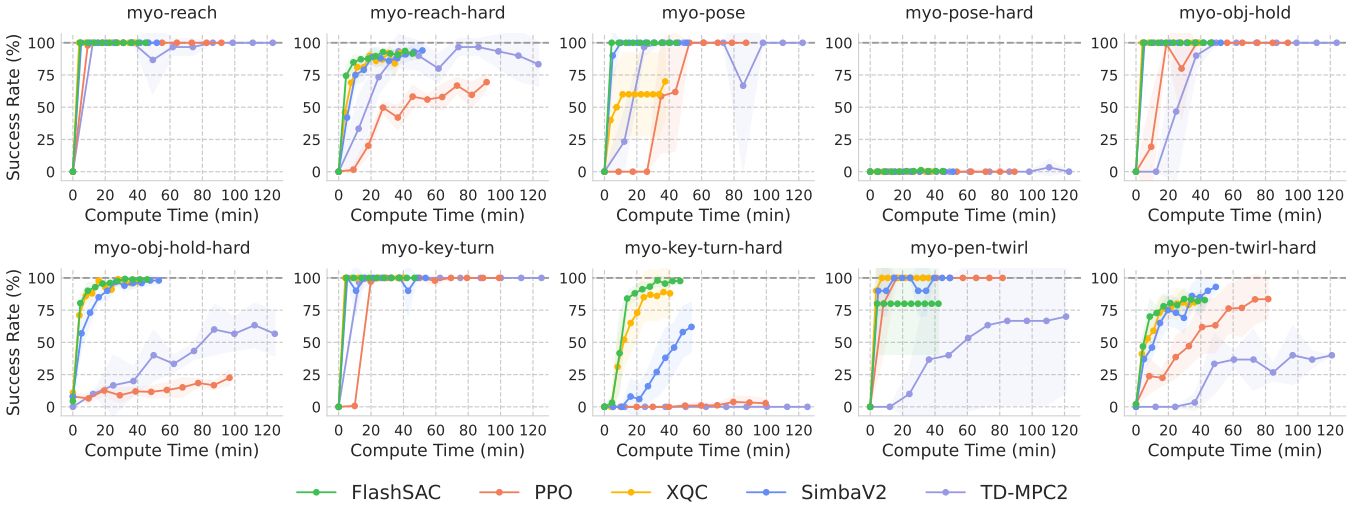

Fig. 25: **MyoSuite Learning Curves (Compute Efficiency).** Average episode returns in MyoSuite environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 1M environment steps, except for PPO (4M).

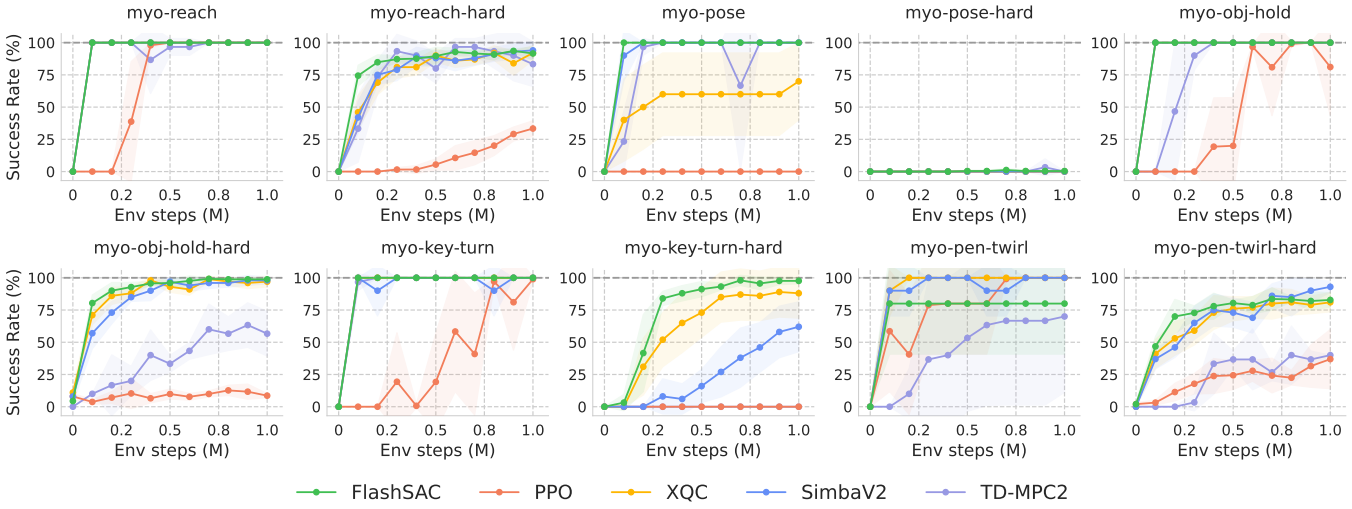

Fig. 26: **MyoSuite Learning Curves (Sample Efficiency).** Average episode returns in MyoSuite environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

I. *Vision-based RL*

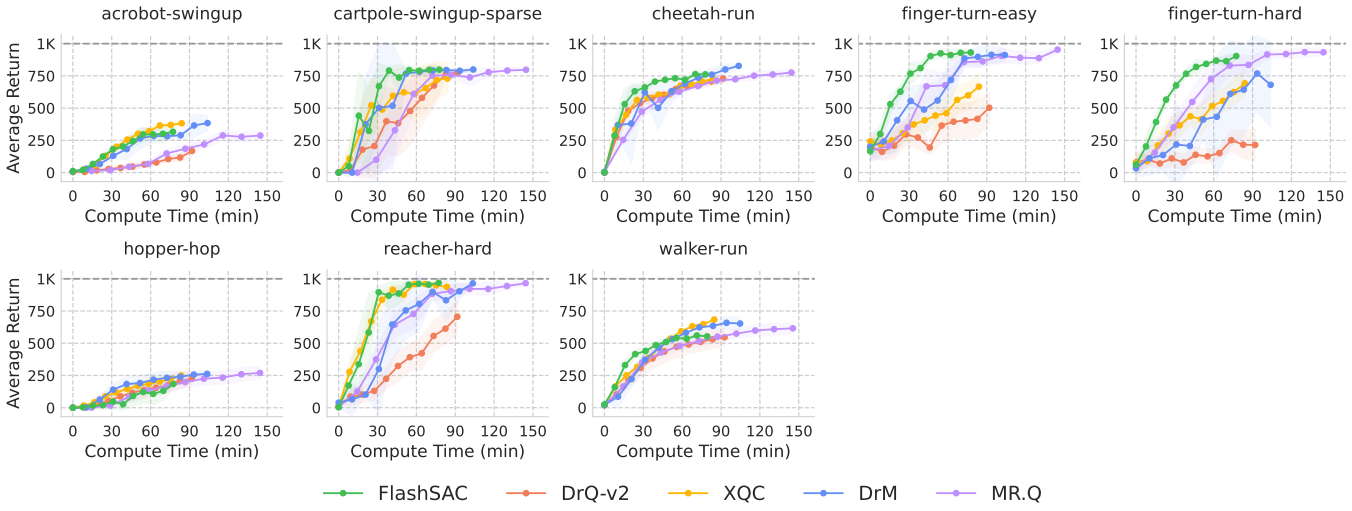

Fig. 27: **DMC-Visual Learning Curves (Compute Efficiency).** Average episode returns in DMC-Visual environments, plotted against total compute time. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score. All methods are trained for 1M environment steps.

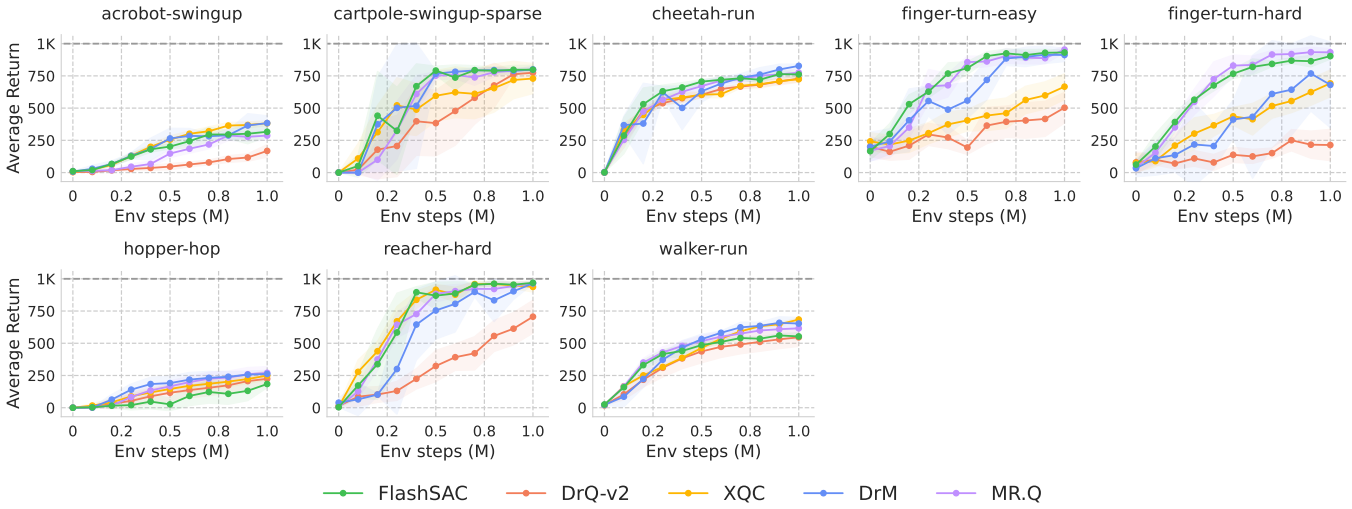

Fig. 28: **DMC-Visual Learning Curves (Sample Efficiency).** Average episode returns in DMC-Visual environments, plotted against environment steps. Results are averaged over random seeds of each algorithm, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

J. Component Ablation

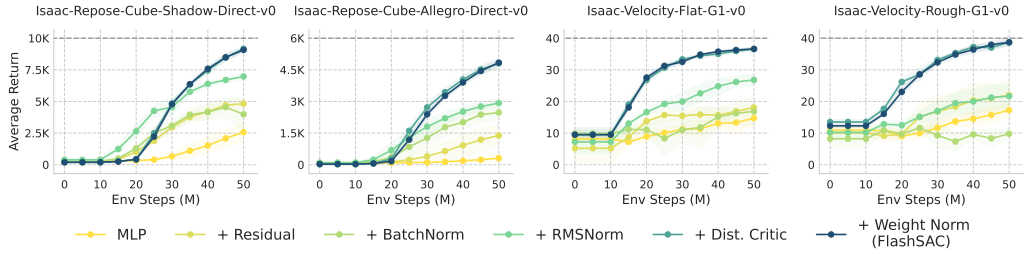

Fig. 29: **Component Ablation Learning Curves.** Average episode returns in IsaacLab environments, plotted against environment steps. Results are averaged over random seeds of each configuration, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

K. Hyperparameter Ablation

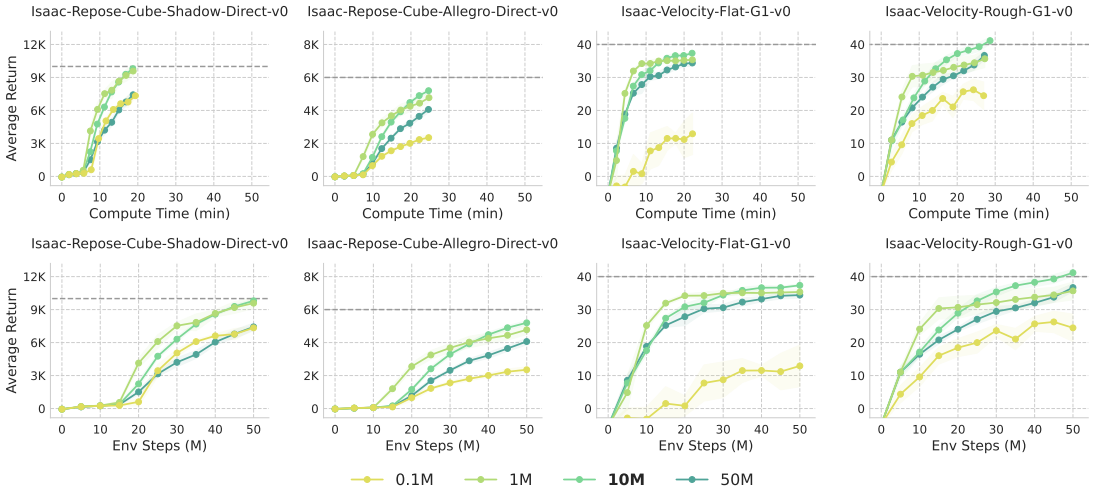

Fig. 30: **Hyperparameter Ablation Learning Curves - Buffer Size.** Each configuration indicates the maximum size of replay buffer. Results are averaged over random seeds of each configuration, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

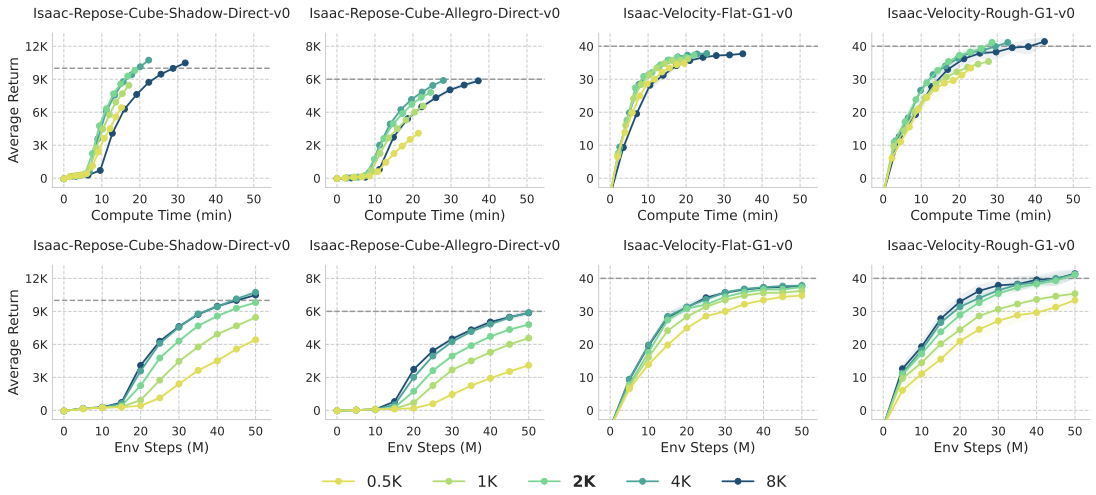

Fig. 31: **Hyperparameter Ablation Learning Curves - Batch Size.** Each configuration indicates size of mini-batch used for training. Results are averaged over random seeds of each configuration, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

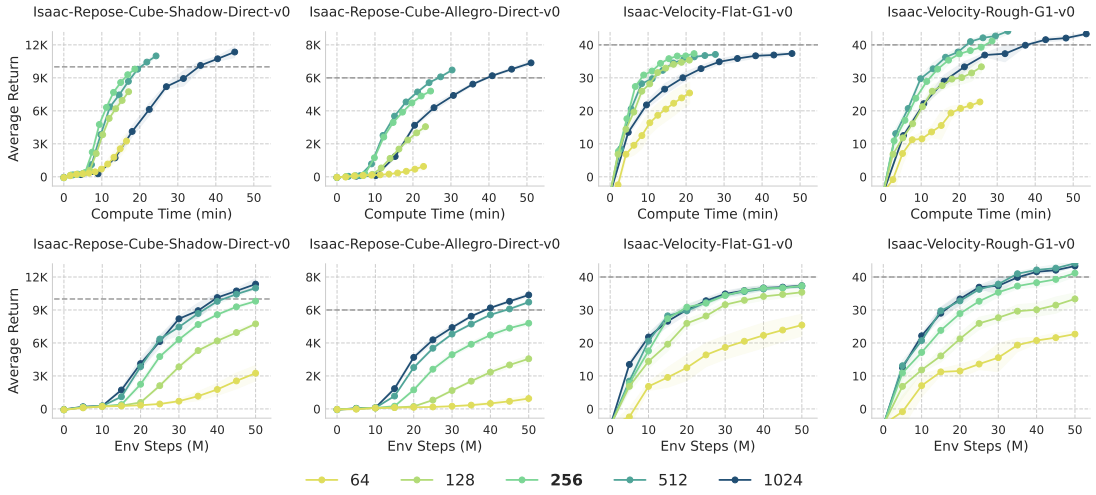

Fig. 32: **Hyperparameter Ablation Learning Curves - Network Width.** Each configuration indicates the hidden dimension of critic network, while the actor network is proportionally scaled as well. Results are averaged over random seeds of each configuration, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

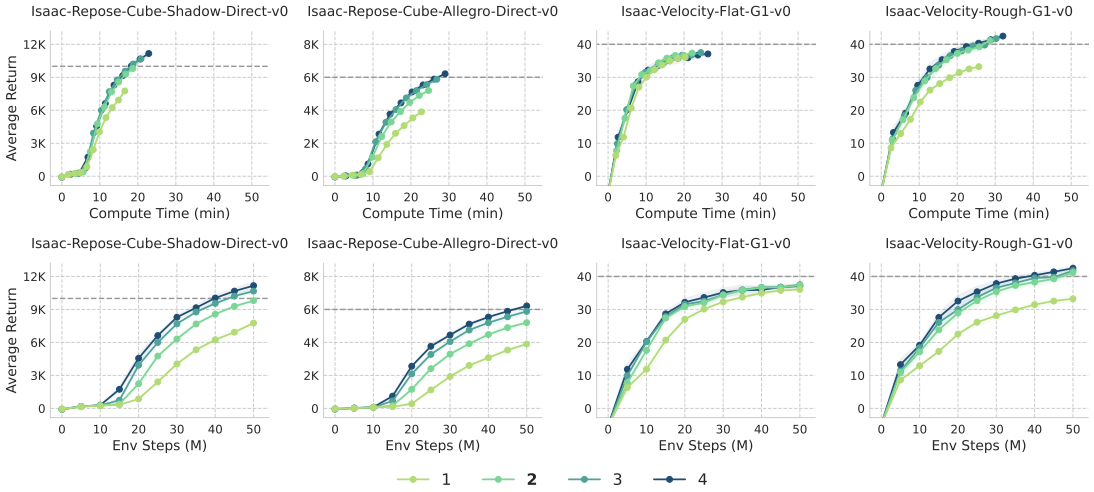

Fig. 33: **Hyperparameter Ablation Learning Curves - Network Depth.** Each configuration indicates the number of residual blocks in both actor and critic networks. Average episode returns in IsaacLab environments, plotted against environment steps. Results are averaged over random seeds of each configuration, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.

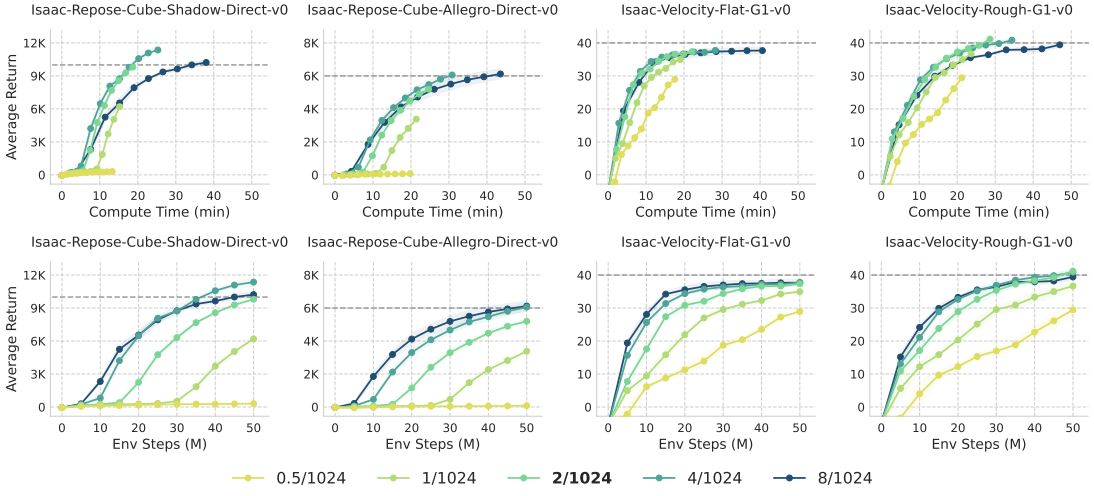

Fig. 34: **Hyperparameter Ablation Learning Curves - UTD Ratio.** Each configuration indicates the update-to-data ratio for training. Average episode returns in IsaacLab environments, plotted against environment steps. Results are averaged over random seeds of each configuration, with shaded regions indicating 95% bootstrap confidence intervals and dotted lines denoting normalize score.