UniLab: A Heterogeneous Architecture for Robot RL Beyond GPU-Dominant Paradigms

Anonymous Authors

Fig. 1: Teaser. Representative robot-control tasks in UniLab; "Uni" means unified cross-platform training. Teaser image rendered with MotrixSim.

***Abstract*—Simulation-based RL for contemporary robot control is increasingly organized around GPU-resident simulation: physics, rollout collection, and learning are placed on a single GPU-centric execution path. This paradigm has greatly improved training speed, but it has also encouraged a default assumption that efficient training requires physics to reside on the GPU. We revisit this assumption. Our view is that, in simulation-dominated robot control, the essential question is not which processor runs physics, but whether simulation throughput, policy learning, and runtime synchronization form an efficient end-to-end loop. We present UniLab, a heterogeneous CPU-simulation / GPU-learning architecture that decouples CPU-parallel simulation from GPU policy updates through a unified runtime for data movement, buffering, and synchronization. UniLab is implemented as a complete and extensible training system using MuJoCoUni and MotrixSim CPU-batched physics backends, supporting PPO, FastSAC, FlashSAC, and APPO. On representative simulation-based robot control tasks, UniLab improves end-to-end training efficiency by 3–10× under the same hardware configuration, while reducing dependence on the NVIDIA CUDA-based software stack and supporting cross-platform execution on the Apple macOS platform and the AMD ROCm and Intel XPU accelerator backends. These results show that GPU simulation is an effective path to efficient training, but not a necessary one, broadening the practical system choices available for robot RL training.**

I. Introduction

Training infrastructure has become a first-order factor in simulation-based robot RL: faster training reduces the wall-clock cost of a single experiment, shortens system and algorithm iteration cycles, and expands the range of tasks that can be studied under practical hardware budgets. The dominant answer in recent years has been clear: place physics simulation, rollout collection, and learning on a GPU-centric execution path; Isaac Gym, Isaac Lab, MuJoCo Playground, mjlab, ManiSkill3, and Genesis show that large-scale GPU-resident environment parallelism can greatly accelerate robot control training [1, 2, 3, 4, 5, 6]. This success has shaped the current community default that efficient training should be organized around GPU-resident physics, tying high-throughput experimentation to a narrower set of GPU-resident software environments.

Robot RL training, however, is a closed-loop system coupling data generation, policy updates, and synchronization constraints, not a simulator benchmark alone. In simulation-dominated tasks, end-to-end efficiency depends on simulation throughput, learner utilization, collector–learner synchronization, data movement and buffering overhead, and whether hardware is allocated to the stage that actually limits wall-clock time: the learner may wait for rollouts, collectors may wait for new parameters, and data movement or buffering may erase parallel gains. Whether physics runs on the GPU is therefore one design choice within a broader systems organization problem.

High-throughput environment execution is also possible outside GPU-resident physics. General RL systems have long used CPU-side vectorized or batched environments, and robot RL has precedents for CPU-distributed or CPU-parallel

simulation, including OpenAI's Rubik's-cube hand system and recent RaiSim-based locomotion work [7, 8, 9, 10, 11, 12, 13]. Algorithmic data dependencies further shape this organization: PPO preserves the strongest rollout/update synchronization constraint; APPO allows collection and learning to overlap while remaining close to the on-policy setting; and off-policy methods such as FastSAC and FlashSAC further relax the dependence of each update on trajectories from the latest policy [14, 15]. This ordering lets us study algorithms as synchronization regimes rather than as separate algorithmic contributions: PPO tests whether CPU simulation can sustain strictly synchronized training, APPO tests collector–learner overlap once synchronization is relaxed, and FastSAC/FlashSAC test the replay-based producer–consumer path. This motivates the systems question studied here: can CPU-side batched rigid-body simulation, GPU-side policy learning, and the runtime path between them form an efficient training loop?

This paper asks whether efficient simulation-based robot control training must rely on GPU-resident simulation. Our thesis is that simulation-dominated robot control training requires high-throughput, well-coordinated simulation-learning execution, rather than GPU-resident simulation itself. We focus on representative robot control tasks in simulation, leaving real-world RL and vision-dominated settings outside the scope of this paper.

We present UniLab, a heterogeneous CPU-simulation / GPU-learning training architecture. CPU-side MuJoCoUni [16] and MotrixSim [17] backends perform batched rigid-body simulation and data generation, GPU resources perform policy and value learning, and a unified runtime coordinates data movement, buffering, and synchronization. UniLab is a training-system organization rather than a new policy optimization algorithm; it is implemented as a complete and extensible training system with unified training and evaluation entrypoints and explicit task/backend interfaces, while supporting PPO, FastSAC, FlashSAC, and APPO in one framework.

Across representative simulated robot-control benchmarks, UniLab improves end-to-end training efficiency by 3–10× on the same single-GPU/single-CPU workstation, while reducing dependence on the NVIDIA CUDA-based software stack and supporting execution on Apple macOS, AMD ROCm, and Intel XPU backends. Our contributions are threefold:

Systems framing. We recast efficient robot RL training as a systems organization problem for the simulation-learning closed loop, rather than a consequence of GPU-resident physics alone.

Heterogeneous training architecture. We present UniLab, which connects CPU-batched physics backends, a GPU learner, data buffering, and parameter synchronization through a unified runtime, while supporting PPO, FastSAC, FlashSAC, and APPO in one framework.

End-to-end evidence. We show 3–10× wall-clock gains across robot embodiments, control workloads, and practical algorithms, together with execution evidence on macOS, ROCm, and XPU backends.

TABLE I: Representative robot RL training systems.

System	Phys.	Batch	Coupling
`IsaacGym`	PhysX	GPU-C	GPU-sync
`IsaacLab`	PhysX	GPU-C	GPU-sync
`Genesis`	Taichi	GPU-C/M/R	GPU-sync
`MJP`	MJX	GPU-C	GPU-sync
`MjLab`	MJWarp	GPU-C	GPU-sync
UniLab	MJU/Mtx	CPU	H-async/sync

Note. GPU-C/M/R: GPU batched physics on CUDA/Metal/ROCm. GPU-sync: synchronized GPU simulation–learning; H-async/sync: CPU simulation with GPU learning. MJU/Mtx/MJP: MuJoCoUni/MotrixSim/MuJoCo_playground.

II. Related Work

A. GPU-resident robot learning.

The dominant systems path for efficient robot RL training has been to place physics simulation, rollout collection, and learning on a GPU-centric execution path [1, 18, 19]. MuJoCo provides a widely used foundation for robot control simulation [20], while Isaac Gym, Isaac Lab, MuJoCo Playground, mjlab, ManiSkill3, and Genesis have made large-scale GPU-resident environment parallelism a standard practice for robot learning [1, 2, 3, 4, 5, 6]. Table I summarizes these systems along the axes most relevant to this paper: physics execution path, simulation–learning organization, and algorithmic data dependency.

B. Systems lesson from GPU simulation.

The central lesson from GPU-resident systems is the integration of fast physics execution with tightly coupled rollout collection and learner updates. For on-policy methods such as PPO, this organization fits synchronized batched rollout/update cycles and has proven effective across robot-control workloads [14, 21, 22, 23, 24, 25, 26, 27, 28]. We adopt this systems lesson but separate the training-system principle from one hardware path: efficient training requires low-overhead data generation, learning, and synchronization, while GPU kernels are most effective for regular, dense, and statically shaped execution; dynamic active contact sets, sparse interactions, collision handling, contact solving, closed-chain or other constraint handling, and contact-rich manipulation all stress this execution model.

C. CPU-parallel environment execution.

High-throughput environment execution also has a history outside GPU-resident physics. In general RL, EnvPool, RLlib, Tianshou, and PufferLib use CPU-side vectorized, batched, or parallel rollout collection as core system components [7, 8, 9, 10]. Robot RL also has CPU-distributed or CPU-parallel precedents, including OpenAI's Rubik's-cube hand system and recent RaiSim-based locomotion work [11, 12]. These examples show that CPU-side environment parallelism is viable; UniLab asks whether, under the same hardware setting, modern CPU-batched simulation and a GPU learner can form an efficient end-to-end training path through a low-overhead runtime rather than only at extreme worker-cluster scale.

D. Replay-based robot-control acceleration.

Algorithmic data dependencies further shape the system organization. PPO is the practical default in many large-scale robot-training workloads, but its on-policy updates preserve strong synchronization between rollout generation and learner updates. Replay-based methods such as SAC and TD3 can reuse past experience and relax this dependence, while FastTD3, FastSAC, and FlashSAC show that this direction can accelerate high-dimensional robot control [15, 29, 30, 31, 32]. UniLab studies the complementary systems question: when data dependencies are relaxed, how can CPU simulation and GPU learning be coordinated to improve end-to-end wall-clock efficiency?

III. UniLab Architecture

This section describes UniLab as an end-to-end training loop that combines CPU-side batched rigid-body simulation, GPU-side policy and value learning, and a unified runtime for coordinating the data path between them.

A. Design Objective and Requirements

The design objective is to improve the efficiency of the full simulation-learning loop without requiring GPU-resident simulation. UniLab follows hardware roles: CPUs generate large-scale simulation data, GPUs perform dense learning updates, and the runtime minimizes coordination cost. This objective induces three requirements.

CPU-side simulation throughput. CPU-side batched rigid-body simulation must sustain enough throughput to continuously generate data for the workloads studied here.

Non-blocking GPU learning. The GPU learner should consume buffered experience rather than idling behind rollout generation.

Controlled runtime overhead. Data movement, buffering, and parameter synchronization must remain low-overhead so that the heterogeneous split does not degenerate into blocking handoffs.

B. UniLab Execution Architecture

Figure 2 summarizes the system organization: **CPU workers** generate trajectories or transitions, the **GPU learner** performs policy and value updates, and the **unified runtime** coordinates data movement, buffering, scheduling, and parameter synchronization.

Collection–update timing and overlap. UniLab supports both synchronized and loosely coupled collection–update timing. Standard PPO uses a synchronized rollout/update cycle. Our APPO implementation follows the asynchronous on-policy formulation described by Luo et al. [33]: the collector writes fixed-horizon rollouts, behavior-policy log probabilities, and bootstrap information into a shared ring buffer while continuing to step the next rollout on the CPU; the learner drains available rollouts and performs V-trace correction and PPO-style updates on the GPU, with the V-trace clipping values listed in Appendix C.42. CPU collection and GPU learning therefore overlap in wall-clock time with parameter synchronization near rollout boundaries. FastSAC and FlashSAC use replay-based timing: collectors insert transition batches into a shared replay buffer, while the learner performs multiple updates from device batches; both variants use the same optimized runtime path, which requests CPU replay packing and device transfer for the next batch one tick ahead so they overlap with current learner updates.

Figure 3 shows the FastSAC case, where collector-side work is staged ahead of learner computation and the main visible synchronization point is actor-weight handoff.

Runtime abstraction. The unified runtime lets synchronized and loosely coupled execution share one system stack, connecting robot assets, task configurations, simulation backends, and learning algorithms through explicit interfaces.

C. CPU Physics Backends and Task Interface

Batched CPU physics. UniLab realizes CPU-side throughput through backend-native batched environment execution: CPU workers advance environments at batch granularity and generate trajectories or transitions for the downstream learner.

Backend contract. The current system connects two practical CPU-side simulation backends under a shared runtime contract. MuJoCoUni provides a CPU-batched MuJoCo runtime backend [16]; the MotrixSim backend maps the same task and runtime contract onto the MotrixSim physics and rendering stack [17].

Task and randomization interface. This contract covers task state, actions, observation-related data, reset and interval randomization hooks, terrain context, and playback capabilities, allowing physical parameters, observation perturbations, and task-condition changes to be scheduled by the training system rather than scattered across task scripts.

This design separates physics semantics, determined by the backend model and solver, from training throughput, determined by batched execution, data movement, and runtime coordination; the same learner binding can also target macOS, ROCm, and XPU, with backend-dependent throughput evaluated in Section IV-E.

IV. Experiments

We evaluate three questions: whether CPU simulation provides enough throughput, whether heterogeneous CPU-simulation / GPU-learning improves end-to-end wall-clock efficiency, and whether the result is robust across task families and algorithms. The primary metric is end-to-end training efficiency; throughput measurements explain the mechanism.

A. Experimental Setup

Controlled comparisons use the same default Linux hardware: one NVIDIA RTX 4090 GPU, one AMD Ryzen 9 9950X3D CPU, and 64 GB of 4800 MT/s memory. Unless otherwise stated, UniLab results in the main experiments use the MuJoCoUni backend, while Apple macOS, AMD ROCm, and Intel XPU results are included as portability evidence. The task set spans locomotion, motion tracking, manipulation,

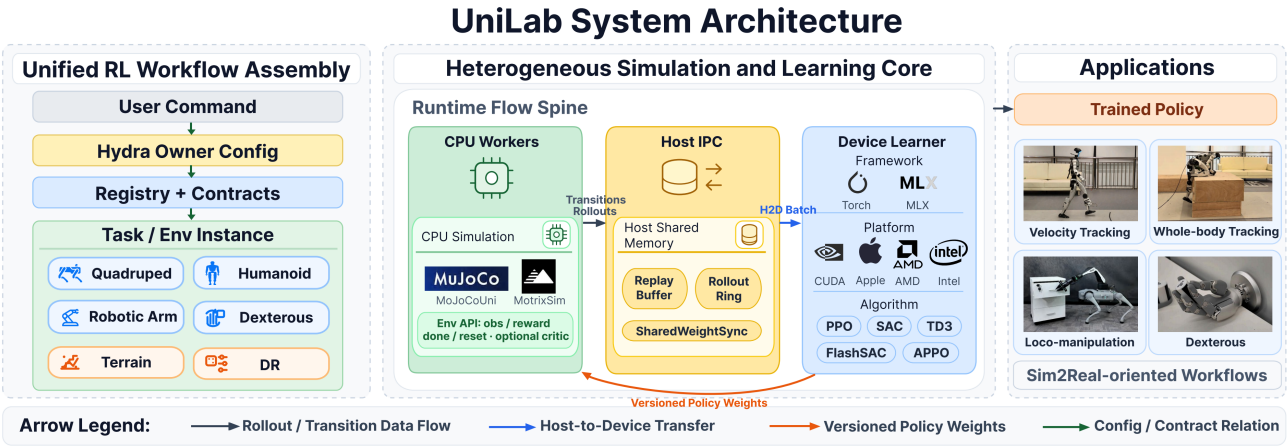


Fig. 2: UniLab system architecture. The figure shows the data, scheduling, and parameter-synchronization paths between CPU-side batched physics backends, the unified runtime, and the GPU learner.

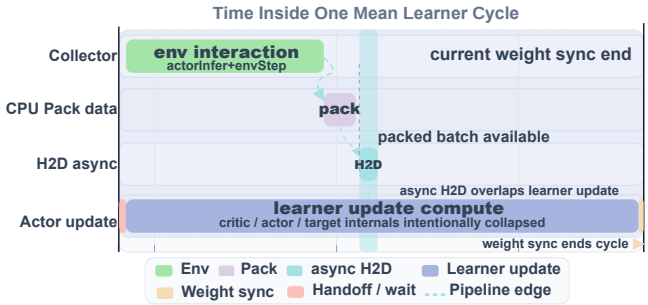


Fig. 3: Collection–update timing and overlap.

and manipulation-locomotion across quadruped, wheeled-quadruped, humanoid, and dexterous-hand / in-hand manipulation embodiments. We organize algorithms by their synchronization constraints: PPO is the strictly synchronized on-policy baseline, APPO is the near-on-policy case where rollout collection can overlap with learning, and FastSAC/FlashSAC provides replay-based producer–consumer off-policy evidence. For comparisons against external baselines, we use their public task-resolved configurations and align controllable factors including observation spaces, action spaces, rewards, sensor noise, and the main domain-randomization settings, while preserving each system's native execution details. The reported results therefore reflect practical system-level wall-clock performance under the same hardware setting on representative task configurations. Detailed experimental setup is provided in Appendix C.

B. Can CPU Simulation Provide Enough Throughput for Robot RL?

In common robot-RL training settings, CPU physics does not necessarily provide lower throughput than GPU-based simulation; its relative advantage is more pronounced in workloads with complex contact and dexterous manipulation. Figure 4 and Table II show that batched CPU simulation provides the simulator-side capacity required by the heterogeneous execution model over the environment counts studied here.

TABLE II: CPU env-step throughput (10^3 steps/s) by task and chip.

	Go2		G1		Hand	
Chip	MJ	Mtx	MJ	Mtx	MJ	Mtx
A18 Pro	55.7	122.9	28.4	18.1	183.9	134.1
M5 Max	288.0	797.8	178.8	127.7	1118.4	982.9
R9-8945HX	246.2	704.2	154.6	113.6	434.1	542.2
TR-9980X	915.9	2662.7	517.9	410.4	1991.5	2622.6
i7-11800H	82.1	162.0	34.7	23.8	176.8	151.6
Xeon 8558	1002.4	847.2	424.6	379.5	2566.3	397.7

Note. Values are 10^3 env steps/s; MJ = MuJoCoUni backend; Mtx = MotrixSim backend.

End-to-end training gives complementary evidence. In Figure 5(a), GPU-resident MjLab and CPU-step / GPU-learner UniLab achieve comparable efficiency on the same PPO task (Time Usage: 120.5/111.4 min vs. 109.3/109.2 min). Since synchronized PPO leaves little opportunity to hide rollout latency, PPO serves here as the strict synchronization stress test: this result indicates that CPU simulation is not the dominant bottleneck for this workload.

C. Can CPU-Sim / GPU-Learn Improve End-to-End Efficiency?

Given sufficient CPU-side throughput for strictly synchronized PPO, the next question is whether heterogeneous organization translates into end-to-end gains as data dependencies become looser. APPO remains near the on-policy regime but overlaps rollout collection with learning through correction; FastSAC/FlashSAC organize data generation as replay-based producer–consumer paths. Once the runtime decouples the learner from the collector, these more loosely coupled settings obtain 3–10× improvements in end-to-end training efficiency across multiple robot control tasks.

Figure 5 spans humanoid, motion-tracking, and dexterous-in-hand manipulation tasks and follows the progression from PPO to APPO and replay-based FastSAC/FlashSAC, indicating that the gain comes from learner–collector decoupling rather than a single task or algorithm configuration.

To further explain this gain, we add a learner-cycle ablation. Holosoma is the FastSAC codebase used here, and

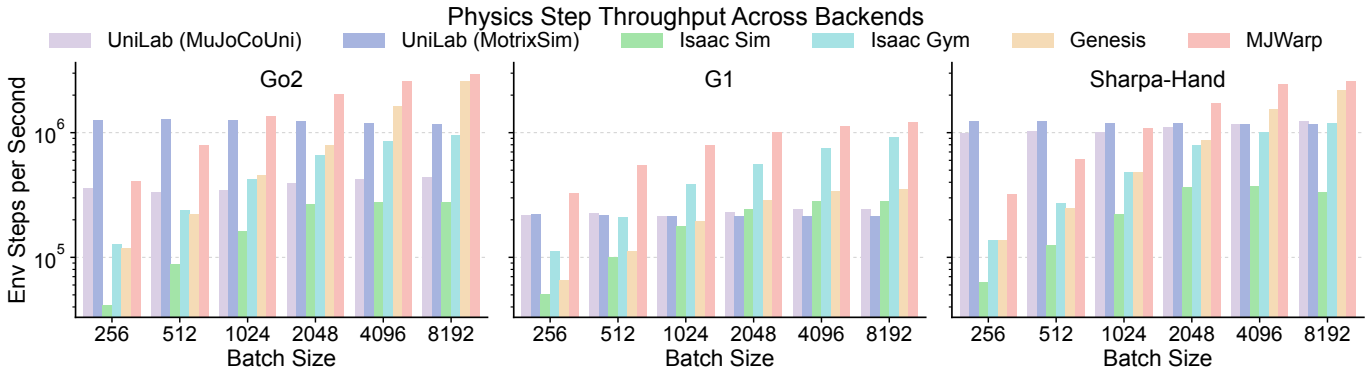


Fig. 4: CPU simulation throughput across representative robot control scenes. The figure establishes the simulator-side capacity that underlies the end-to-end training results.

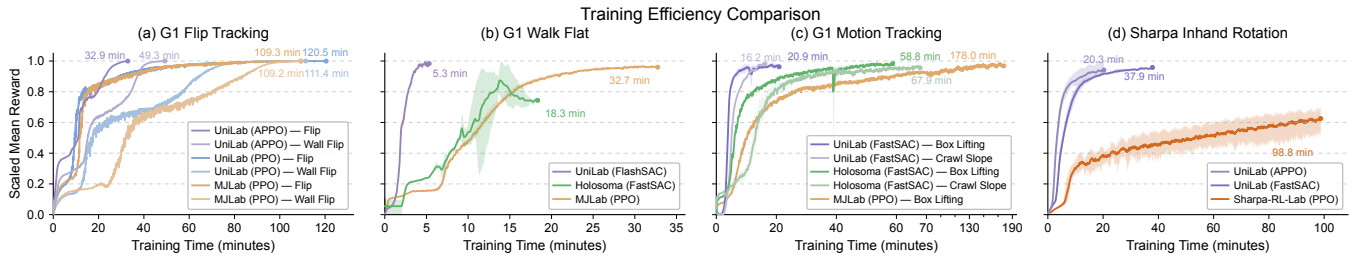


Fig. 5: End-to-end training efficiency on representative robot control tasks. Representative speedups: 3.3× on G1 Flip, 8.4× on G1 Walk Flat, and 11.0× on G1 Motion Tracking.

MjWarp denotes its MuJoCo Warp backend [34]; Figure 6 separates heterogeneous placement from runtime engineering alone: UniLab-MuJoCoUni completes collector work before the learner update ends, while attaching the same runtime to MjWarp lengthens the cycle because collector-side GPU simulation and learner updates share the same accelerator and contend for resources.

Figure 7 gives an overview of the six to-real experiments and complements the end-to-end simulation results with deployment-side coverage.

D. Dexterous In-Hand Rotation as a Systems Stress Test

SharpaWaveHand in-hand rotation adds contact-rich evidence beyond locomotion and motion tracking. The baseline starts from the public Sharpa-rl-lab PPO recipe on Isaac Lab, with object-scale randomization adjusted to match UniLab; it uses a fixed gravity direction with a built-in curriculum, whereas UniLab trains directly under randomized gravity directions without curriculum. In this task, the CPU MuJoCo version trains better, and UniLab reaches stronger HORA teacher policies within a shorter wall-clock budget; under the same-number friction setting, UniLab-SAC still reaches 1000+ reward in comparable time. The task uses a 22-DOF tactile hand to rotate a randomized free object and shows that UniLab supports dense simulation, stable learning, and different synchronization constraints in dexterous teacher training.

E. Cross-Platform Evidence

Finally, we report Apple macOS, AMD ROCm, and Intel XPU results to show practical trainability outside a single CUDA-centric setup, without claiming absolute throughput parity with the main Linux/CUDA workstation. Table III summarizes wall-clock training time across representative devices and tasks. Cross-platform execution is a practical consequence of the UniLab interface design.

V. Conclusion

This paper presented UniLab, a heterogeneous CPU-simulation / GPU-learning architecture for robot RL. By coordinating data movement, buffering, and synchronization through a unified runtime, UniLab improves end-to-end training efficiency by 3–10× across multiple robot embodiments, control workloads, and practical algorithms, while reducing dependence on the NVIDIA CUDA-based software stack and supporting Apple macOS, AMD ROCm, and Intel XPU backends. These results show that efficient training depends on high-throughput, well-coordinated simulation-learning execution, rather than requiring physics to reside on the GPU; UniLab therefore provides a systems counterexample showing that the design space for efficient training is broader than the current GPU-centric default suggests.

VI. Discussion

Our claim is not that GPU-resident simulation is obsolete. GPU simulation may remain preferable when simulator throughput is no longer the bottleneck or when larger

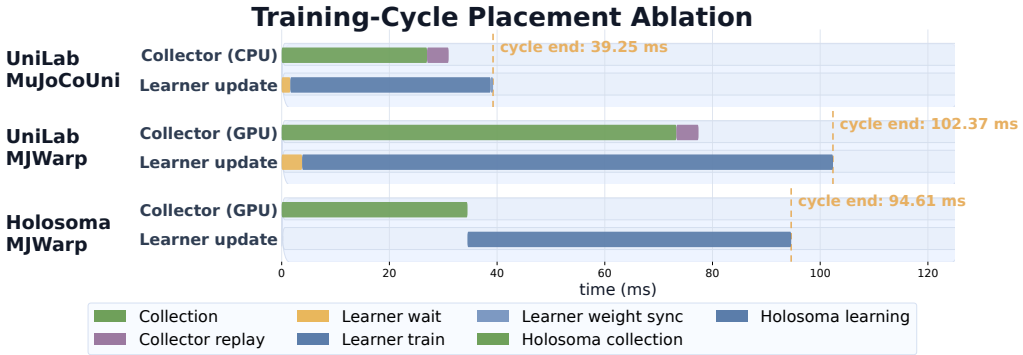


Fig. 6: Training-cycle placement ablation. Holosoma is the FastSAC codebase used here, and MjWarp is its MuJoCo Warp backend. The figure compares where simulation collection and learning are placed during one learner cycle.

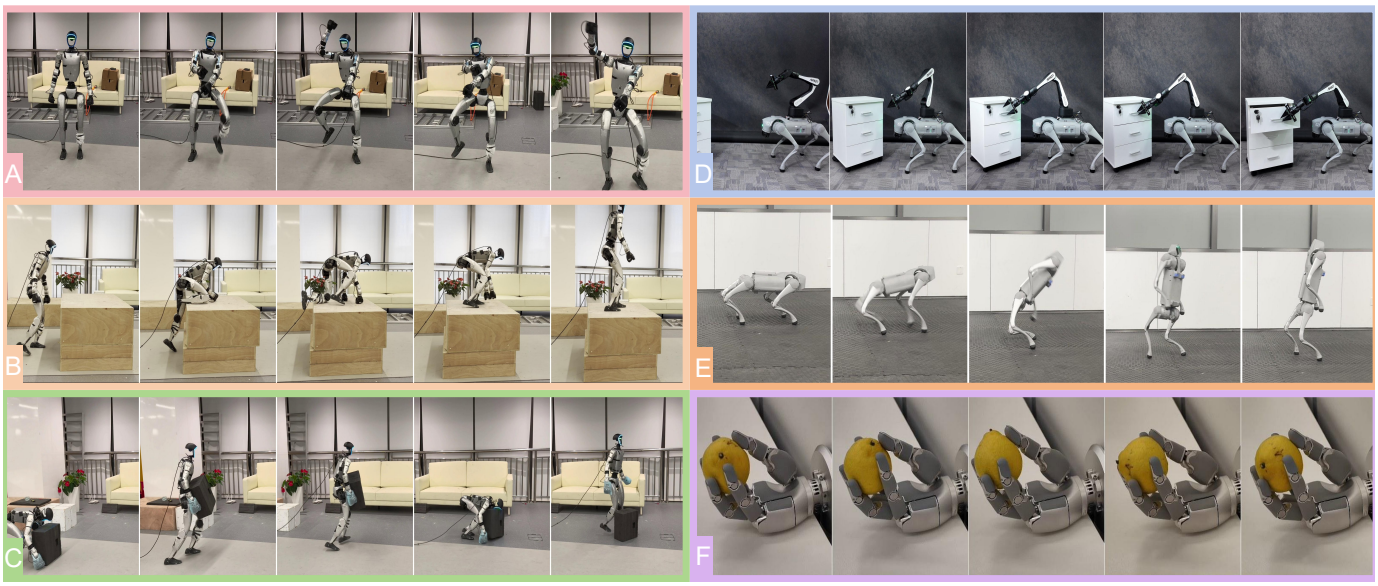


Fig. 7: To-real experiment overview across six real-robot tasks.

TABLE III: Wall-clock training time (min.).

Device	FastSAC/G1 WBT	FastSAC/G1 Walk	FlashSAC/Go2 Joy.	PPO/G1 Flip
RTX 4090 (Baseline)	58.8	18.3	6.0	109.0
RTX 4090 + AMD 9950X3D	18.5	3.0	1.1	16.4
AMD 8060S + AMD AI MAX 395	33.6	9.4	4.2	19.6
M5 Max	75.0	18.8	4.5	16.8

accelerator-rich configurations are a better fit. UniLab broadens the design space for simulation-dominated robot control.

The speed of a GPU-centric stack comes from two coupled designs: simulation, rollout collection, and learning share a low-overhead execution path, while the physics backend is organized as GPU-friendly parallel computation. The former is a training-system organization principle; the latter is one hardware path for realizing it. This path is effective for regular, dense, and statically shaped computation, but dynamic contacts, sparse interactions, collision handling, and constraint solving can increase backend engineering pressure and make implementations depend more on specialization, buffer tuning, and static-allocation assumptions. Thus, this paper does not challenge the value of GPU simulators; it challenges the necessity claim that efficient robot RL training must use GPU-resident physics.

VII. Limitations

The main limitations follow from three assumptions. First, UniLab is most advantageous when training is simulation-dominated and simulation can be meaningfully decoupled from learning; on strictly synchronized pipelines or vision-based workloads dominated by rendering, perception, and representation learning, CPU/GPU decoupling may not hide the dominant cost and may therefore yield smaller gains. Second, our claim concerns end-to-end training efficiency in a controlled single-CPU/single-GPU setting, not absolute peak throughput at extreme scale; multi-GPU or larger distributed configurations may change the bottleneck and the hardware-

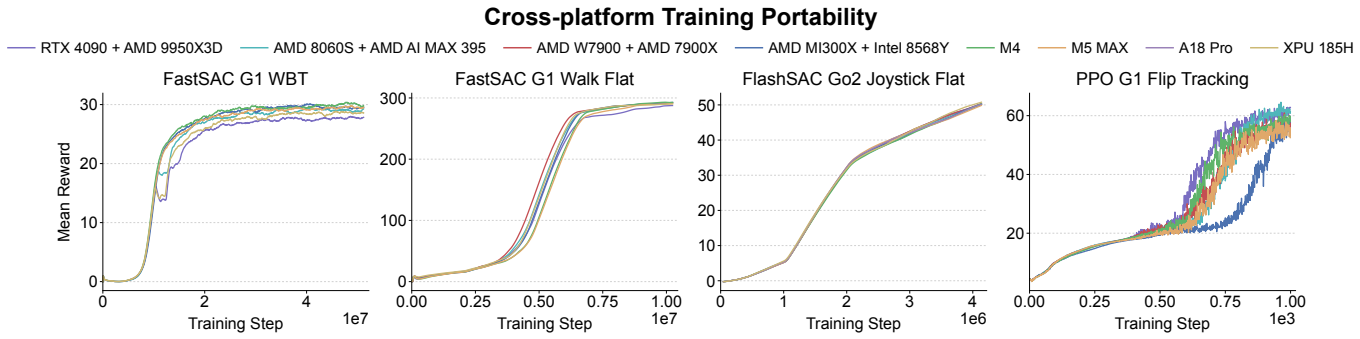


Fig. 8: Cross-platform training overview on representative devices. The figure shows training curves and final performance on different platforms, complementing Table III.

allocation tradeoff. Third, the current implementation focuses on rigid-body robot control rather than deformable objects, soft bodies, or fluids. Future work should extend the same runtime analysis to vision-dominated tasks, larger systems, and non-rigid physics to identify where the heterogeneous design fails and what scheduling, communication, or backend changes are needed.

References

[1] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[2] M. Mittal, P. Roth, J. Tigue, A. Richard, O. Zhang, P. Du, A. Serrano-Muñoz, X. Yao, R. Zurbrügg, N. Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

[3] K. Zakka, B. Tabanpour, Q. Liao, M. Haiderbhai, S. Holt, J. Y. Luo, A. Allshire, E. Frey, K. Sreenath, L. A. Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

[4] K. Zakka, Q. Liao, B. Yi, L. Le Lay, K. Sreenath, and P. Abbeel. mjlab: A Lightweight Framework for GPU-Accelerated Robot Learning. 2026. URL https://arxiv.org/abs/2601.22074.

[5] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T. kai Chan, Y. Gao, X. Li, T. Mu, N. Xiao, A. Gurha, V. N. Rajesh, Y. W. Choi, Y.-R. Chen, Z. Huang, R. Calandra, R. Chen, S. Luo, and H. Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *Robotics: Science and Systems*, 2025.

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

[7] J. Weng, M. Lin, S. Huang, B. Liu, D. Makoviichuk, V. Makoviychuk, Z. Liu, Y. Song, T. Luo, Y. Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022.

[8] Z. Wu, E. Liang, M. Luo, S. Mika, J. E. Gonzalez, and I. Stoica. RLlib flow: Distributed reinforcement learning is a dataflow problem. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/2bce32ed409f5ebcee2a7b417ad9beed-Paper.pdf.

[9] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022. URL http://jmlr.org/papers/v23/21-1127.html.

[10] J. Suarez. PufferLib 2.0: Reinforcement learning at 1m steps/s. *Reinforcement Learning Journal*, 6:1378–1388, 2025.

[11] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[12] Y. Kim, H. Oh, J. Lee, J. Choi, G. Ji, M. Jung, D. Youm, and J. Hwangbo. Not only rewards but also constraints: Applications on legged robot locomotion. *IEEE Transactions on Robotics*, 40:2984–3003, 2024.

[13] O. Pearce. Exploring utilization options of heterogeneous architectures for multi-physics simulations. *Parallel Computing*, 87:35–45, 2019.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. 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.

[16] Y. Jia and J. Wu. Mujocouni: Persistent batched runtime primitives for mujoco. *arXiv preprint arXiv:2605.24922*, 2026.

[17] Motphys Team. Motrixsim: A physics simulation engine for robotics and embodied ai, 2026. URL https://motrixsim.readthedocs.io/. Python binary package.

[18] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[19] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.

[20] E. Todorov, T. Erez, and Y. 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.

[21] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[22] G. B. Margolis and P. Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

[23] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid locomotion via reinforcement learning. *The International Journal of Robotics Research*, 43(4):572–587, 2024.

[24] Z. Wang, Y. Jia, L. Shi, H. Wang, H. Zhao, X. Li, J. Zhou, J. Ma, and G. Zhou. Arm-constrained curriculum learning for loco-manipulation of a wheel-legged robot. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10770–10776. IEEE, 2024.

[25] T. He, J. Gao, W. Xiao, Y. Zhang, Z. Wang, J. Wang, Z. Luo, G. He, N. Sobanbab, C. Pan, et al. Asap: Aligning simulation and real-world physics for learning agile humanoid whole-body skills. *arXiv preprint arXiv:2502.01143*, 2025.

[26] Z. Cao, L. Yan, Y. Zhang, S. Chen, J. Ma, T. Zhan, S. Fu, Y. Jia, C. Lu, and Y. Gao. Hiwet: Hierarchical world-frame end-effector tracking for long-horizon humanoid loco-manipulation. *arXiv preprint arXiv:2602.06341*, 2026.

[27] S. Bharthulwar, S. Tao, and H. Su. Staggered environment resets improve massively parallel on-policy reinforcement learning. *Advances in Neural Information Processing Systems*, 38:133342–133375, 2026.

[28] A. A. Shahid, Y. Narang, V. Petrone, E. Ferrentino, A. Handa, D. Fox, M. Pavone, and L. Roveda. Scaling population-based reinforcement learning with gpu accelerated simulation. *arXiv preprint arXiv*, 2404, 2024.

[29] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[30] Y. Seo, C. Sferrazza, H. Geng, M. Nauman, Z.-H. Yin, and P. Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.

[31] Y. Seo, C. Sferrazza, J. Chen, G. Shi, R. Duan, and P. Abbeel. Learning sim-to-real humanoid locomotion in 15 minutes. *arXiv preprint arXiv:2512.01996*, 2025.

[32] D. Kim, Y. Lee, M. Park, K. Kim, I. Nahendra, T. Seno, S. Min, D. Palenicek, F. Vogt, D. Kragic, et al. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. *arXiv preprint arXiv:2604.04539*, 2026.

[33] M. Luo, J. Yao, R. Liaw, E. Liang, and I. Stoica. Impact: Importance weighted asynchronous architectures with clipped target networks. *arXiv preprint arXiv:1912.00167*, 2019.

[34] Google DeepMind. Mujoco warp (mjwarp), 2026. URL https://mujoco.readthedocs.io/en/3.3.7/mjwarp/. Software documentation.

Appendix

TABLE OF CONTENTS

Appendix A. Off-Policy Replay Path Case Study 9
A.1 Baseline GPU-Cache SAC Path . . . 9
A.2 Sample-Before-Transfer Replay Pipeline . . . 10
A.3 Trace-Based Attribution . . . 10
A.4 Ablating the Path from GPU-Cache SAC to Sample-Before-Transfer . . . 11
A.5 Buffer and Communication Overhead . . . 12
A.6 What the Traces Do and Do Not Establish . . . 13

Appendix B. Domain Randomization Backends and Lifecycle 13
B.1 Runtime Lifecycle . . . 13
B.2 Backend Implementation . . . 13
B.3 Supported Randomization Families . . . 14
B.4 Implications for Cross-Backend Experiments . . . 14

Appendix C. Task and Algorithm Details 15
C.1 Training Curves . . . 15
C.2 Bidirectional Sim2Sim Cross-Backend Validation . . . 17
C.3 Task Specifications . . . 18
C.31 Locomotion . . . 18
C.32 Motion Tracking . . . 23
C.33 Manipulation-Locomotion . . . 25
C.34 Dexterous-Hand and In-Hand Manipulation . . . 26
C.4 Algorithm Hyperparameters . . . 28
C.41 PPO . . . 28
C.42 APPO . . . 28
C.43 SAC . . . 29

APPENDIX A. OFF-POLICY REPLAY PATH CASE STUDY

This section complements the main-paper system-attribution analysis with a detailed case study of the SAC replay-based execution path.

Unless otherwise stated, all timeline statistics in this section (Appendix A) are computed from Perfetto traces collected on the same A100 machine: one NVIDIA A100 80 GB PCIe GPU with driver 560.35.05 and CUDA 12.6, two Intel Xeon Gold 5320 CPUs with 104 logical CPU threads, and 188 GiB system memory. A learner cycle is measured from the end of one `learner/weight_sync_write` event to the end of the next such event; the first five cycles are discarded as warmup, and each retained cycle corresponds to 2048 environment steps. Reported per-cycle values are means over the retained cycles.

A.1 Baseline GPU-Cache SAC Path

We use SAC-A to denote the straightforward SAC baseline used for comparison in this case study, not a separate SAC algorithm. It corresponds to the GPU-cache replay path before the sample-before-transfer pipeline. This baseline is already a heterogeneous design: a CPU collector process runs a CPU actor synchronized from learner weights, advances the batched environment, and writes transitions into shared CPU replay storage. The learner process holds the SAC actor and critic networks on the accelerator and periodically publishes updated actor weights back to the collector. This organization already separates CPU simulation from GPU learning.

The remaining cost lies in the replay boundary. In the CUDA path, the learner maintains a device-side replay cache. When the learner samples, newly appended replay rows are lazily synchronized into this GPU cache, random indices are moved to the device, and the sampled batch is gathered from the cached replay tensors before SAC updates are performed. Thus, replay-cache maintenance and random replay access are part of the learner's hot update path. This increases GPU-resident replay storage and inserts replay-management work before the critic, actor, and target-network updates.

A.2 Sample-Before-Transfer Replay Pipeline

UniLab moves the replay boundary from the replay buffer to the sampled batch. The collector still performs CPU actor inference, environment stepping, and replay insertion. Once the learner requests the next training batch, the collector samples rows from a replay snapshot on the CPU and packs them into one of two shared pack slots. On CUDA, these pack slots are registered as pinned host-memory sources for asynchronous H2D transfer. A learner-side background H2D submit thread then transfers the packed batch into the cold GPU batch slot while the learner consumes the current hot slot.

This distinction matters for interpreting the memory path. The main replay storage remains CPU shared replay storage; the CUDA-specific pinned-memory path applies to the shared pack slots used as H2D sources. The learner consumes device-resident views from the hot GPU batch slot, executes SAC critic, actor, entropy-temperature, and target-network updates, and swaps the hot and cold slots at the next batch handoff.

Figure 9 places the baseline and double-buffer paths on the same retained learner-cycle time axis. In the figure, *env*, *replay*, *H2D*, *sync*, *stall*, and *gap* denote environment stepping, replay-buffer insertion, host-to-device transfer, actor-weight publication or consumption, waiting, and the delay from learner weight publication to the next first learner update, respectively. The key comparison is the change in replay ownership and timing: the baseline keeps replay sampling and lazy synchronization on the learner-side critical path, whereas the double-buffer path prepares the next sampled batch early enough to overlap CPU packing and H2D transfer with GPU learner updates.

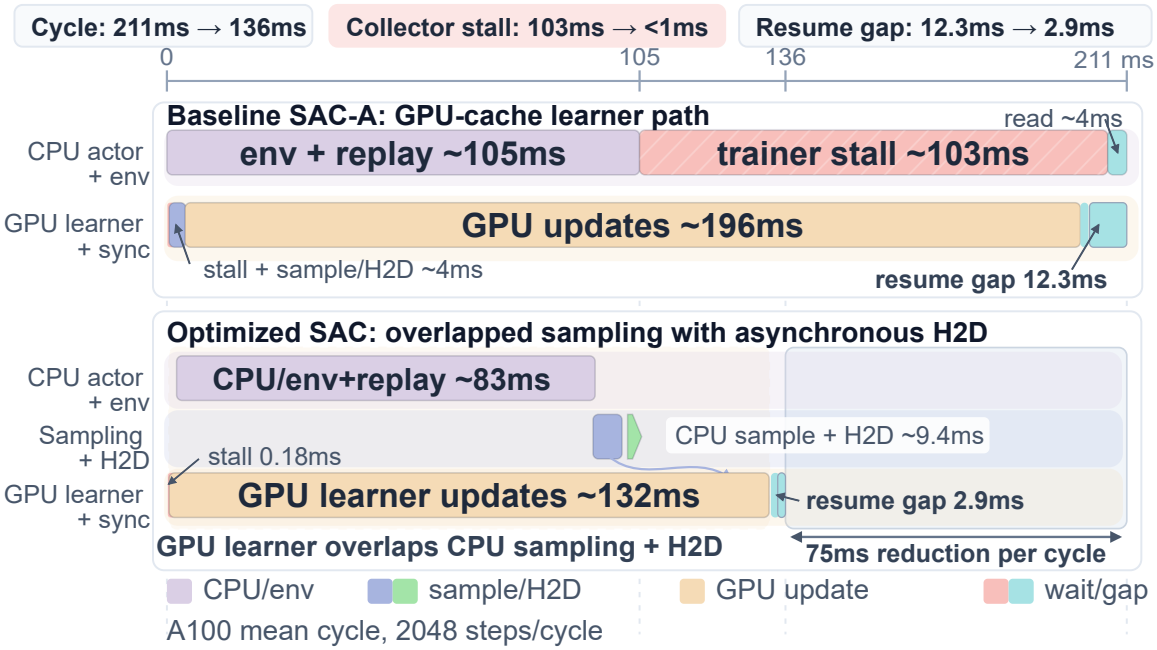


Fig. 9: Baseline SAC-A and optimized SAC learner-cycle timelines on A100. Durations are means per retained learner cycle using the cycle definition above; each cycle corresponds to 2048 environment steps. The optimized double-buffer path reduces cycle time from 211 ms to 136 ms, collector stall from 103 ms to below 1 ms, and resume gap from 12.3 ms to 2.9 ms.

A.3 Trace-Based Attribution

We analyze A100 Perfetto traces for the baseline GPU-cache SAC path and the UniLab double-buffer path. These traces provide mechanism and timing evidence: they show where replay sampling, H2D transfer, learner updates, and weight publication occur. Because learner-update kernels also differ in duration across traces, the attribution is interpreted together with the ablation below.

Figure 10 summarizes the optimized trace from four complementary views. Panel A compares standalone simulator throughput with throughput inside the SAC training pipeline; the reported efficiency is their ratio. Panels B–D use the retained learner-cycle definition above and report per-cycle means. In Panel B, *Lrn*, *Env*, *Pack*, *Inf*, *H2D*, *Add*, and *Sync* denote learner update, environment stepping, CPU replay sampling and batch packing, collector CPU actor inference, host-to-device batch transfer, replay-buffer insertion, and actor-weight publication or consumption. Panel C groups the cycle-level timing terms: *Cyc*, *Lrn*, *Col*, and *Ovl* denote learner-cycle duration, learner-update time, collector-active time, and their overlap. Panel D isolates collector-side actor inference by comparing it with total collector-active time.

In the traced 500-iteration window, the double-buffer path reduces training time from 107.50 s to 70.58 s, a 34.34% reduction in wall-clock time. After dropping the first five cycles, the mean learner cycle decreases from 211.31 ms to 136.10 ms. With

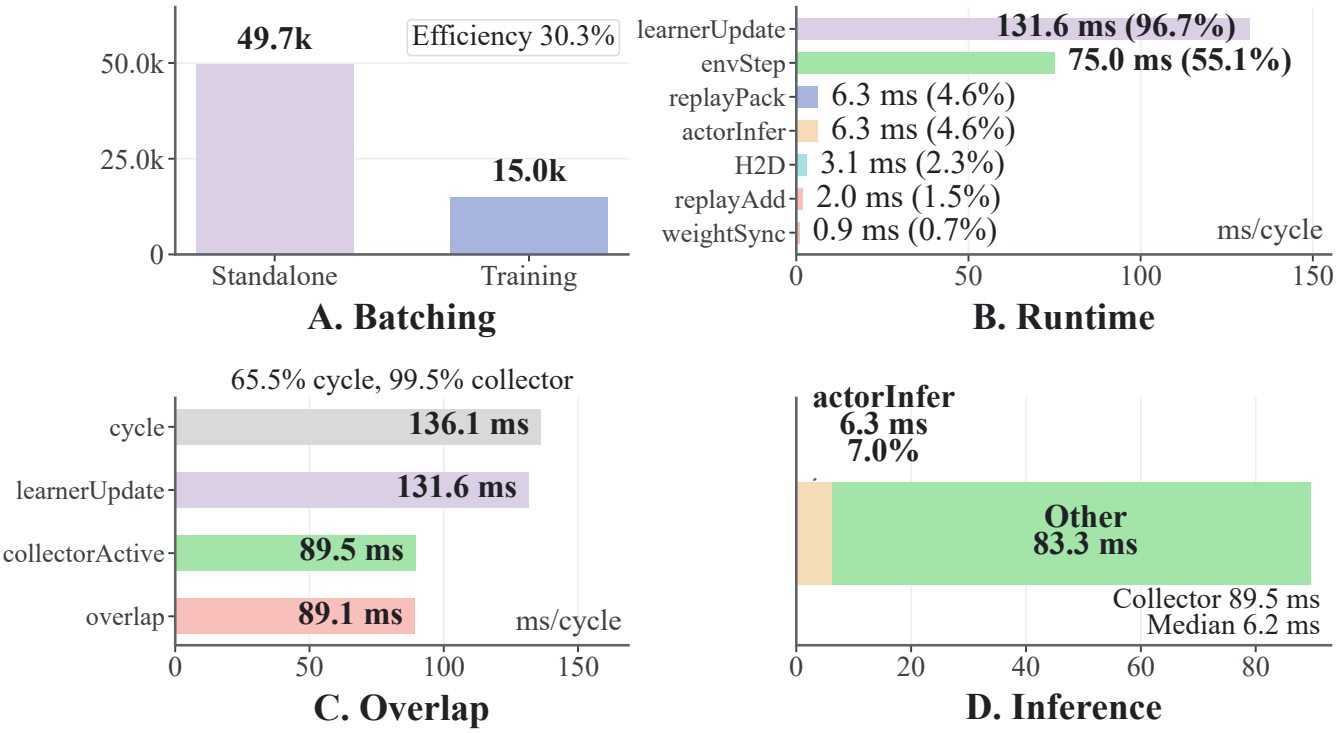


Fig. 10: System-attribution summary for the optimized SAC trace. Panel A reports batching efficiency, with *Eff.* defined as training-pipeline throughput divided by standalone simulator throughput. Panels B–D summarize runtime components, simulation-learning overlap, and collector-side CPU actor-inference cost.

2048 environment steps per learner cycle, this corresponds to an increase from 9.69k to 15.05k environment steps per second.

The clearest change is on the replay hot path. In the baseline trace, `learner/replay_sample` takes 3.64 ms on average and includes lazy replay synchronization, with `replay/h2d_lazy_sync` taking 1.88 ms on the CPU wrapper path and `gpu/replay_h2d_lazy_sync` taking 1.84 ms on the GPU event path. In the UniLab trace, learner-side replay consumption is reduced to 0.23 ms on average. Replay preparation still exists, but it is moved out of the learner hot path: CPU packing takes 6.30 ms, and GPU H2D transfer takes 3.13 ms, while 99.50% of collector-active time overlaps with learner updates. The remaining H2D handoff wait is about 0.055 ms per cycle.

A.4 Ablating the Path from GPU-Cache SAC to Sample-Before-Transfer

The trace-based attribution gives timing evidence for the replay-path change, but it does not by itself separate replay-data residency from transfer orchestration. We run a SAC replay-path ablation on the same A100 machine. The four variants preserve SAC's objective and update equations; only the replay boundary changes, moving from learner-side GPU-cache replay to sampled-batch transfer and then to the CPU-pinned double-buffer path.

The variants form a controlled migration chain. C is the old-SAC-like GPU-cache compatibility control: replay samples are still served through a learner-side GPU replay cache with lazy synchronization of newly appended rows and random gather from cached replay tensors. B keeps the same GPU-cache replay organization, but uses the modern ablation framework; its improvement over C therefore primarily reflects scheduling and runner-level overlap rather than a change in replay residency. A removes the GPU-cache resident replay component and moves the boundary to sampled-batch transfer, but it uses a synchronous/pageable transfer path rather than the full pinned asynchronous pipeline. The baseline keeps A's CPU-resident sampled-batch boundary and adds registered pinned pack slots, one-tick asynchronous H2D, and hot/cold GPU batch slots.

The figure reports four complementary measurements. Panel A reports wall-clock E2E time as means over three seeds, with sample-standard-deviation error bars. Panel B reports learner-cycle medians, with diamonds marking p95 cycle time. Panel C focuses on the learner-side replay boundary. We report *Replay sample mean* using the learner-side `learner/replay_sample` event: in GPU-cache variants, this event includes learner-side sampling, gather, and lazy replay-cache synchronization; in sampled-batch variants, it measures the learner-side batch handoff and consumption path, not the CPU packing work that is scheduled earlier. *GPU wait mean* reports learner-side boundary waiting before update computation, including waiting for data or replay-batch readiness, and should not be read as total kernel-level GPU idle time.

The black marks indicate the corresponding medians. Panel D reports the mean peak CUDA reserved memory and separates the replay-cache portion as the GPU-cache component. This component is present in C and B because the learner maintains a full GPU replay cache, and absent in A and the baseline because they keep replay CPU-resident and retain only sampled GPU batch slots.

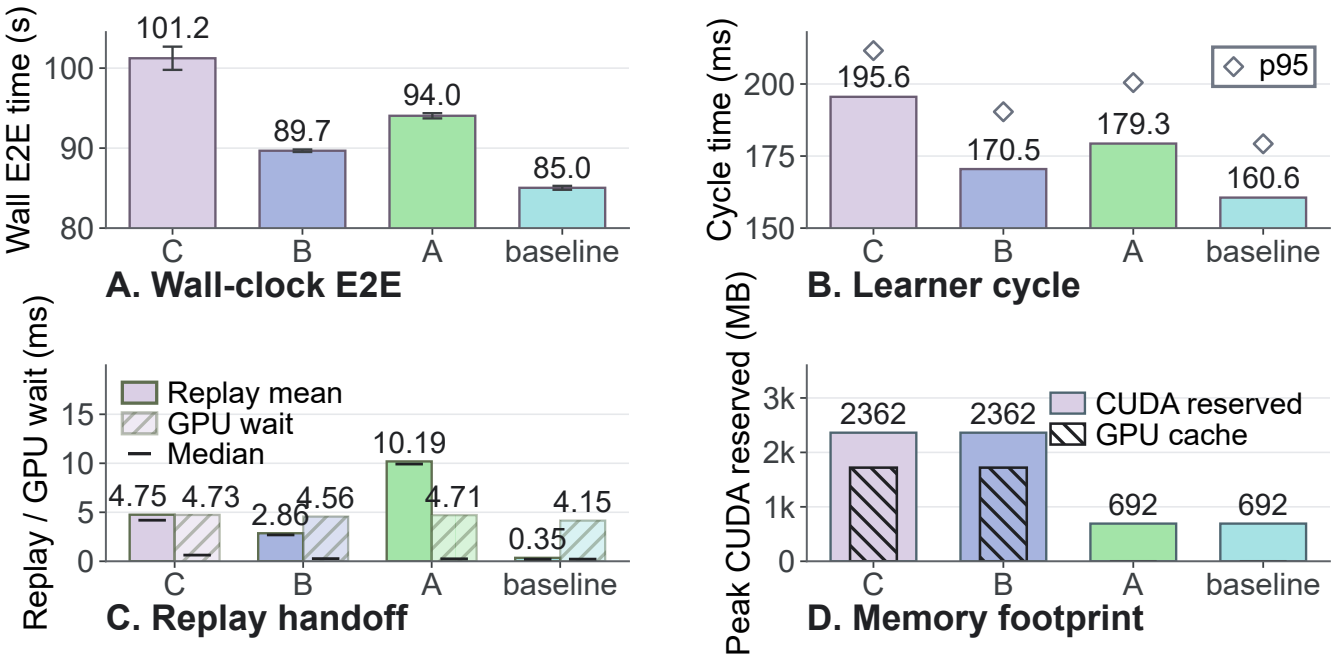


Fig. 11: C-to-baseline ablation for the SAC replay path. Wall-clock E2E bars are three-seed means with sample-standard-deviation error bars. Learner-cycle bars are medians, with diamonds marking p95 cycle time. Panel C reports learner-side replay-sample and boundary-wait statistics; Panel D reports peak CUDA reserved memory and the measured GPU-cache component.

With these definitions, Figure 11 shows that the variants affect different metrics for different reasons. Moving from C to B improves wall-clock time within the GPU-cache family, while CUDA reserved memory remains unchanged; this is consistent with a scheduling improvement rather than a memory-residency change. Moving from B to A removes the measured GPU-cache component and reduces peak CUDA reserved memory from 2362 MB to 692 MB, but the synchronous/pageable sampled-batch handoff becomes visible on the learner boundary, increasing replay-sample time and hurting E2E time. Moving from A to the baseline keeps the low-memory CPU-resident replay design and changes the transfer mechanism instead: pinned shared pack slots, one-tick asynchronous H2D, and hot/cold GPU slots reduce learner-side replay consumption from 10.19 ms to 0.35 ms and reduce wall time from 94.04 s to 85.04 s without reintroducing the GPU-cache component. Relative to C, the final baseline reduces wall time from 101.23 s to 85.04 s while also removing the measured GPU-cache footprint.

This ablation connects the trace-based attribution to the end-to-end result. The gain comes from relocating the replay-runtime boundary, not from changing the SAC loss. Replay work remains, but shifts in ownership and timing: the learner consumes ready device batches instead of maintaining and sampling a capacity-scaled GPU replay cache on the hot update path.

A.5 Buffer and Communication Overhead

Figure 12 combines a timeline overhead breakdown with an auxiliary replay-placement benchmark. Panel A breaks down the data movement, synchronization, and residual waiting observed inside the optimized SAC timeline. We separate the counted overhead into three groups. *Data movement* covers CPU-side replay sampling and batch packing, H2D submission, H2D transfer, and transfer-completion wait. *Weight sync* covers learner weight publication, device-to-host (D2H) weight copy, and collector-side weight read/update checking. *Boundary wait* covers collector and learner waiting at the cycle boundary. The figure also reports signal-ready context separately: the collector has already prepared the next batch and issued a ready signal, while the learner has not yet reached the next batch boundary. We treat this interval as scheduling and backpressure context rather than data-copy cost or learner-update waiting, so it is excluded from the counted overhead total.

In this trace, the counted data-movement, synchronization, and boundary-wait overhead total is 15.82 ms per cycle, or 11.62% of the 136.10 ms mean learner cycle. Data movement is the largest counted component at 10.07 ms per cycle, weight synchronization contributes 4.79 ms, and residual boundary waiting contributes 0.96 ms. The signal-ready interval is larger, but it is reported outside the headline total because it describes readiness and backpressure context rather than a direct data-copy or learner-wait cost.

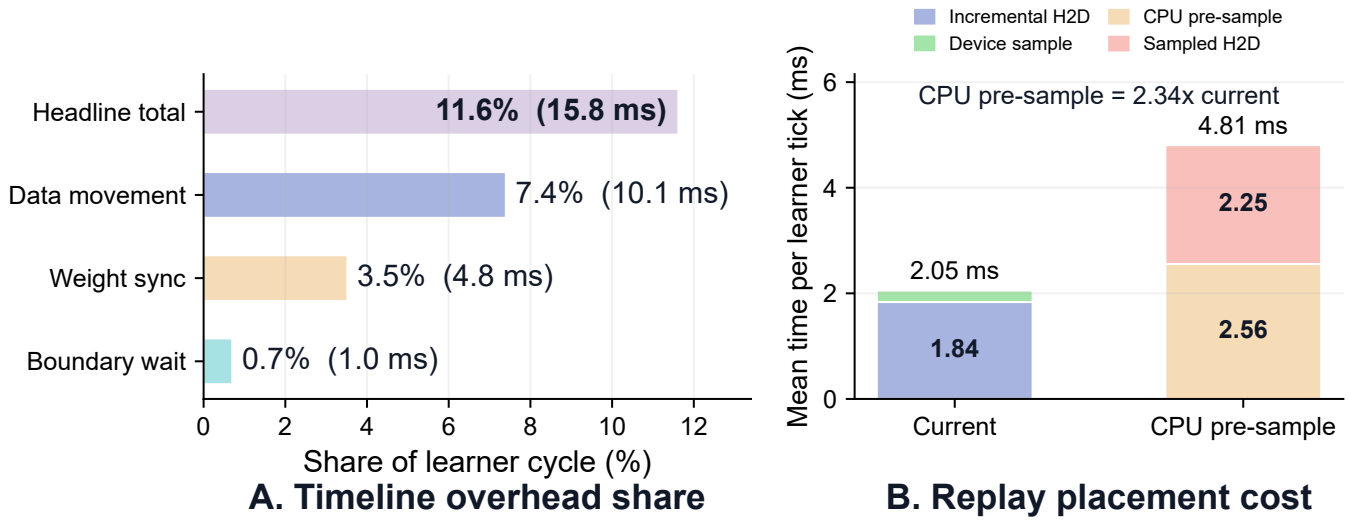


Fig. 12: SAC buffer and communication overhead. Values in Panel A are means per retained learner cycle from the optimized SAC timeline; Panel A groups counted data-movement, weight-synchronization, and boundary-wait overhead by share of the mean learner cycle, with signal-ready context shown separately. Panel B reports an auxiliary replay-placement benchmark comparing current incremental-transfer/device-sampling and CPU pre-sample plus sampled-batch H2D schemes.

Panel B provides an auxiliary replay-placement benchmark under the same configuration, separate from the retained-cycle accounting in Panel A. In this benchmark, the current placement combines incremental H2D with device-side random sampling and costs 2.05 ms per learner tick. A CPU pre-sample plus sampled-batch H2D placement costs 4.81 ms per learner tick, or 2.34× the current placement cost. This comparison is not part of the counted timeline overhead; instead, it shows why replay placement must be interpreted together with the overlap and scheduling structure of the full pipeline.

A.6 What the Traces Do and Do Not Establish

The traces establish the execution-path change and its timing consequences: replay preparation moves from learner-side GPU-cache sampling to collector-side CPU packing plus asynchronous H2D staging, new-batch preparation is almost fully overlapped with learner computation, and actor-weight publication remains an explicit synchronization boundary. They do not, by themselves, establish stronger claims about peak GPU memory, exact H2D volume reduction, or cross-algorithm generality; those claims require memory counters, byte counters for the baseline lazy-sync path, or corresponding TD3 / FlashSAC measurements.

Appendix B. Domain Randomization Backends and Lifecycle

Domain randomization in UniLab is implemented as a task/backend contract rather than as an algorithm-level feature. A task-owned `DomainRandomizationProvider` samples the quantities that are meaningful for the workload, while the simulator backend advertises which physical overrides it can apply. The runtime mediator, `DomainRandomizationManager`, validates this contract, applies cold-start model variants before backend materialization, injects reset payloads into sparse environment resets, and schedules interval perturbations before physics stepping. This keeps randomization tied to the same lifecycle that already controls state reset and batched simulation.

B.1 Runtime Lifecycle

Table V separates the lifecycle stages used by the current implementation. The important systems detail is that reset-time randomization is sparse: only the environments listed in `env_ids` receive a new state and a new randomization payload. Interval randomization is different: it is checked once per vectorized environment step, before the backend advances physics. Per-observation noise and command sampling are task-side operations and do not require backend-specific physical overrides, although they may depend on backend sensor reads that must complete first.

B.2 Backend Implementation

MuJoCoUni. MuJoCoUni implements reset-time randomization through `BatchEnvPool.reset(env_ids, initial_state, randomization=None)` which receives both the new physics state and an optional dictionary of model-field patches. Each payload has leading dimension `len(env_ids)`, so reset cost and randomization work scale with the number of environments that actually terminate. Fields that affect MuJoCo derived constants are patched before the reset/forward path and refreshed with `mj_setConst`; other fields are written directly. Geometry-level changes are handled

TABLE IV: Trace-based attribution summary for the SAC replay path. The table reports what can be supported directly by the A100 timeline traces and where additional evidence is needed.

Mechanism	Evidence from traces	Strength	Safe interpretation
End-to-end cycle reduction	Traced window decreases from 107.50 s to 70.58 s; post-warmup cycle decreases from 211.31 ms to 136.10 ms.	Strong	The double-buffer path is 1.52× faster over the traced window and 1.55× faster per post-warmup cycle.
Learner replay hot path	`learner/replay_sample` decreases from 3.64 ms to 0.23 ms on average.	Strong	Replay sampling no longer dominates the learner-side handoff before SAC updates.
Collector-side CPU packing	Optimized trace records CPU-side replay packing with 32768 sampled transitions per batch.	Strong	Random replay packing is performed on the CPU side before transfer.
Asynchronous H2D preparation	CPU pack takes 6.30 ms and GPU H2D takes 3.13 ms; 99.50% of collector-active time overlaps with learner updates.	Strong	Replay preparation is still present, but it is largely hidden behind GPU learner computation.
Hot/cold GPU batch slots	Optimized trace records alternating hot/cold batch-slot swaps between the two device slots.	Strong	The learner consumes one device batch while the next batch is prepared in the cold slot.
Pinned shared pack slots	Trace metadata reports registered pinned shared host memory, with pinned and direct-pinned flags enabled.	Medium–strong	The trace supports pinned H2D source slots; the implementation should be cited for the `cudaHostRegister` detail.
Actor-weight publication	`learner/weight_sync_write` decreases from 1.71 ms to 0.94 ms; weight-copy events remain visible.	Strong	Actor-weight publication remains a synchronous boundary, although it is small in this trace.
GPU replay memory footprint	The new trace transfers 56.36 MB for 32768 samples; two sampled batch slots are about 112.7 MB under this layout.	Estimate	This supports a memory-footprint estimate, not a measured peak-memory claim.

before runtime execution by compiling compatible model variants and assigning each vectorized environment to one variant before the pool is materialized.

MotrixSim. MotrixSim implements the same task/backend contract with MotrixSim-native override APIs. During `set_state`, the backend resets the selected data slice, clears staged body forces for those environments, applies init-time geometry-size overrides, applies supported reset randomization, writes the new DOF state, and runs forward kinematics. Mass and COM randomization use link mass and center-of-mass overrides. Friction, gravity, and actuator-gain randomization are conditional capabilities: they are enabled only when the loaded MotrixSim model exposes the corresponding override methods, and gain randomization requires all actuators to be position actuators. Object or geom-size variants are represented as per-environment size overrides rather than separate MuJoCo model binaries.

B.3 Supported Randomization Families

Table VI lists backend capabilities and current task-side coverage. The table distinguishes a backend's ability to apply a field from whether a particular task configuration enables that field. When a task requests reset terms that a backend does not support, the manager filters unsupported reset payload entries and logs the skipped terms; some task providers additionally fail validation when the term is required for that workload.

B.4 Implications for Cross-Backend Experiments

The shared contract lets a task express randomization once, but the effective randomization set is still backend-dependent. For cross-backend experiments, we therefore report the resolved task configuration separately from backend capabilities. A randomization item should be interpreted as active only when the task configuration enables it and the selected backend advertises support for the corresponding field. This distinction matters for fair comparison: for example, MuJoCoUni exposes a wider reset-field surface for inertial fields, whereas MotrixSim can match many common locomotion and manipulation settings through link, geom, gravity, actuator, and external-force override APIs when the loaded model supports them.

Lifecycle	Trigger	Owner	Randomized state
Backend initialization	DR init hook before backend `materialize()`	Task provider builds an init plan; backend materializes variants	Persistent model or geometry variants assigned per environment, such as object-scale variants expressed through `GeomSizeOverride`.
Sparse reset	Environment creation and any later reset of terminated or truncated `env_ids`	Task provider samples reset state and optional reset payload; backend applies supported fields	Initial pose, velocity, commands, cached object/grasp state, mass, COM, gravity, friction, actuator gains, and other supported physical fields.
Scheduled interval	Each vectorized `step`; active at the configured interval or when an equivalent task plan is non-empty	Task provider builds an interval plan; backend stages the perturbation for the upcoming physics step	Push forces and body-force perturbations. The current capability contract includes body velocity deltas, but neither backend advertises support for them.
Observation construction	Every task observation update	Task code	Actor observation noise, history/bias terms, and task-specific observation perturbations. These are backend-independent unless they require backend sensors to be read first.
Evaluation and playback	Same environment contract as training unless the configuration is changed	Training/evaluation entrypoint and task config	The backend does not reinterpret randomization for evaluation; training and evaluation share the same environment contract to avoid implicit behavioral differences between the two modes. Deterministic runs should disable the relevant task switches or set degenerate ranges and fixed seeds.

TABLE V: Domain-randomization lifecycle used by the current UniLab runtime.

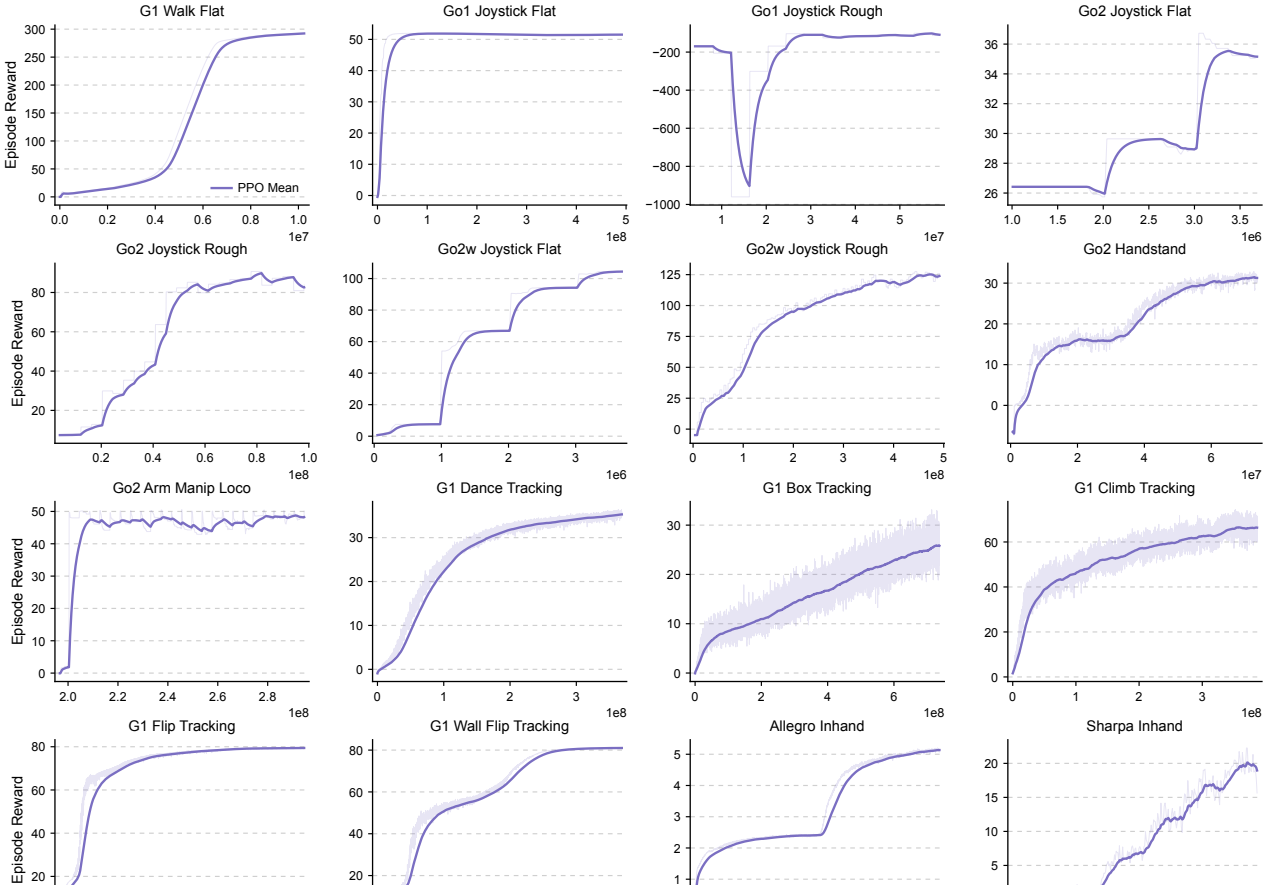
APPENDIX C. TASK AND ALGORITHM DETAILS

C.1 Training Curves

This subsection collects per-task training curves for the three on-policy and off-policy algorithm families used in the main paper. Each panel plots episode reward against environment steps; the curves are aggregated over seeds and smoothed with a fixed-width moving average. Rewards use the same task-side scales as in Section C.3, so absolute values are comparable across seeds for the same task but not across tasks. The task subsets shown for each algorithm match the per-task override tables in Section C.4: PPO is reported on all sixteen benchmark tasks, APPO on the six tasks with a registered APPO configuration, and SAC / FlashSAC on the five tasks with a replay configuration.

Family	Lifecycle	MuJoCoUni	MotrixSim	Current task coverage
Model or geometry variants	Init	Precompiled `MjModel` variants with per-env assignments.	Per-env geometry-size overrides applied to the MotrixSim data/model path.	Sharpa in-hand object-scale variants.
Initial state and task conditions	Reset	Backend receives the sampled `qpos`/`qvel`; task provider owns pose, velocity, command, grasp, motion-frame, and terrain-spawn sampling.	Same task-level reset contract after conversion to MotrixSim DOF layout.	Locomotion commands and spawn pose; motion-tracking reference frames; in-hand grasp/object resets.
Base/body mass	Reset	`base_mass_delta` and full `body_mass`.	Base-link mass delta and full link-mass override.	Locomotion, manipulation-locomotion, motion tracking, and dexterous-hand tasks where config enables mass DR.
Base/body COM	Reset	`base_com_offset` and full `body_ipos`.	Base-link COM offset and full link COM override.	Locomotion, motion tracking, and hand/object tasks where config enables COM DR.
Gravity	Reset	`gravity` payload.	Conditional gravity override support.	Motion-tracking and dexterous-hand configs that enable gravity or gravity-direction DR.
Contact friction	Reset	Full `geom_friction` payload.	Conditional collision-geom friction overrides; non-collision geoms must remain at defaults because the MotrixSim friction override API is only exposed on geoms with nonzero collision group or affinity.	Ground friction in manipulation-locomotion; foot friction in G1 tracking; object friction in dexterous-hand tasks.
Actuator gains	Reset	`kp` and `kd` payloads for position actuators.	Conditional per-actuator Kp/damping overrides; available only for all-position-actuator models.	Go2/G1 locomotion and tracking tasks when `randomize_kp` or `randomize_kd` is enabled.
Inertia and armature	Reset	`body_iquat`, `body_inertia`, and `dof_armature`.	Not advertised in the current capability set.	Armature randomization is used by manipulation-locomotion configs when enabled; body inertial tensors are backend capability rather than broad task coverage.
External perturbations	Interval	Push and arbitrary body-force payloads are staged through `xfrc_applied`.	Push forces are supported through link external force; arbitrary body force is conditional on link `add_external_force`.	Locomotion push perturbations; Sharpa object force perturbations.
Observation noise and biases	Observation step or reset	Task-side NumPy noise after backend sensor reads.	Same task-side path.	Locomotion actor noise; G1 tracking actor noise and reset biases; hand joint/contact observation noise where configured.

TABLE VI: Supported domain-randomization families and backend-specific limits.



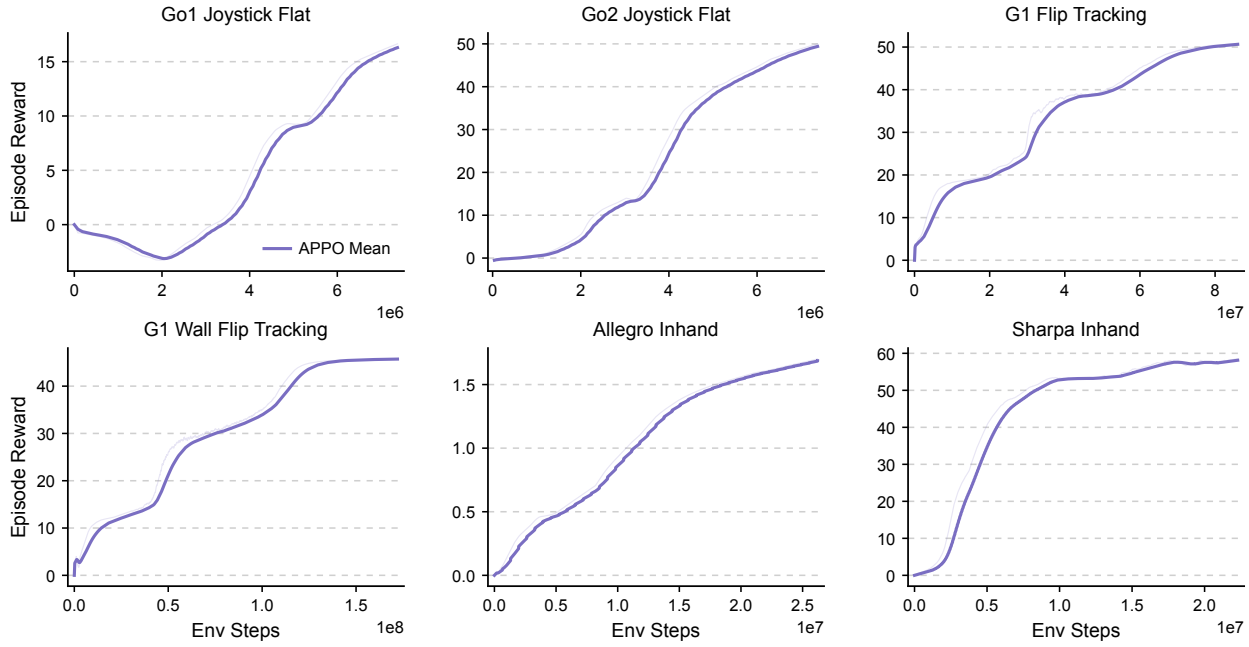


Fig. 14: APPO training curves for the six tasks with a registered APPO configuration: Go1 / Go2 Joystick Flat, G1 Flip and Wall Flip Tracking, Allegro Inhand, and Sharpa Inhand (HORA teacher).

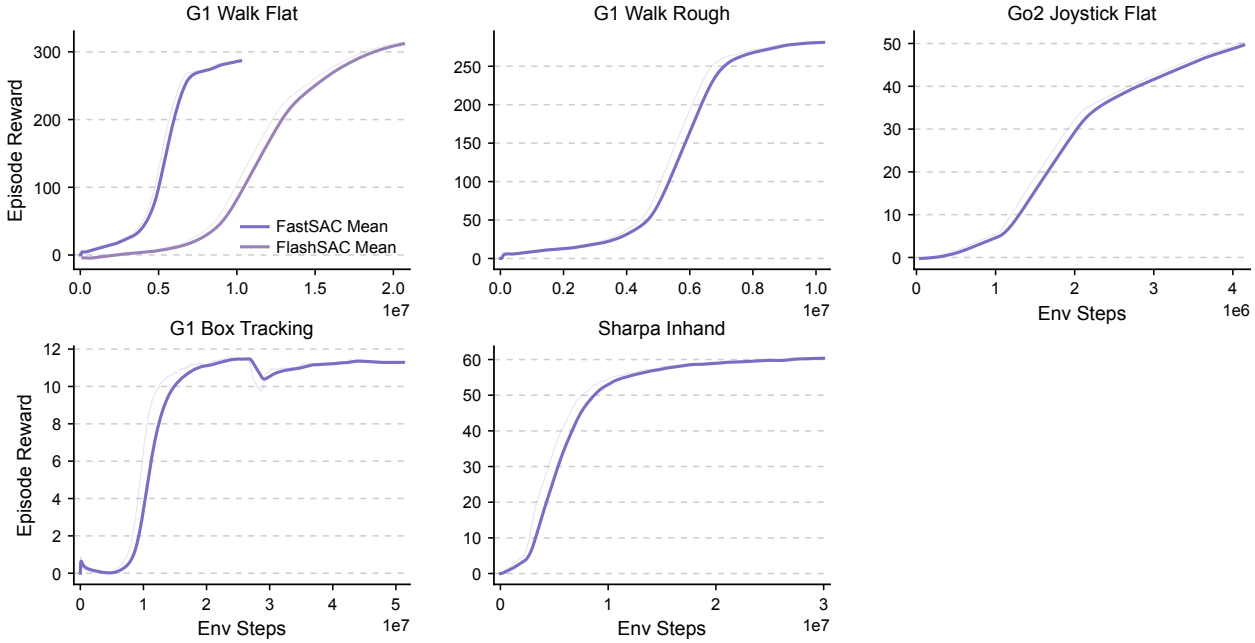


Fig. 15: FastSAC and FlashSAC training curves for the five tasks with a replay configuration. G1 Walk Flat reports both FastSAC and FlashSAC; the remaining four tasks (G1 Walk Rough, Go2 Joystick Flat, G1 Box Tracking, Sharpa Inhand) report FastSAC only.

C.2 Bidirectional Sim2Sim Cross-Backend Validation

a) Objective.: This subsection evaluates whether a policy trained against one simulation backend transfers to the other without retraining. We rollout each checkpoint on both MuJoCoUni and MotrixSim and compare its behaviour. Because each task uses backend-specific reward shaping at training time, the absolute reward values from different training backends are not directly comparable; what is meaningful is the change of each metric when the same policy is moved between backends, since this isolates the gap introduced by the simulator rather than by the policy.

b) Metrics.: We report three quantities per (policy, evaluation-backend) pair:

- **Mean episodic return:** the mean cumulative reward over 100 evaluation episodes, computed with the reward scale stored with the checkpoint at training time.
- **Success rate:** the fraction of the 100 episodes that finish without early termination.
- **MPJPE (m):** the mean per-joint position error against the reference trajectory, averaged over time and joints. MPJPE is only defined for motion-tracking tasks; locomotion rows report a dash.

c) Protocol.: All numbers come from zero-shot rollouts of a single trained checkpoint per policy: no fine-tuning, no retraining, and no per-backend adaptation. For each task we evaluate two checkpoints—one trained on MuJoCoUni and one trained on MotrixSim—and run each checkpoint on both backends, yielding four rows per task. The native rows (train backend = test backend) act as the reference; the cross rows (train backend $\neq$ test backend) measure the sim2sim gap. Each cell aggregates 100 episodes. Cross-backend evaluation uses the reward weights and normalization constants associated with the policy's training backend, ensuring that changes reflect simulator transfer rather than reward-definition changes.

TABLE VII: Bidirectional sim2sim cross-backend evaluation. For each task, the four rows correspond to: native MotrixSim (train=test=MotrixSim), forward transfer (MotrixSim→MuJoCoUni), native MuJoCoUni (train=test=MuJoCoUni), and reverse transfer (MuJoCoUni→MotrixSim). Mean return is comparable only within the four rows of a single task: reward scales are task- and training-backend-specific. A dash in the MPJPE column marks locomotion rows, where no reference trajectory exists.

Test type	Task	Train backend	Test backend	Mean return	Success	MPJPE (m)	Episodes
G1 Walk Flat (SAC)							
Native	G1 Walk Flat	MotrixSim	MotrixSim	408.38	1.00	—	100
Forward	G1 Walk Flat	MotrixSim	MuJoCoUni	405.46	1.00	—	100
Native	G1 Walk Flat	MuJoCoUni	MuJoCoUni	354.41	1.00	—	100
Reverse	G1 Walk Flat	MuJoCoUni	MotrixSim	354.00	1.00	—	100
G1 Motion Tracking, dance reference (SAC)							
Native	G1 Motion Tracking	MotrixSim	MotrixSim	45.26	1.00	0.0217	100
Forward	G1 Motion Tracking	MotrixSim	MuJoCoUni	45.17	1.00	0.0219	100
Native	G1 Motion Tracking	MuJoCoUni	MuJoCoUni	45.36	1.00	0.0197	100
Reverse	G1 Motion Tracking	MuJoCoUni	MotrixSim	45.34	1.00	0.0204	100
G1 Shuttle-Run Tracking (PPO)							
Native	G1 Shuttle-Run	MotrixSim	MotrixSim	45.80	1.00	0.0515	100
Forward	G1 Shuttle-Run	MotrixSim	MuJoCoUni	42.28	0.92	0.0568	100
Native	G1 Shuttle-Run	MuJoCoUni	MuJoCoUni	32.64	0.97	0.0519	100
Reverse	G1 Shuttle-Run	MuJoCoUni	MotrixSim	31.95	0.91	0.0532	100
G1 Wall Flip Tracking (PPO)							
Native	G1 Wall Flip	MotrixSim	MotrixSim	84.46	1.00	0.0447	100
Forward	G1 Wall Flip	MotrixSim	MuJoCoUni	79.46	1.00	0.0596	100
Native	G1 Wall Flip	MuJoCoUni	MuJoCoUni	80.61	1.00	0.0431	100
Reverse	G1 Wall Flip	MuJoCoUni	MotrixSim	77.44	1.00	0.0620	100

Across the four tasks, success rate stays at 1.00 for both locomotion and the two acyclic tracking tasks (dance and wall flip); only the shuttle-run policies drop noticeably (0.97/1.00 native vs. 0.92/0.91 cross), and MPJPE remains within $0.0030\,\mathrm{m}$ of the native baseline for the dance clip, within $0.0053\,\mathrm{m}$ for the shuttle run, and within $0.0189\,\mathrm{m}$ for the wall flip. These margins are small relative to the per-joint reference scale and indicate that the policies generalize across the two backends without backend-specific adaptation.

C.3 Task Specifications

This subsection lists the per-task observation space, action space, command and termination logic, domain randomization, and reward weights for every task evaluated in the main paper. Tasks are grouped by family: locomotion, motion tracking, manipulation-locomotion, and dexterous-hand in-hand manipulation. When MuJoCoUni and MotrixSim share a value it is reported once; backend-specific differences are called out explicitly.

1) Locomotion:

a) Go1 Joystick Flat.: `Go1JoystickFlat` runs on the flat Go1 scene with simulation step $\Delta t_{\mathrm{sim}} = 0.01\,\mathrm{s}$, control step $\Delta t_{\mathrm{ctrl}} = 0.02\,\mathrm{s}$, maximum episode $20\,\mathrm{s}$, and initial base position $(0, 0, 0.34)$.

Observation space. The actor observation is 49-dimensional:

$$o_t = [\omega_t,\, -g_t,\, q_t - q_{\mathrm{default}},\, \dot{q}_t,\, a_{t-1},\, c_t,\, \phi_t], \tag{1}$$

where $\omega_t \in \mathbb{R}^3$ is the body-frame gyro, $g_t \in \mathbb{R}^3$ is the up-vector sensor, $q_t - q_{\text{default}} \in \mathbb{R}^{12}$ is the joint-position offset, $\dot{q}_t \in \mathbb{R}^{12}$ is joint velocity, $a_{t-1} \in \mathbb{R}^{12}$ is the previous action, $c_t \in \mathbb{R}^3$ is the velocity command, and $\phi_t \in \mathbb{R}^4$ is the four-leg gait phase. The critic observation appends local linear velocity, giving 52 dimensions.

Action space. The action is a 12-dimensional joint-position offset. The environment maps policy output a_t to actuator targets by $q_t^{\text{cmd}} = q_{\text{default}} + 0.25\, a_t$, using PD gains $K_p = 35.0$, $K_d = 0.5$.

Commands and termination. The velocity-command range is $[(-0.6, -0.4, -0.8), (1.0, 0.4, 0.8)]$. An episode terminates early when the up-vector z component satisfies $g_t^z \leq 0.5$. Gait frequency is 2 Hz.

Domain randomization. Reset-time domain randomization is applied as listed in Table VIII.

TABLE VIII: Domain randomization for `go1_joystick_flat`.

Item	MuJoCoUni
Velocity-command range	$[(-0.6, -0.4, -0.8), (1.0, 0.4, 0.8)]$
`randomize_kp`	true (default)
`randomize_kd`	true (default)
Kp/Kd multiplier range	$[0.9, 1.1]$

Reward design. The reward is $\Delta t_{\text{ctrl}} \sum_i w_i r_i$. Table IX lists the active reward scales.

TABLE IX: Reward terms for `go1_joystick_flat`.

Term	MuJoCoUni
Linear velocity tracking	1.0
Yaw angular velocity tracking	0.2
Vertical linear velocity	-5.0
Roll/pitch angular velocity	-0.1
Base height	-100.0
Action rate	-0.005
Joint deviation from default	-0.1
Contact phase agreement	0.24
Swing-foot height	4.0

The velocity-tracking terms use $\exp(-e^2/\sigma^2)$ with $\sigma = 0.25$; the base-height penalty uses a target of 0.3 m; the swing-foot term uses $\exp(-e_z^2/0.01)$ gated by the swing phase ($\phi_i \geq 0.6$).

b) Go1 Joystick Rough.: `Go1JoystickRough` adds procedurally generated terrain to the Go1 quadruped. The configuration is identical across the two backends. Simulation step $\Delta t_{\text{sim}} = 0.01$ s, control step $\Delta t_{\text{ctrl}} = 0.02$ s, maximum episode 20 s.

Observation space. The actor observation is 45-dimensional (policy group):

$$o_t = [0.25\omega_t,\ -g_t,\ c_t,\ q_t - q_{\text{default}},\ 0.05\dot{q}_t,\ a_{t-1}], \tag{2}$$

where gyro and joint velocity are pre-scaled. The critic observation is 48-dimensional (adds base linear velocity) plus a height-scan vector (default 187 points from an 11×17 grid around the robot base).

Action space. The action is 12-dimensional with per-joint scaling: hip joints use `hip_action_scale`$= 0.125$, non-hip joints use `non_hip_action_scale`$= 0.25$. PD gains $K_p = 35.0$, $K_d = 0.5$. Actions are clipped to $[-100, 100]$.

Commands and termination. The velocity-command range is $[(-1, -1, -1), (1, 1, 1)]$ with heading command enabled and resampling every 10 s. Terrain is procedurally generated on an 8×8 m cell grid (6×6 cells, border width 20 m). Termination occurs when the robot moves more than 3 m beyond its terrain cell boundary. No gravity-based termination is used.

Domain randomization. Reset-time domain randomization is listed in Table X.

TABLE X: Domain randomization for `go1_joystick_rough`.

Item	MuJoCoUni	MotrixSim
`randomize_base_mass`	true	true
Added mass range	$[-1.0, 3.0]$	$[-1.0, 3.0]$
`random_com`	true	true
`randomize_kp`	true	true
Kp multiplier range	$[0.5, 2.0]$	$[0.5, 2.0]$
`randomize_kd`	true	true
Kd multiplier range	$[0.5, 2.0]$	$[0.5, 2.0]$
`push_robots`	true	true
Push interval (steps)	625	625
Max push force	$[1.0, 1.0, 0.5]$	$[1.0, 1.0, 0.5]$

Reward design. Table XI lists the active reward scales.

Tracking-style terms use $\sigma = 0.25$ and the base-height penalty uses a target of 0.33 m on both backends.

TABLE XI: Reward terms for `go1_joystick_rough`.

Term	MuJoCoUni	MotrixSim
Linear velocity tracking	3.0	3.0
Yaw angular velocity tracking	1.5	1.5
Vertical linear velocity	-2.0	-2.0
Roll/pitch angular velocity	-0.05	-0.05
Joint torques L2	-2.5×10^{-5}	-2.5×10^{-5}
Joint acceleration L2	-2.5×10^{-7}	-2.5×10^{-7}
Joint position limits	-5.0	-5.0
Joint power	-2.0×10^{-5}	-2.0×10^{-5}
Stand still	-2.0	-2.0
Hip position	-0.5	-0.5
Joint position penalty	-1.0	-1.0
Joint mirror	-0.05	-0.05
Action rate	-0.01	-0.01
Undesired contacts	-1.0	-1.0
Contact forces	-1.5×10^{-4}	-1.5×10^{-4}
Feet air time	0.5	0.5
Feet air time variance	-1.0	-1.0
Feet contact without cmd	0.1	0.1
Feet slide	-0.1	-0.1
Feet height body	-5.0	-5.0
Feet gait	0.5	0.5
Upward	1.0	1.0

c) Go2 Joystick Flat.: `Go2JoystickFlat` runs on the flat Go2 scene with simulation step $\Delta t_{\text{sim}} = 0.01\,\text{s}$, control step $\Delta t_{\text{ctrl}} = 0.02\,\text{s}$, maximum episode $20\,\text{s}$, and initial base position $(0, 0, 0.42)$.

Observation space. The actor observation is 49-dimensional:

$$o_t = [\omega_t,\, -g_t,\, q_t - q_{\text{default}},\, \dot{q}_t,\, a_{t-1},\, c_t,\, \phi_t], \tag{3}$$

where $\omega_t \in \mathbb{R}^3$ is the body-frame gyro reading, $g_t \in \mathbb{R}^3$ is the up-vector sensor value, $q_t - q_{\text{default}} \in \mathbb{R}^{12}$ is the joint-position offset from the default pose, $\dot{q}_t \in \mathbb{R}^{12}$ is joint velocity, $a_{t-1} \in \mathbb{R}^{12}$ is the previous action, $c_t \in \mathbb{R}^3$ is the velocity command, and $\phi_t \in \mathbb{R}^4$ is the foot phase. The critic observation appends local linear velocity, giving a 52-dimensional privileged observation. Observation noise uses the default level.

Action space. The action is a 12-dimensional joint-position command offset. The environment maps policy output a_t to actuator targets by

$$q_t^{\text{cmd}} = q_{\text{default}} + 0.25 a_t, \tag{4}$$

using the shared Go2 PD gains $K_p = 35.0$ and $K_d = 0.5$.

Commands and termination. The velocity-command range is $[(-0.6, -0.4, -0.8), (1.0, 0.4, 0.8)]$. An episode terminates early when the up-vector z component satisfies $g_t^z \leq 0.5$.

Domain randomization. Reset-time domain randomization is applied as listed in Table XII.

TABLE XII: Domain randomization for `go2_joystick_flat`.

Item	MuJoCoUni
Velocity-command range	$[(-0.6, -0.4, -0.8), (1.0, 0.4, 0.8)]$
`randomize_kp`	true
`randomize_kd`	true
Kp/Kd multiplier range	$[0.9, 1.1]$

Reward design. The reward is the control-step-scaled sum $\Delta t_{\text{ctrl}} \sum_i w_i r_i$. Table XIII lists the active reward scales.

Tracking-style terms use $\sigma = 0.25$ and the base-height penalty uses a target of $0.3\,\text{m}$. The swing-foot term rewards swing feet near $0.1\,\text{m}$ height; the contact term compares measured foot contact with the gait phase.

d) Go2 Joystick Rough.: `Go2JoystickRough` shares the same architecture as Go1 Joystick Rough (procedural terrain, height scan, heading command) but uses the Go2 robot model. The configuration is identical across the two backends. Simulation step $\Delta t_{\text{sim}} = 0.01\,\text{s}$, control step $\Delta t_{\text{ctrl}} = 0.02\,\text{s}$, maximum episode $20\,\text{s}$.

Observation space. Same structure as Go1 Joystick Rough: actor 45-dimensional (pre-scaled gyro, gravity, command, joint offset, joint velocity, last action), critic 48-dimensional plus height-scan (187 points).

Action space. 12-dimensional with `hip_action_scale`$= 0.125$, `non_hip_action_scale`$= 0.25$. PD gains $K_p = 35.0$, $K_d = 0.5$. Clip to $[-100, 100]$.

TABLE XIII: Reward terms for `go2_joystick_flat`.

Term	MuJoCoUni
Linear velocity tracking	1.0
Yaw angular velocity tracking	0.2
Vertical linear velocity	-5.0
Roll/pitch angular velocity	-0.1
Base height	-100.0
Action rate	-0.005
Joint deviation from default pose	-0.1
Contact phase agreement	0.24
Swing-foot height	4.0

Commands and termination. Velocity-command range $[(-1,-1,-1),(1,1,1)]$, heading command enabled, resampling every 10 s. Terrain: 8×8 m cells, 6×6 grid, border 20 m. Termination: terrain out-of-bounds (3 m buffer). No gravity-based termination.
Domain randomization. Identical to Go1 Joystick Rough (Table X): base mass $[-1,3]$ kg, COM offset, Kp/Kd $[0.5,2.0]$, push robots every 625 steps with force $[1,1,0.5]$.
Reward design. Same reward terms and weights as Go1 Joystick Rough (Table XI), with base-height target 0.33 m and tracking sigma 0.25.

e) Go2W Joystick Flat.: `Go2WJoystickFlat` is a wheeled-legged quadruped with 12 leg joints and 2 wheel joints. The configuration is identical across the two backends. Simulation step $\Delta t_{\text{sim}} = 0.01$ s, control step $\Delta t_{\text{ctrl}} = 0.02$ s, maximum episode 20 s.
Observation space. The actor observation is 53-dimensional:

$$o_t = [\omega_t,\, -g_t,\, q_t^{leg} - q_{\text{def}}^{leg},\, \dot{q}_t^{leg},\, \dot{q}_t^{wheel},\, a_{t-1},\, c_t], \tag{5}$$

where the leg joint offset and velocity are 12-dimensional each, wheel velocity is 2-dimensional, and actions include both leg (12) and wheel (2) outputs. The critic observation is 72-dimensional (adds linear velocity, motor control targets, and wheel control targets).
Action space. 14-dimensional: 12 leg joints with `action_scale`$= 0.5$ and 2 wheel joints with `wheel_action_scale`$=$ 10.0. Leg PD gains $K_p = 50.0$, $K_d = 1.5$; wheel uses velocity control with $K_d^{wheel} = 0.5$.
Commands and termination. Velocity-command range $[(0,0,-1),(1,0,1)]$ (forward and yaw only). An episode terminates early when $g_t^z \le 0.5$.
Domain randomization. Kp/Kd randomization is disabled. No other domain randomization is enabled in the flat variant.
Reward design. Table XIV lists the active reward scales.

TABLE XIV: Reward terms for `go2w_joystick_flat`.

Term	MuJoCoUni	MotrixSim
Linear velocity tracking	1.0	1.0
Yaw angular velocity tracking	0.75	0.75
Vertical linear velocity	-5.0	-5.0
Roll/pitch angular velocity	-0.1	-0.1
Base height	-100.0	-100.0
Orientation	-2.0	-2.0
Action rate	-0.005	-0.005
Joint deviation from default	-0.5	-0.5
Torques	-2.0×10^{-4}	-2.0×10^{-4}
Alive	0.5	0.5
Upward	1.0	1.0

Tracking-style terms use $\sigma = 0.25$ and the base-height penalty uses a target of 0.4 m on both backends.

f) Go2W Joystick Rough.: `Go2WJoystickRough` adds procedural terrain to the wheeled-legged Go2W. Simulation step $\Delta t_{\text{sim}} = 0.01$ s, control step $\Delta t_{\text{ctrl}} = 0.02$ s, maximum episode 20 s.
Observation space. The actor observation is 53-dimensional (pre-scaled gyro 0.25ω, gravity, command, leg joint offset, leg velocity $0.05\dot{q}$, last action including wheel). The critic observation is 56-dimensional (adds linear velocity) plus height-scan (187 points).
Action space. 14-dimensional: 12 leg joints with `action_scale`$= 0.5$ and 2 wheel joints with `wheel_action_scale`$=$ 10.0. Leg PD gains $K_p = 35.0$, $K_d = 0.5$; wheel $K_d^{wheel} = 0.5$. Clip to $[-100, 100]$.
Commands and termination. Velocity-command range $[(-1,-1,-1),(1,1,1)]$, heading command enabled, resampling every 10 s. Terrain: 8×8 m cells, 6×6 grid, border 20 m. Termination: terrain out-of-bounds (3 m buffer).
Domain randomization. Table XV lists the domain-randomization settings. Kp/Kd randomization is enabled under MotrixSim and disabled under MuJoCoUni.

TABLE XV: Domain randomization for `go2w_joystick_rough`.

Item	MuJoCoUni	MotrixSim
`randomize_base_mass`	true	true
Added mass range	$[-1.0, 3.0]$	$[-1.0, 3.0]$
`random_com`	true	true
COM offset x	$[-0.05, 0.05]$	$[-0.05, 0.05]$
`randomize_kp`	false	true
Kp multiplier range	$[0.5, 1.0]$	$[0.5, 1.0]$
`randomize_kd`	false	true
Kd multiplier range	$[0.5, 1.0]$	$[0.5, 1.0]$
`push_robots`	true	true
Push interval (steps)	500	625
Max push force	$[0.5, 0.5, 0.0]$	$[1.0, 1.0, 0.5]$
Push body	`base_link`	`base_link`

Reward design. Table XVI lists the active reward scales. MotrixSim adds an orientation penalty that MuJoCoUni does not use, and the two backends differ on the hip-position penalty weight.

TABLE XVI: Reward terms for `go2w_joystick_rough`.

Term	MuJoCoUni	MotrixSim
Linear velocity tracking	3.0	3.0
Yaw angular velocity tracking	1.5	1.5
Vertical linear velocity	-2.0	-2.0
Roll/pitch angular velocity	-0.05	-0.05
Orientation	/	-2.0
Joint torques L2	-2.5×10^{-5}	-2.5×10^{-5}
Joint acceleration L2	-2.5×10^{-7}	-2.5×10^{-7}
Wheel-joint acceleration L2	-2.5×10^{-9}	-2.5×10^{-9}
Joint position limits	-5.0	-5.0
Joint power	-2.0×10^{-5}	-2.0×10^{-5}
Action rate	-0.01	-0.01
Stand still	-2.0	-2.0
Hip position	-2.0	-0.5
Joint position penalty	-1.0	-1.0
Joint mirror	-0.05	-0.05
Upward	1.0	1.0

Tracking-style terms use $\sigma = 0.25$ and the base-height penalty uses a target of $0.4\,\mathrm{m}$ on both backends.

g) G1 Walk Flat.: `G1WalkFlat` is a 29-DOF humanoid locomotion task on the flat G1 scene. Simulation step $\Delta t_{\mathrm{sim}} = 0.01\,\mathrm{s}$, control step $\Delta t_{\mathrm{ctrl}} = 0.02\,\mathrm{s}$, maximum episode $20\,\mathrm{s}$, initial base position $(0, 0, 0.754)$.

Observation space. The actor observation is 98-dimensional:

$$o_t = [\omega_t,\, -g_t,\, q_t - q_{\mathrm{default}},\, \dot{q}_t,\, a_{t-1},\, c_t,\, \phi_t], \tag{6}$$

where joint offset, joint velocity, and last action are 29-dimensional each, and $\phi_t \in \mathbb{R}^2$ is the bipedal gait phase (left, right). The critic observation is 101-dimensional (adds base linear velocity).

Action space. 29-dimensional joint-position offset. Action mapping $q_t^{\mathrm{cmd}} = q_{\mathrm{default}} + s\, a_t$ with backend-dependent scale: MuJoCoUni uses $s = 0.25$, MotrixSim uses $s = 0.5$. PD gains $K_p = 50.0$, $K_d = 1.0$ (from G1 base config).

Commands and termination. Under MuJoCoUni the velocity-command range follows the task default; under MotrixSim it is fixed to $[(0.4, 0, 0), (0.7, 0, 0)]$ (forward walking only) with the gait phase initialized at the configured offset and the reset base-velocity limited to $0.05\,\mathrm{m/s}$. Termination occurs when body tilt exceeds the configured maximum or the base height drops below the configured minimum: MuJoCoUni uses 25° and $0.55\,\mathrm{m}$, MotrixSim uses 35° and $0.5\,\mathrm{m}$.

Domain randomization. The velocity curriculum is disabled. Under MotrixSim Kp/Kd randomization is additionally disabled. Observation noise is configured identically on both backends: noise level 1.0, joint-angle scale 0.01, joint-velocity scale 1.5, gyro scale 0.2.

Reward design. Table XVII lists the active reward scales. MotrixSim introduces several gait-shaping terms (feet phase contrast, feet phase contact, double-stance penalty, under-speed penalty, upper-body pose) that are not used under MuJoCoUni.

Shaping parameters used by the table above: velocity-tracking $\sigma = 0.25$, gait frequency $1.5\,\mathrm{Hz}$, feet-phase swing height $0.09\,\mathrm{m}$, feet-phase tracking $\sigma = 0.008$, base-height target $0.754\,\mathrm{m}$ (MuJoCoUni) / $0.765\,\mathrm{m}$ (MotrixSim). The 29-entry pose-weight vector is identical on both backends. Under MotrixSim, the gait reward is gated by a minimum forward speed of $0.05\,\mathrm{m/s}$.

TABLE XVII: Reward terms for `g1_walk_flat`.

Term	MuJoCoUni	MotrixSim
Linear velocity tracking	2.0	2.0
Yaw angular velocity tracking	0.2	0.25
Feet phase	1.0	1.2
Feet phase contrast	/	1.5
Feet phase contact	/	1.0
Feet double stance	/	-1.0
Under speed	/	-0.2
Upper body pose	/	-0.05
Vertical linear velocity	-1.0	-1.0
Roll/pitch angular velocity	-0.25	-0.2
Base height	-500.0	-120.0
Orientation	-5.0	-2.5
Action rate	-0.01	-0.005
Pose (weighted)	-0.1	-0.05

h) G1 Walk Rough.: `G1WalkRough` is the rough-terrain variant trained with SAC on both backends. The environment shares the same 29-DOF humanoid structure as G1 Walk Flat with terrain-aware reset behaviour.

Observation space. Same structure as G1 Walk Flat: actor 98-dimensional, critic 101-dimensional. Refer to the G1 Walk Flat paragraph for the layout.

Action space. 29-dimensional joint-position offset with action scale $s = 1.0$ (raised from the task default 0.25). PD gains $K_p = 50.0$, $K_d = 1.0$.

Commands and termination. The gait phase is initialized at the configured offset and the reset base-velocity is limited to 0.5 m/s. A base-velocity curriculum is enabled with initial scale 0.5, maximum scale 1.0, level-down threshold 150, level-up threshold 750, and degree 0.001. Under MotrixSim, Kp/Kd randomization is disabled and the simulation step is set to 0.01 s. Termination uses a maximum tilt of $65°$ and a minimum base height of 0.3 m.

Domain randomization. Under MuJoCoUni, observation noise uses level 1.0 with joint-angle scale 0.01 and joint-velocity scale 0.1 (other channels zero). Under MotrixSim, the default noise level is used. MuJoCoUni enables policy symmetry at the algorithm level; MotrixSim does not. Mass, COM, and push randomization are not enabled on either backend.

Reward design. Table XVIII lists the active reward scales. MotrixSim uses a tighter feet-phase tracking sigma than MuJoCoUni.

TABLE XVIII: Reward terms for `g1_walk_rough`.

Term	MuJoCoUni	MotrixSim
Linear velocity tracking	2.0	2.2
Yaw angular velocity tracking	1.5	1.8
Penalty ang vel xy	-1.0	-1.2
Penalty orientation	-10.0	-12.0
Penalty action rate	-4.0	-2.5
Pose (weighted)	-0.5	-0.6
Penalty feet orientation	-20.0	-5.0
Feet phase	5.0	6.0
Alive	10.0	12.0

Shaping parameters used by the table above: velocity-tracking $\sigma = 0.25$, gait frequency 1.5 Hz, feet-phase swing height 0.09 m, feet-phase tracking $\sigma = 0.04$ (MuJoCoUni) / 0.008 (MotrixSim), close-feet threshold 0.15 m, base-height target 0.754 m. Termination thresholds (maximum tilt $65°$, minimum base height 0.3 m) are repeated here from the Commands and termination paragraph for reference.

2) Motion Tracking: The motion-tracking family imitates a reference motion clip on the 29-DOF G1 humanoid. All five tasks (`g1_motion_tracking`, `g1_climb_tracking`, `g1_flip_tracking`, `g1_wall_flip_tracking`, `g1_box_tracking`) share the same observation/action layout and reward-term library; they differ in reference motion clip, scene, sampling mode, and termination thresholds. The shared structure is described once under G1 Motion Tracking; per-variant deltas follow.

a) G1 Motion Tracking.: `G1MotionTracking` runs on the flat G1 scene with a dance reference clip. The configuration is identical across the two backends. Simulation step from the G1 base config, control step $\Delta t_{\text{ctrl}} = 0.02$ s, maximum episode 10 s.

Observation space. The actor observation is 176-dimensional:

$$o_t^{actor} = [m_t^{joint},\, p_{b,t}^{ref},\, R_{b,t}^{ref},\, v_t^{base},\, \omega_t,\, q_t - q_{\text{default}},\, \dot{q}_t,\, a_{t-1}], \tag{7}$$

where $m_t^{joint} \in \mathbb{R}^{58}$ is the reference joint position and velocity (29+29), $p_{b,t}^{ref} \in \mathbb{R}^3$ is the reference anchor position in body frame, $R_{b,t}^{ref} \in \mathbb{R}^6$ is the reference anchor orientation (6D rotation representation), and the remaining channels mirror the locomotion observation layout for 29 joints. The critic observation appends per-body privileged transforms for all 14 tracked bodies (3D position + 6D orientation each, $14 \times 9 = 126$ extra dims), giving a 302-dimensional critic.

The 14 tracked bodies are: `pelvis`, left/right {`hip_roll_link`, `knee_link`, `ankle_roll_link`}, `torso_link` (anchor body), left/right {`shoulder_roll_link`, `elbow_link`, `wrist_yaw_link`}.

Action space. 29-dimensional joint-position offset with a per-joint 29-element action scale (not a single scalar). PD gains from the G1 base config ($K_p = 50.0$, $K_d = 1.0$).

Commands and termination. There is no joystick command channel; the reference motion plays the role of command. Termination occurs when any of the following holds: anchor-position z-error exceeds $0.25\,\mathrm{m}$, end-effector z-error exceeds $0.25\,\mathrm{m}$, or a non-EE body falls below $0.05\,\mathrm{m}$ when undesired-contact termination is enabled. Anchor-orientation termination is disabled.

Reference-clip sampling. The per-environment start frame is chosen at reset by one of four modes: always frame zero, random clip start, uniform over all frames, or failure-weighted adaptive bin sampling. G1 Motion Tracking uses the adaptive mode.

Domain randomization. Under MuJoCoUni, observation noise uses the environment defaults. Under MotrixSim, noise level 1.0 is enabled with joint-angle scale 0.01, joint-velocity scale 1.5, and gyro scale 0.2. Base-mass, COM, gravity, push, and Kp/Kd randomization are not enabled.

Reward design. Each motion-tracking term has the form $\exp(-e^2/\sigma^2)$ where e is the reference-tracking error in the corresponding channel; non-tracking penalties use squared / L2 forms identical to the locomotion library. Table XIX lists the active scales.

TABLE XIX: Reward terms for `g1_motion_tracking`.

Term	MuJoCoUni	MotrixSim
motion_global_root_pos	0.5	1.0
motion_global_root_ori	0.5	0.5
motion_body_pos	1.0	1.0
motion_body_ori	1.0	1.0
motion_body_lin_vel	1.0	1.0
motion_body_ang_vel	1.0	1.0
action_rate_l2	-0.1	-0.05
joint_limit	-10.0	-10.0
undesired_contacts	/	-0.1

Shaping parameters (per-channel tracking σ used inside the $\exp(-e^2/\sigma^2)$ form) are identical on the two backends: $\sigma_{\text{root_pos}} = 0.3$, $\sigma_{\text{root_ori}} = 0.4$, $\sigma_{\text{body_pos}} = 0.3$, $\sigma_{\text{body_ori}} = 0.4$, $\sigma_{\text{body_lin_vel}} = 1.0$, $\sigma_{\text{body_ang_vel}} = 3.14$, $\sigma_{\text{joint_pos}} = 0.2$, $\sigma_{\text{joint_vel}} = 1.0$. The joint-position and joint-velocity tracking terms are not used in this configuration.

b) G1 Climb Tracking.: The climb variant uses a scene with a 20-rung wall and a matched climbing reference clip. Maximum episode is $15\,\mathrm{s}$, sampling is adaptive, simulation step is $0.005\,\mathrm{s}$, undesired-contact termination is enabled, and anchor/end-effector z-error thresholds are both $0.3\,\mathrm{m}$. The episode is not truncated when the clip ends.

The per-joint action scale is roughly 0.55 for hip and ankle joints, 0.35 for the knee, 0.44 for waist and shoulder joints, and 0.07 for wrist joints. Observation, action, and termination structure match G1 Motion Tracking.

Reward design. Reward scales are identical across the two backends (Table XX). Compared to G1 Motion Tracking, the climb variant additionally weights end-effector vertical tracking, joint-position tracking, and joint-velocity tracking to encourage limb coordination with the reference clip.

TABLE XX: Reward terms for `g1_climb_tracking` (mj/mx identical).

Term	Weight
motion_global_root_pos	0.5
motion_global_root_ori	0.5
motion_body_pos	2.0
motion_body_ori	1.5
motion_body_lin_vel	1.0
motion_body_ang_vel	1.0
motion_ee_body_pos_z	2.0
motion_joint_pos	0.5
motion_joint_vel	0.25
action_rate_l2	-0.005
joint_limit	-10.0
undesired_contacts	-0.1

The sigma values match the G1 Motion Tracking defaults (Table XIX).

c) G1 Flip Tracking.: The flip variant uses the flat G1 scene with a 360-degree flip reference clip. Sampling always starts from frame zero, the episode is not truncated when the clip ends, and the simulation step is $0.005\,\mathrm{s}$.

Under MuJoCoUni, the anchor and end-effector z-error thresholds are both $0.5\,\mathrm{m}$, undesired-contact termination is enabled, and the per-joint action scale matches G1 Climb Tracking. Under MotrixSim, the corresponding thresholds and action scale use the baseline environment settings.

TABLE XXI: Reward terms for `g1_flip_tracking`.

Term	MuJoCoUni	MotrixSim
motion_global_root_pos	0.5	1.0
motion_global_root_ori	0.5	0.5
motion_body_pos	2.0	1.0
motion_body_ori	1.5	1.0
motion_body_lin_vel	1.0	1.0
motion_body_ang_vel	1.0	1.0
motion_ee_body_pos_z	2.0	/
action_rate_l2	-0.005	-0.05
joint_limit	-10.0	-10.0
undesired_contacts	-0.1	/

d) G1 Wall Flip Tracking.: The wall-flip variant uses a flat scene with a wall and a wall-flip reference clip. Sampling always starts from frame zero, the episode is not truncated when the clip ends, the simulation step is $0.005\,\mathrm{s}$, anchor and end-effector z-error thresholds are both $0.5\,\mathrm{m}$, undesired-contact termination is enabled, and the per-joint action scale matches G1 Climb Tracking. Under MotrixSim, the solver runs three substep iterations per control step.

Reward scales are identical across the two backends and match the climb-tracking weights (Table XX); only the reference clip and the wall-scene termination geometry change.

e) G1 Box Tracking.: The box variant uses a flat scene with a large box and a box-manipulation reference clip. It extends G1 Motion Tracking with explicit object-state tracking: the critic observation appends 12 extra dimensions (object position, object 6D orientation, object linear velocity, all in body frame) for a 314-dimensional critic. The actor observation remains 176-dimensional. Under MuJoCoUni, the simulation step is $0.005\,\mathrm{s}$; under MotrixSim, the base step is used.

Under MotrixSim, empirical normalization is enabled, an asymmetric critic observation group is declared, and observation noise uses level 1.0 with joint-angle scale 0.01, joint-velocity scale 1.5, and gyro scale 0.2.

TABLE XXII: Reward terms for `g1_box_tracking`.

Term	MuJoCoUni	MotrixSim
motion_global_root_pos	0.5	1.0
motion_global_root_ori	0.5	0.5
motion_body_pos	1.0	1.0
motion_body_ori	1.0	1.5
motion_body_lin_vel	1.0	1.0
motion_body_ang_vel	1.0	1.5
action_rate_l2	-0.1	-0.1
joint_limit	-10.0	-10.0
undesired_contacts	-0.1	-0.1
object_global_ref_position_error_exp	2.0	4.0
object_global_ref_orientation_error_exp	2.0	3.0

Body-tracking sigmas inherit the G1 Motion Tracking defaults; the object-tracking sigmas used inside $\exp(-e^2/\sigma^2)$ differ between backends: $\sigma_{\text{object_pos}} = 0.2$ (MuJoCoUni) / 0.12 (MotrixSim), $\sigma_{\text{object_ori}} = 0.3$ (MuJoCoUni) / 0.2 (MotrixSim).

3) Manipulation-Locomotion: This subsection covers tasks where locomotion is coupled with an upper-body manipulation or posture objective. Two tasks are included: Go2 Hand Stand (rear-leg balance) and Go2 Arm Manip Loco (locomotion with a 6-DOF arm tracking an end-effector goal).

a) Go2 Hand Stand.: `Go2HandStand` rewards the robot for inverting onto its front legs while maintaining a target torso height. Under MuJoCoUni the simulation step is $0.005\,\mathrm{s}$; under MotrixSim it is $0.01\,\mathrm{s}$. Control step $\Delta t_{\text{ctrl}} = 0.02\,\mathrm{s}$, maximum episode $20\,\mathrm{s}$.

Observation space. The actor observation is 42-dimensional:

$$o_t = [\omega_t,\, -g_t,\, q_t - q_{\text{default}},\, \dot{q}_t,\, a_{t-1}], \tag{8}$$

where joint offset, joint velocity, and last action are 12-dimensional each. No velocity command channel; the task is a fixed posture-tracking objective. The critic observation is 46-dimensional (adds base linear velocity 3 and torso height 1).

Action space. 12-dimensional joint-position offset with default `action_scale`. PD gains $K_p = 35.0$, $K_d = 0.5$.
Commands and termination. The target pose is inverted handstand: target torso height $z_{des} = 0.55\,\text{m}$, desired gravity $g_{des} = (-1, 0, 0)$ (body x-axis aligned with world gravity). Termination occurs when the up-vector z-component satisfies $g_t^z \leq -0.25$ (robot fully inverted past the target) or when undesired contacts on front legs/hips/thighs occur. A rear-leg gait at 2 Hz (RL phase 0, RR phase 0.5) provides a phase signal for the gait-aware reward terms.
Domain randomization. Under MotrixSim, Kp/Kd randomization is disabled. No additional domain randomization is enabled on top of the environment defaults.
Reward design. Table XXIII lists the active reward scales. The two backends differ only in the front-leg contact weight.

TABLE XXIII: Reward terms for `go2_handstand`.

Term	MuJoCoUni	MotrixSim
Contact (front-leg)	-1.0	-0.5
Height	1.0	1.0
Orientation (alignment to target gravity)	1.0	1.0
Pose (deviation from default)	-0.3	-0.3
Penalty contact (penalty bodies)	-0.2	-0.2
Action rate	-0.01	-0.01
Target pose (`tar`)	0.3	0.3
Feet air time (rear legs)	1.0	1.0
World z-velocity penalty	-1.0	-1.0

Shaping parameters: velocity-tracking $\sigma = 0.25$ and base-height target $0.3\,\text{m}$ on both backends. The `height` term uses $\exp(-|z_{des} - h|/0.1)$, the `orientation` term uses $[0.5\cos\angle(g, g_{des}) + 0.5]^2$, and the `tar` reward is gated by $h \geq 0.8\, z_{des}$.

b) Go2 Arm Manip Loco.: `Go2ArmManipLoco` is available only on the MuJoCoUni backend. Simulation step $\Delta t_{\text{sim}} = 0.01\,\text{s}$, control step $\Delta t_{\text{ctrl}} = 0.02\,\text{s}$, maximum episode $20\,\text{s}$.
Observation space. The per-step observation is 79-dimensional:

$$o_t = [v_t^{base},\, \omega_t,\, -g_t,\, c_t,\, \phi_t,\, q_t - q_{\text{default}},\, \dot{q}_t,\, p_t^{ee},\, p_t^{goal},\, e_t^{ee},\, a_{t-1}], \tag{9}$$

where $\phi_t \in \mathbb{R}^4$ is the four-foot gait phase, joint offset/velocity are 18-dimensional each (12 leg + 6 arm), p_t^{ee}, p_t^{goal}, $e_t^{ee} \in \mathbb{R}^3$ are end-effector position, goal, and error in body frame, and the action history is 18-dimensional. The actor observation drops base linear velocity (to avoid bypassing the on-board estimator) and uses a per-step history of H_a frames; the critic observation keeps linear velocity and uses H_c frames. Default history length is $H_a = H_c = 1$.
Action space. 18-dimensional: 12 leg-joint offsets with action scale 0.25 (hip-joint scale 0.125) and 6 arm-joint offsets with arm action scale zero. Leg PD gains $K_p = 35.0$, $K_d = 0.5$. With the arm scale set to zero, the policy controls only the legs; the arm follows the IK-derived target from the end-effector goal.
End-effector goal sampling. The end-effector goal is sampled in spherical coordinates around the body: radius $\in [0.3, 0.6]\,\text{m}$, azimuth $\in [-1.26, 1.05]\,\text{rad}$, polar angle $\in [-2.36, 2.36]\,\text{rad}$. Trajectory time $\in [1.0, 3.0]\,\text{s}$, hold time $\in [0.5, 2.0]\,\text{s}$. Collision bounds: upper $[0.3, 0.15, -0.115]$, lower $[-0.2, -0.15, -0.515]$, underground limit $z = -0.57$. The IK uses damping 0.05, gain 1.0, Δq-clip 0.2, with target-orientation tracking.
Commands and termination. Velocity-command range $[(-0.6, -0.4, -0.8), (1.0, 0.4, 0.8)]$, zero-command probability 0.15 (for stable standing), command resampling every $4.0\,\text{s}$. The velocity curriculum is disabled. Termination: $g_t^z \leq 0.5$.
Domain randomization. The following terms are enabled: body-mass multiplier $[0.9, 1.1]$, random COM offset $x \in [-0.03, 0.03]$, ground-friction multiplier $[0.8, 1.2]$, DOF-armature multiplier $[0.8, 1.2]$, base pushes every 500 steps with maximum force $[1.2, 1.2, 0.6]$, and Kp/Kd multipliers $[0.9, 1.1]$. Base-mass and gravity randomization are not enabled.
Reward design. Table XXIV lists the active reward scales.

Shaping parameters: velocity-tracking $\sigma = 0.25$, base-height target $0.3\,\text{m}$, object-distance kernel $\sigma = 0.1$.

4) Dexterous-Hand and In-Hand Manipulation: This subsection covers in-hand manipulation tasks where a multi-finger hand rotates a free object about a specified axis. Two tasks are included: Allegro Inhand Rotation (16-DOF hand, ball) and Sharpa Inhand Rotation (22-DOF hand with tactile sensing).

a) Allegro Inhand Rotation.: `AllegroInhandRotation` rotates a free ball about a fixed world-axis using the 16-DOF Allegro hand. The configuration is identical across the two backends. Simulation step $\Delta t_{\text{sim}} = 0.005\,\text{s}$, control step $\Delta t_{\text{ctrl}} = 0.05\,\text{s}$ (10 simulator steps per control step), maximum episode $20\,\text{s}$.
Observation space. The observation is 105-dimensional, organized as a lag-history of 3 steps of a 35-dimensional per-step frame:

$$f_t = [\tilde{q}_t^{hand},\, q_t^{target},\, p_t^{ball}], \tag{10}$$

where $\tilde{q}_t^{hand} \in \mathbb{R}^{16}$ is the normalized hand joint position (mapped from joint limits to $[-2, 2]$), $q_t^{target} \in \mathbb{R}^{16}$ is the current incremental joint target, and $p_t^{ball} \in \mathbb{R}^3$ is the ball position. Actor and critic share a single observation group; the critic has no privileged channel.

TABLE XXIV: Reward terms for `go2_arm_manip_loco`.

Term	MuJoCoUni
Linear velocity tracking	2.0
Yaw angular velocity tracking	0.5
Vertical linear velocity	-5.0
Roll/pitch angular velocity	-0.1
Roll	-5.0
Base height	-100.0
Leg pose	-0.1
Action rate	-0.005
Stand still	-0.5
Contact phase agreement	0.24
Swing-foot height	4.0
Foot drag	-0.1
Object distance (to EE goal)	2.0
Object distance L2	-0.5
Arm collision	-1.0

Action space. 16-dimensional in $[-1, 1]$. The environment maps actions to actuator targets incrementally: $q_t^{target} = q_{t-1}^{target} + s\,\mathrm{clip}(a_t)$ with $s = 1/24 \approx 0.0417$, then clipped to the actuator-range limits. PD gains $K_p = 1.0$, $K_d = 0.1$. Torque is clipped to $[-0.5, 0.5]$.

Commands and termination. The rotation axis is the world z-axis, $\hat{n} = (0, 0, 1)$. The episode terminates when the ball height drops below $0.125\,\mathrm{m}$.

Domain randomization. All online domain randomization is disabled (base-mass, COM, gravity, push, joint noise, ball-velocity noise, ball-z offset). Reset-time grasp variation is the only source of initialization diversity.

Reward design. Reward scales are identical across the two backends (Table XXV).

TABLE XXV: Reward terms for `allegro_inhand` (mj/mx identical).

Term	Weight
rotate ($\mathtt{clip}(\boldsymbol{\omega}^{ball} \cdot \hat{n}, -0.5, 0.5)$)	1.25
obj_linvel ($\sum_i \lvert v_i^{ball} \rvert$)	-0.3
pose_diff ($\sum_j (q_j - q_j^{init})^2$)	-0.3
torque ($\sum_j \tau_j^2$)	-0.1
work ($(\sum_j \tau_j \dot{q}_j)^2$)	-2.0

Shaping parameters: angular-velocity clip range $[-0.5, 0.5]$ rad/s inside the rotate term, and ball-height reset threshold $0.125\,\mathrm{m}$ used by the termination check. The reward is $\Delta t_{\mathrm{ctrl}} \sum_i w_i r_i$.

b) Sharpa Inhand Rotation.: `SharpaInhandRotation` rotates a cylinder using the 22-DOF Sharpa hand with tactile sensing. The numbers reported here correspond to the MuJoCoUni HORA teacher configuration, which is shared by the APPO, SAC, and PPO HORA comparisons. Simulation step $1/240\,\mathrm{s}$, control step $\Delta t_{\mathrm{ctrl}} = 12/240 = 0.05\,\mathrm{s}$, maximum episode $20\,\mathrm{s}$.

Observation space. The per-step policy frame is 49-dimensional:

$$f_t = [\tilde{q}_t^{hand},\, q_t^{target},\, F_t^{tactile}], \tag{11}$$

with $\tilde{q}_t^{hand} \in \mathbb{R}^{22}$, $q_t^{target} \in \mathbb{R}^{22}$, and tactile forces $F_t^{tactile} \in \mathbb{R}^5$ (one per fingertip). The frame is stacked over a 3-step lag history, giving a 147-dimensional policy observation. The privileged tail is 9-dimensional: object position delta (3), friction scale (1), mass (1), COM offset (3), and object scale (1). In flattened mode the single observation group is $147+9 = 156$-dimensional. In separated mode the actor receives the 147-dimensional policy observation, while the critic receives the 156-dimensional concatenation.

Action space. 22-dimensional in $[-1, 1]$. Incremental position control: $q_t^{target} = q_{t-1}^{target} + s\,\mathrm{clip}(a_t)$, $s = 1/24$, with joint limits scaled by 0.9. PD gains are set per actuator and randomized at reset within $[0.5, 2.0]$ around their nominal values.

Commands and termination. Rotation axis $\hat{n} = (0, 0, 1)$. Termination uses object world-z height bounds $[p_z^{obj} - 0.5\Delta h, p_z^{obj} + 0.5\Delta h]$ with $\Delta h = 0.04\,\mathrm{m}$ centered on the reset object position; the fallback bounds are $[0.59906, 0.63906]\,\mathrm{m}$. The rotation rollout does not use angular-violation termination; angular deviation is used only during offline grasp-state filtering.

Domain randomization. A rich randomization stack is enabled on both backends (Table XXVI). The reported Sharpa teacher runs use eight cylinder scales $\{0.8 \ldots 1.5\}$, excluding 1.6 to align with the external baseline setup. The active gravity setting is fixed-magnitude direction randomization; full-vector gravity randomization is disabled.

Reward design. Reward scales are identical across the two backends (Table XXVII).

The rotate term clips its angular-velocity argument to $[-0.5, 0.5]$ rad/s before applying the weight.

TABLE XXVI: Domain randomization for `sharpa_inhand`.

Item	MuJoCoUni	MotrixSim
Scale list (cylinder sizes)	$\{0.8 \ldots 1.5\}$	$\{0.8 \ldots 1.5\}$
`randomize_gravity_direction`	true	true
Gravity magnitude	9.81	9.81
`randomize_gravity` (full vector)	false	false
`randomize_pd_gains`	true	true
K_p scale range	$[0.5, 2.0]$	$[0.5, 2.0]$
K_d scale range	$[0.5, 2.0]$	$[0.5, 2.0]$
`randomize_friction`	true	true
Friction scale range	$[0.75, 1.25]$	$[0.75, 1.25]$
Elastomer/metal/object base friction	$2.0, 1.0, 2.0$	$2.0, 1.0, 2.0$
`randomize_com`	true	true
COM offset range	$[-0.01, 0.01]$	$[-0.01, 0.01]$
`randomize_mass`	true	true
Mass range	$[0.01, 0.25]$	$[0.01, 0.25]$
Force perturbation scale	2.0	2.0
Force probability per step	0.25	0.25
Force decay / interval	$0.9/0.08$	$0.9/0.08$
Joint observation noise scale	0.02	0.02
Contact latency	0.005	0.005
Contact sensor noise	0.01	0.01

TABLE XXVII: Reward terms for `sharpa_inhand` (mj/mx identical).

Term	Weight
rotate ($\texttt{clip}(\boldsymbol{\omega}^{obj} \cdot \hat{n}, -0.5, 0.5)$)	2.5
obj_linvel ($\sum_i \lvert v_i^{obj} \rvert$)	-0.3
pose_diff ($\sum_j (q_j - q_j^{def})^2$)	-0.4
torque (squared virtual torque)	-0.1
work ($(\sum_j \tau_j \dot{q}_j)^2$)	-0.5
object_pos ($1/(\lVert p^{obj} - p^{anchor} \rVert + 10^{-3})$)	0.003

C.4 Algorithm Hyperparameters

This subsection lists the per-algorithm global defaults and the per-task overrides applied on top of them. Reward weights and environment-side hyperparameters are documented in Section C.3; only algorithm-side training hyperparameters appear here.

1) PPO: We report PPO hyperparameters as global defaults followed by per-task overrides.

a) PPO global defaults.: Table XXVIII lists the global defaults inherited by every PPO task before any per-task override.

b) PPO overrides across tasks.: Tables XXIX–XXXI summarize the PPO-side overrides for each task. Fields not listed are inherited from Table XXVIII. When the two backends differ, the value is written as "mj / mx"; identical values are written once.

For Go1 Joystick Flat, empirical normalization, the lower action-noise std, and the lower learning rate / entropy coefficient are applied only under MotrixSim. Go2 Joystick Flat uses the same lowered values on both backends. Go1 / Go2 Joystick Rough share a single hyperparameter set between the two backends with action-noise std 1.0 and entropy coefficient 0.01. Go2W Joystick Flat and Rough use the same hyperparameters on both backends apart from rollout length and iteration count in the Rough case.

Asymmetric observation groups means the actor and critic see different channels, enabling privileged critic observations. G1 Walk Flat enables this only under MotrixSim. G1 Box Tracking uses `num_envs`$= 1024$, `max_iterations`$= 30000/40000$, empirical normalization false / true, entropy coefficient $0.005/0.002$, save interval 500, and asymmetric observation groups under MotrixSim.

Go2 Arm Manip Loco is available only under MuJoCoUni; the MotrixSim column does not apply. Allegro Inhand and generic Sharpa Inhand use identical hyperparameters on both backends. The Sharpa column reports the HORA PPO setting used for the main Sharpa PPO comparison.

2) APPO: APPO is the asynchronous on-policy variant used in UniLab; it shares PPO's clipped-surrogate objective but allows the learner to consume rollouts produced with a slightly stale policy. Only APPO training hyperparameters appear here; task-side values (rewards, observation/action spaces, domain randomization) are documented in the task specifications above.

a) APPO global defaults.: Table XXXII lists the global defaults inherited by every APPO task before any per-task override.

b) APPO overrides across tasks.: Tables XXXIII and XXXIV list the per-task overrides. Fields not listed are inherited from Table XXXII. When the two backends differ, the value is written as "mj / mx"; identical values appear once.

TABLE XXVIII: PPO global default hyperparameters.

Field	Default Value
Runner / Environment	
algo	ppo
algo_log_name	rsl_rl_ppo
seed	1
num_envs	4096
num_steps_per_env	24
max_iterations	101
save_interval	100
empirical_normalization	false
runner_class_name	OnPolicyRunner
obs_groups.default	[policy]
Policy network	
policy.class_name	ActorCritic
policy.actor_hidden_dims	$[512, 256, 128]$
policy.critic_hidden_dims	$[512, 256, 128]$
policy.activation	elu
policy.init_noise_std	1.0
Algorithm	
algorithm.class_name	FinalObservationAwarePPO
algorithm.value_loss_coef	1.0
algorithm.use_clipped_value_loss	true
algorithm.clip_param	0.2
algorithm.entropy_coef	0.01
algorithm.num_learning_epochs	5
algorithm.num_mini_batches	4
algorithm.learning_rate	1.0×10^{-3}
algorithm.schedule	adaptive
algorithm.gamma	0.99
algorithm.lam	0.95
algorithm.desired_kl	0.01
algorithm.max_grad_norm	1.0
algorithm.adaptive_kl_beta	0.9
algorithm.adaptive_lr_growth	1.1
algorithm.adaptive_lr_decay	1.2
algorithm.adaptive_lr_update_interval	5

TABLE XXIX: PPO overrides for locomotion tasks (Go1, Go2, Go2W families). Cells with a slash report "mj / mx" values; single values are shared between the two backends.

Field	Go1 Flat	Go1 Rough	Go2 Flat	Go2 Rough	Go2W Flat	Go2W Rough
num_envs	4096	4096	1024	4096	1024	2048
num_steps_per_env	24	24	24	24	24	48/24
max_iterations	151	600	151	1000	151	5000/2000
empirical_normalization	false / true	false	true	false	true	false
init_noise_std	inherited / 0.5	1.0	0.5	1.0	0.5	inherited
learning_rate	$10^{-3}/3\times10^{-4}$	10^{-3}	3×10^{-4}	10^{-3}	3×10^{-4}	inherited
entropy_coef	$0.01/10^{-3}$	0.01	10^{-3}	0.01	10^{-3}	inherited

Go1/Go2 Joystick Flat run under MuJoCoUni only. The Sharpa HORA setting uses `runtime_impl=hora_appo`, separated actor/critic observations, a 9-dimensional privileged embedding, and HORA actor/critic model classes. The generic non-HORA Sharpa APPO setting is not the teacher recipe reported for the Sharpa HORA curves.

All four motion-tracking tasks use the same APPO configuration on both backends. G1 Box Tracking is not available under APPO.

3) SAC: SAC is the entropy-regularized off-policy actor-critic used in UniLab for replay-buffer experiments. The shared family contains two implementation variants: the standard SAC trainer (`algo: sac`) and the FlashSAC accelerated variant (`algo: flashsac`). Both consume rollouts via a replay buffer and respect the off-policy producer/consumer protocol described in the main text.

a) SAC global defaults.: Table XXXV lists the global defaults inherited by every SAC task before any per-task override.

TABLE XXX: PPO overrides for humanoid locomotion and tracking tasks. Cells with a slash report "mj / mx" values; single values are shared between the two backends.

Field	G1 Walk Flat	G1 Motion	G1 Climb	G1 Flip	G1 Wall Flip
num_envs	2048	1024	1024	1024	1024
num_steps_per_env	24	24	24	24	24
max_iterations	2200	15000	20000	20000/30000	20000/12000
empirical_normalization	false / true	false	true	true / false	true
init_noise_std	inherited / 0.5	inherited	inherited	inherited	inherited
learning_rate	$10^{-3}/3\times10^{-4}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}
entropy_coef	$0.01/5\times10^{-3}$	0.005	0.005	0.005	0.005
desired_kl	inherited	inherited	0.01	0.01/ inh.	0.01
save_interval	100	500	500	500	500
obs_groups	symmetric / asymmetric	symmetric	asymmetric	asymmetric	asymmetric

TABLE XXXI: PPO overrides for handstand, arm-loco, and dexterous tasks.

Field	Go2 HandStand	Go2 Arm Loco	Allegro	Sharpa HORA PPO
num_envs	1024	4096	16384	2048
num_steps_per_env	24	24	8	24
max_iterations	3000	151	201	301
empirical_normalization	inherited	true	true	true
init_noise_std	0.5	0.5	inherited	inherited
learning_rate	inherited	3×10^{-4}	inherited	10^{-3}
entropy_coef	0.005	10^{-3}	0.01	0.01
value_loss_coef	inherited	inherited	4.0	4.0
desired_kl	inherited	inherited	0.02	0.02
save_interval	inherited	inherited	inherited	50
obs_groups	asymmetric	symmetric	symmetric	actor/critic both [actor]

b) FlashSAC global defaults.: Table XXXVI gives the resolved FlashSAC defaults. The two recipes differ in actor/critic capacity, replay-buffer warmup length, and the addition of FlashSAC-specific reward / temperature parameters.

c) SAC and FlashSAC overrides across tasks.: Tables XXXVII and XXXIX summarize the per-task overrides. Fields not listed are inherited from the corresponding default tables. When the two backends differ, the value is written as "mj / mx"; identical values appear once.

G1 Motion Tracking under SAC shares the same environment as the PPO/APPO version; under MotrixSim only the backend identity and the Kp/Kd randomization switches differ from MuJoCoUni.

The Sharpa HORA SAC setting keeps the standard SAC replay objective and uses a HORA-style SAC actor that consumes the 9-dimensional privileged tail described in Section C.34. Task-side reward, observation, action, and domain-randomization values are therefore documented once in the Sharpa task subsection rather than repeated here.

Both FlashSAC tasks are available only under MuJoCoUni.

TABLE XXXII: APPO global default hyperparameters.

Field	Default Value
Runner / Environment	
algo	appo
algo_log_name	appo
seed	1
num_envs	2048
steps_per_env	24
max_iterations	150
save_interval	50
obs_groups.actor	{policy: 0}
Actor / Critic networks	
actor.class_name	rsl_rl.models.MLPModel
actor.hidden_dims	$[512, 256, 128]$
actor.activation	elu
actor.obs_normalization	false
actor.distribution_cfg.class_name	GaussianDistribution
actor.distribution_cfg.init_std	1.0
actor.distribution_cfg.std_type	scalar
critic.class_name	rsl_rl.models.MLPModel
critic.hidden_dims	$[512, 256, 128]$
critic.activation	elu
critic.obs_normalization	false
Algorithm	
algorithm.num_learning_epochs	5
algorithm.num_mini_batches	4
algorithm.clip_param	0.2
algorithm.gamma	0.99
algorithm.lam	0.95
algorithm.value_loss_coef	1.0
algorithm.entropy_coef	0.01
algorithm.learning_rate	1.0×10^{-3}
algorithm.max_grad_norm	1.0
algorithm.use_clipped_value_loss	true
algorithm.schedule	adaptive
algorithm.desired_kl	0.01
algorithm.adaptive_kl_factor	1.2
algorithm.adaptive_lr_factor	1.1
algorithm.optimizer	adam
algorithm.tau (target update)	1.0
algorithm.target_update_freq	1
algorithm.vtrace_clip_rho	1.0
algorithm.vtrace_clip_c	1.0

TABLE XXXIII: APPO overrides for locomotion and dexterous tasks.

Field	Go1 Flat	Go2 Flat	Allegro MuJoCo	Sharpa HORA
num_envs	inherited	inherited	1024	2048
steps_per_env	inherited	inherited	8	8
max_iterations	150	150	3000	1360
save_interval	inherited	inherited	5000	170
training.replay_queue_size	inherited	inherited	4	8
actor.obs_normalization	inherited	inherited	true	true
critic.obs_normalization	inherited	inherited	true	true
algorithm.value_loss_coef	inherited	inherited	4.0	4.0
algorithm.desired_kl	inherited	inherited	0.025	0.04

TABLE XXXIV: APPO overrides for motion-tracking tasks (mj/mx identical for all four).

Field	G1 Motion	G1 Climb	G1 Flip	G1 Wall Flip
num_envs	1024	1024	1024	1024
steps_per_env	inherited	inherited	inherited	inherited
max_iterations	5000	20000	5000	5000
save_interval	500	500	500	500

TABLE XXXV: SAC global default hyperparameters.

Field	Default Value
Runner / Replay	
algo	sac
algo_log_name	fast_sac
seed	1
num_envs	4096
batch_size	8192
replay_buffer_n	512
updates_per_step	4
learning_starts	1
policy_frequency	4
env_steps_per_sync	1
max_iterations	500
save_interval	500
Network	
actor_hidden_dim	512
critic_hidden_dim	768
num_atoms	101
obs_normalization	true
use_layer_norm	true
use_symmetry	false
Algorithm	
gamma	0.97
tau	0.125
actor_lr	3.0×10^{-4}
critic_lr	3.0×10^{-4}
algo_params.alpha_lr	3.0×10^{-4}
algo_params.alpha_init	0.01
algo_params.target_entropy_ratio	0.0
algo_params.max_grad_norm	0.0
algo_params.amp_dtype	auto
algo_params.use_compile	true

TABLE XXXVI: FlashSAC global default hyperparameters.

Field	Default Value
Runner / Replay	
algo	flashsac
algo_log_name	flash_sac
num_envs	1024
batch_size	2048
replay_buffer_n	512
updates_per_step	2
learning_starts	98
policy_frequency	2
max_iterations	5000
save_interval	1000
Network	
actor_hidden_dim	128
critic_hidden_dim	256
num_atoms	101
obs_normalization	false
use_layer_norm	false
algo_params.actor_num_blocks	2
algo_params.critic_num_blocks	2
Algorithm	
gamma	0.97
tau	0.01
actor_lr	3.0×10^{-4}
critic_lr	3.0×10^{-4}
algo_params.normalize_reward	true
algo_params.normalized_g_max	5.0
algo_params.actor_bc_alpha	0.0
algo_params.actor_noise_zeta_mu	2.0
algo_params.actor_noise_zeta_max	16
algo_params.critic_min_v / critic_max_v	$-5.0/5.0$
algo_params.temp_initial_value	0.01
algo_params.temp_target_sigma	0.15
algo_params.temp_target_entropy	null
algo_params.learning_rate_init / peak / end	$3\times10^{-4}/3\times10^{-4}/1.5\times10^{-4}$
algo_params.learning_rate_warmup_steps	0
algo_params.learning_rate_decay_steps	500,000
algo_params.n_step	1

TABLE XXXVII: SAC overrides for G1 walk and motion-tracking tasks.

Field	G1 Walk Flat	G1 Walk Rough	G1 Motion Tracking
num_envs	2048	2048	2048
learning_starts	10/1	10/1	inherited
max_iterations	5000	5000	25000
save_interval	1000	1000	1000
updates_per_step	8	8	4
policy_frequency	inherited	inherited	2
use_symmetry	true / false	true / false	false
gamma	inherited	inherited	0.99
tau	inherited	inherited	0.05
num_atoms	inherited	inherited	501
algo_params.alpha_init	0.001	0.001	0.1
algo_params.target_entropy_ratio	0.0	0.0	0.5
algo_params.max_grad_norm	inherited	inherited	10.0

TABLE XXXVIII: HORA SAC overrides for Sharpa Inhand Rotation.

Field	Sharpa HORA SAC
`runtime_impl`	`hora_sac`
`num_envs`	1024
`batch_size`	2048
`replay_buffer_n`	1280
`training.env_steps_per_sync`	2
`updates_per_step`	14
`policy_frequency`	2
`learning_starts`	1
`max_iterations`	39063
`save_interval`	1000
`actor_lr`, `critic_lr`, `alpha_lr`	4.5×10^{-4}
`training.use_amp`	true
`algo_params.use_compile`	true
`actor.priv_info_embed_dim`	9
`actor.priv_mlp_hidden_dims`	$[256, 128, 9]$

TABLE XXXIX: FlashSAC overrides for G1 Walk Flat and Go2 Joystick Flat.

Field	G1 Walk Flat	Go2 Joystick Flat
`num_envs`	4096	1024
`batch_size`	inherited	inherited
`replay_buffer_n`	256	4096
`learning_starts`	49	50
`updates_per_step`	8	2
`max_iterations`	5000	4000
`save_interval`	1000	1000
`tau`	0.05	0.05