

MimicLite: Efficient and Effective General Humanoid Motion Tracking

Robotparty Lab

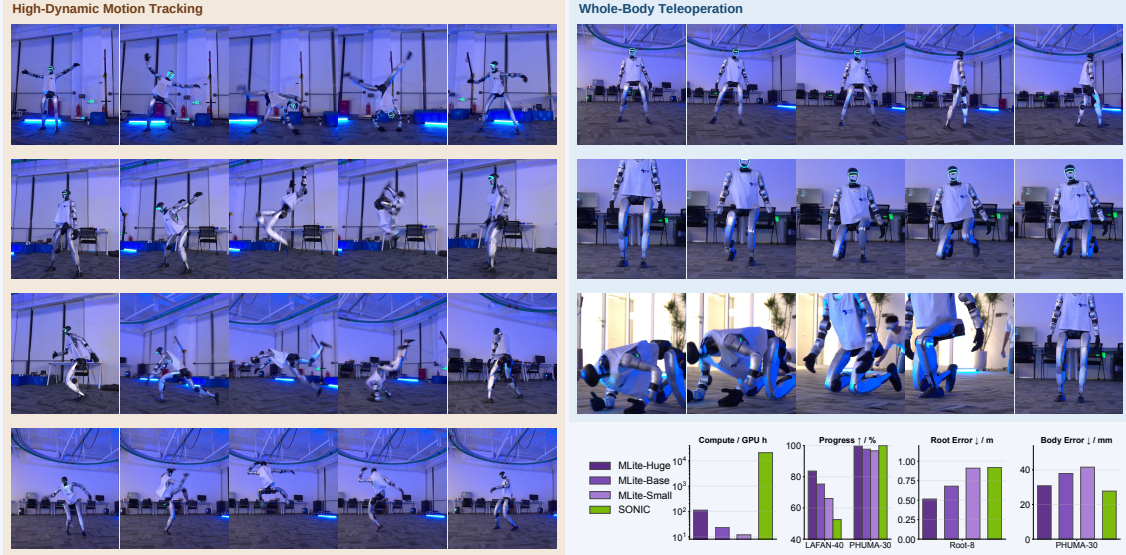


Figure 1: **Left:** a physical Unitree G1 tracks high-dynamic motions, including a cartwheel, an aerial backflip, a tiger-jump-to-shoulder-roll, and a spinning kick. **Upper right:** whole-body teleoperation produces walking, a single-knee-to-double-knee descent, and a hands-down-to-standing recovery. **Lower right:** MimicLite-Huge, MimicLite-Base, and MimicLite-Small use approximately 1/200, 1/900, and 1/2000 of SONIC’s GPU-hours, respectively. Despite this much lower compute cost, MimicLite improves global root tracking over SONIC while achieving comparable local tracking accuracy.

Code: <https://github.com/Roboparty/MimicLite>
Last Updated: July 16, 2026



Abstract: General humanoid motion tracking provides a versatile low-level control prior, but state-of-the-art methods are slow and expensive to train. This cost lengthens iteration cycles, delaying adaptation to new tasks and the training of updated trackers. MimicLite addresses this bottleneck through system-level optimization of the training pipeline, enabling a policy to train in 3 hours on 8 RTX 4090 GPUs while retaining competitive tracking quality. Under a matched MuJoCo evaluation, MimicLite outperforms SONIC in global root tracking and achieves comparable local tracking accuracy. The system enables low-latency, Pico-driven real-time teleoperation of a physical G1 robot, enabling walking, squatting, and single- or double-knee kneeling. The same trained policy can also track highly dynamic motions, such as a tiger jump transitioning into a shoulder roll, spinning kicks, and cartwheels. We report the system design, main results, compute ablations, and remaining limitations.

1 Introduction

Humanoid robots are attractive general-purpose embodiments because their bodies can, in principle, use the same spaces, tools, and demonstrations as people. This morphology also gives humanoids a direct path to learn from human data, including egocentric video, motion-capture datasets, VR teleoperation streams, and human whole-body demonstrations. It also makes control difficult: a humanoid must maintain balance while coordinating many joints, underactuated contacts, and frequent changes in support. A practical low-level controller must therefore combine stability, dynamic motion, and deployable sensing constraints rather than only replay reference poses.

General motion tracking is a scalable abstraction for this problem. Instead of designing a separate reward and controller for each behavior, a tracking policy receives a reference whole-body motion and learns to follow it in simulation under physical constraints. The resulting tracker can serve as a reusable control prior for offline replay, teleoperation, demonstration collection, whole-body manipulation, locomotion, and expressive motion generation [12, 20, 6]. Recent systems such as GMT [2], Any2Track [22], KungfuBot2 [4], SONIC [13], Humanoid-GPT [16], HEFT [11], TeleopIt [1], and TWIST2 [21] show that this abstraction can scale across broad motion distributions, large human-motion corpora, and deployable teleoperation interfaces.

Despite this progress, state-of-the-art general motion trackers remain slow and expensive to train. For example, SONIC [13], a state-of-the-art general motion tracker, reports training on 128 GPUs for seven days, totaling approximately 21,000 GPU-hours. Long training cycles impede rapid iteration over data, observations, rewards, termination rules, action interfaces, and deployment settings, making it costly to adapt a tracker to new tasks.

This report presents MimicLite, a system-level approach to efficient general humanoid motion tracking. MimicLite trains a policy in 3 hours on 8 RTX 4090 GPUs. Under a matched MuJoCo evaluation, the policy improves global root tracking relative to SONIC while maintaining comparable local tracking accuracy. On a physical G1 robot, the same policy supports low-latency, Pico-driven real-time teleoperation for walking, squatting, and single- or double-knee kneeling. It also tracks highly dynamic motions, including a tiger jump transitioning into a shoulder roll, spinning kicks, and cartwheels.

1.1 Research Questions

The central question is: *How can system-level optimization accelerate general humanoid motion-tracker training while preserving tracking quality and deployability?*

This breaks into three concrete questions:

- Which system and training-design choices reduce GPU-hours and wall-clock time?
- How do environment scale, policy capacity, and root-drift handling affect global and local tracking quality?
- Can an efficiently trained policy retain both highly dynamic motion tracking and low-latency real-robot teleoperation?

1.2 Contributions

The main contributions are:

- An efficient training infrastructure that produces a general humanoid motion tracker in 3 hours on 8 RTX 4090 GPUs while achieving strong tracking performance.
- To our knowledge, the first humanoid motion-tracking policy to combine agility and generalizability on a physical robot. A single policy supports low-latency, Pico-driven teleoperation for walking, squatting, and single- or double-knee kneeling. It also tracks highly dynamic motions, including a tiger jump transitioning into a shoulder roll, spinning kicks, and cartwheels.
- Systematic ablations that show how scaling parallel environments and policy capacity improves final tracking quality, and how root-drift truncation preserves local tracking and dynamic-motion completion better than failure termination.

2 Related Work

Humanoid learning. Humanoid reinforcement and imitation learning have progressed from isolated locomotion skills toward unified whole-body behavior. Reinforcement learning optimizes stability and dynamic skills in simulation, while motion capture supplies natural full-body supervision [14, 15]. Recent work combines locomotion, manipulation, and whole-body control under shared policies or pretraining frameworks [3, 8, 18], motivating reusable low-level controllers for downstream interaction rather than isolated task policies.

Universal motion trackers. DeepMimic showed that reference motions can replace hand-designed rewards for physics-based characters [15]. Later systems make tracking a whole-body robot interface: H2O [12], TWIST [20], and TWIST2 [21] support real-time teleoperation and data collection; ASAP [5] adds real residual adaptation; and BeyondMimic [10] and OmniClone [9] connect tracking to generative or engineering-oriented teleoperation. GMT [2], Any2Track [22], KungfuBot2 [4], and SONIC [13] expand universal tracking in data and compute scale. MimicLite complements this work by improving tracking quality while reducing training GPU-hours through a high-throughput, reproducible data-to-deployment stack.

Tracking as a downstream prior. Tracking policies increasingly serve as execution priors for teleoperation, demonstration collection, visual interaction, loco-manipulation, and multimodal control. HDMI [19] and HumanX [17] connect human videos with humanoid interaction skills, while WholeBodyVLA [6] and UniAct [7] connect whole-body motion representations to higher-level action or multimodal policies. Efficient tracker training benefits these directions because low-level interfaces, observations, and deployment assumptions often change with the downstream task.

3 System and Method

3.1 Motion Data Layer

MimicLite uses robot generalized coordinates, paired with a robot-model reference, as the canonical on-disk motion representation. This serves as a minimal exchange representation: Cartesian body positions and orientations are derived from the same configuration and therefore need not be duplicated in every motion file. The compact format reduces storage and network-transfer overhead, making motion datasets faster to download and distribute. Motions from LAFAN, 100STYLE, SONIC, and Pico-collected Real data are normalized into this representation; Figure 2 summarizes the current motion libraries.

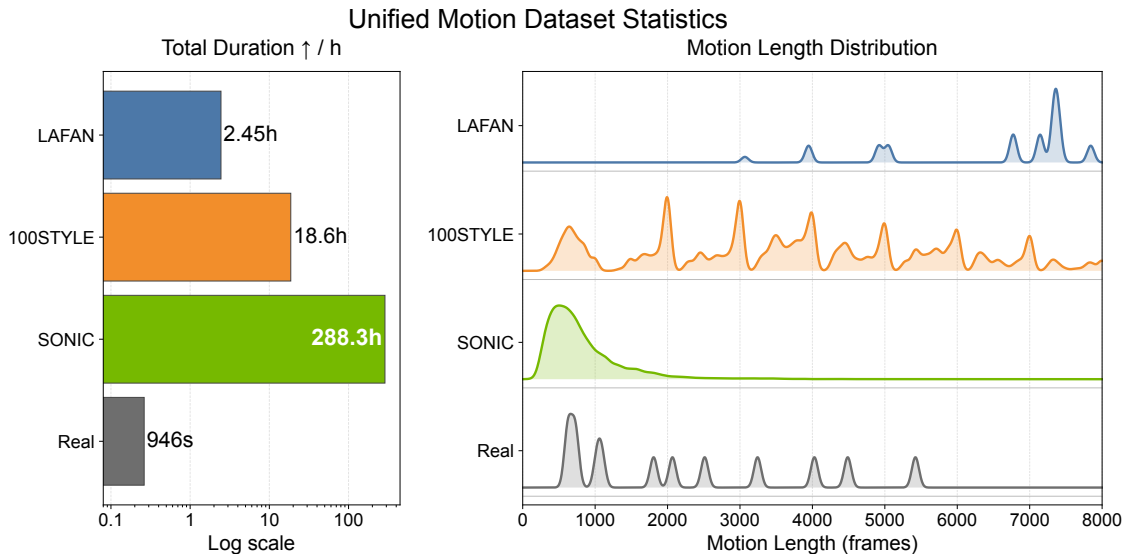


Figure 2: Unified motion datasets used by the tracking pipeline. LAFAN provides high-dynamic reference motions, 100STYLE adds stylized gait diversity, SONIC provides larger-scale motion coverage, and Real contains Pico-collected deployment-like inputs.

Compact storage creates a computation tradeoff because training consumes dense kinematic quantities, including body positions and orientations, that are not stored in the exchange representation. Recomputing forward kinematics whenever a dataset is loaded would reduce the benefit of the compact format. MimicLite resolves this tradeoff with a persistent dataset-level FK cache. On the first load, the loader computes forward kinematics once over the dataset and materializes the derived states required by training. Subsequent loads read these compute-ready states directly from the cache. The source dataset therefore remains compact for exchange and storage, while repeated training runs avoid redundant FK computation.

3.2 Task, Observation, and Action Interface

The task is full-body humanoid motion tracking. At each control step, the policy receives robot proprioception and a reference-motion command, then predicts target joint positions. Before execution, these targets are scaled by joint-specific action ranges, passed through randomized delays, and interpolated over time to mimic the latency of the real robot. A low-level PD controller then converts the processed targets into torques. This design keeps the action interface used during training consistent with deployment.

Observations are organized into three groups:

- **policy**: deployable proprioception, including root angular velocity, projected gravity, joint positions, joint velocities, and recent actions.
- **command**: a short four-frame window of future reference motion.
- **priv**: training-only information for the critic, such as applied torques, body height, and direct tracking-error signals.

The actor receives **policy** and **command**, while the critic also receives **priv**. The four-frame window anticipates upcoming motion without compromising interactive latency.

3.3 Rewards and Termination

Rewards combine tracking objectives for root and local-body pose and velocity, joint positions, and joint velocities with regularization terms for action-rate jitter, excessive joint velocity, joint-limit violations, self-collision, and foot sliding. Each tracking reward uses an exponential error form:

$$r_k = w_k \exp(-e_k/\sigma_k), \quad (1)$$

where e_k is the corresponding pose, velocity, or joint-space error.

Termination captures sustained tracking failure: an episode terminates when root-orientation or local body-position error exceeds its threshold for the configured number of consecutive steps. Because global root drift is only partially observable from deployable policy inputs, exceeding its bound truncates the rollout rather than constituting failure termination.

3.4 Training Stack

The training stack is built on [Roboparty/MimicLite](#). The environment owns simulation stepping, motion sampling, observation construction, reward computation, and termination logic, while the algorithm owns PPO optimization. This separation keeps task and simulation logic independent of the optimizer implementation.

PPO training uses fixed-length rollouts, GAE, clipped policy updates, and entropy regularization:

$$\mathcal{L}_\pi^{\text{PPO}} = -\mathbb{E}_t \left[\min \left(\rho_t \hat{A}_t, \text{clip}(\rho_t, 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) - \beta_H \mathcal{H}(\pi(\cdot \mid o_t)) \right]. \quad (2)$$

To examine whether system-level efficiency gains are specific to one optimization paradigm, MimicLite supports both on-policy and off-policy training. PPO provides the primary on-policy path described above, while the same environment and task interfaces also support an off-policy SAC path, keeping the tracking system fixed while changing the optimization and data-reuse strategy.

For the off-policy comparison, we train SAC with replay, double Q networks, automatic entropy tuning, and an unsquashed Gaussian policy to test whether sample reuse reduces fresh interaction and accelerates early learning. Under the matched protocol, PPO yields stronger long-horizon completion, while SAC serves as a sample-reuse baseline.

3.5 Deployment Interface

MimicLite’s deployment interface connects trained trackers to both MuJoCo validation and physical G1 execution. Its design addresses three practical requirements: accommodating codebase-specific observation conventions, isolating hardware-specific communication, and maintaining a temporally continuous motion reference.

The modular observation interface separates policy-specific input construction from the shared deployment runtime. Importing a policy trained in another codebase requires an observation class that reconstructs its expected inputs and a YAML specification that defines their ordering and parameters; the inference, simulator, and robot interfaces remain unchanged. We have used this interface to integrate policies from HEFT, TeleopIT, Humanoid-GPT, BFM-Zero, SONIC, and TWIST2. The same adapter is used for integrated MuJoCo evaluation and real-robot execution, allowing each imported policy to follow a common path from matched evaluation to deployment.

Policy inference is decoupled from robot I/O through a common robot-state and joint-command interface shared by interchangeable MuJoCo and physical-G1 backends.

For live teleoperation, a timestamped reference buffer aligns and interpolates the policy’s future-motion window. At segment boundaries or after source reconnection, a planar translation and yaw transform preserves root continuity. During stream stalls, the buffer holds the last valid transformed frame and sets reference velocities to zero rather than blocking or resetting the policy. These mechanisms prevent input discontinuities during resets and transient stream failures.

4 Results

4.1 MimicLite System Capabilities

We train three MimicLite policy variants that scale actor capacity and distributed rollout size while keeping 8192 environments per GPU. MimicLite-Small uses a three-layer, 128-unit actor, MimicLite-Base increases the actor width to 256 units, and MimicLite-Huge increases it to 1024 units. All three policies use a 4000-update training budget. Table 1 summarizes their configurations and wall-clock times.

Policy	Actor hidden dimensions	Parallel environments	Iterations	Wall-clock time
MimicLite-Small	[128, 128, 128]	4×8192	4000	3 h 00 min
MimicLite-Base	[256, 256, 256]	8×8192	4000	2 h 57 min
MimicLite-Huge	[1024, 1024, 1024]	32×8192	4000	3 h 30 min

Table 1: Training configurations and total wall-clock times for the three MimicLite policy variants on NVIDIA RTX 4090 GPUs.

The modular observation interface introduced above enables all baselines to use the same evaluation path. Figure 3 compares the three MimicLite variants with SONIC [13], HEFT [11], TeleopIt [1], Humanoid-GPT [16], BFM-Zero [8], and TWIST2 [21] through common MuJoCo rollouts with aligned termination rules and metric definitions. LAFAN-40 evaluates the ability to follow dynamic motions using full-motion completion. PHUMA-30 evaluates out-of-distribution motion tracking. Root-8 consists of Pico-recorded walking commands in forward, backward, and lateral directions and evaluates global root tracking using final XY displacement error. The larger MimicLite policies retain more progress on dynamic LAFAN motions, while SONIC and HEFT remain strongest on selected PHUMA-30 metrics. MimicLite-Huge also reduces final XY root displacement error relative to SONIC and TWIST2 in all three directions.

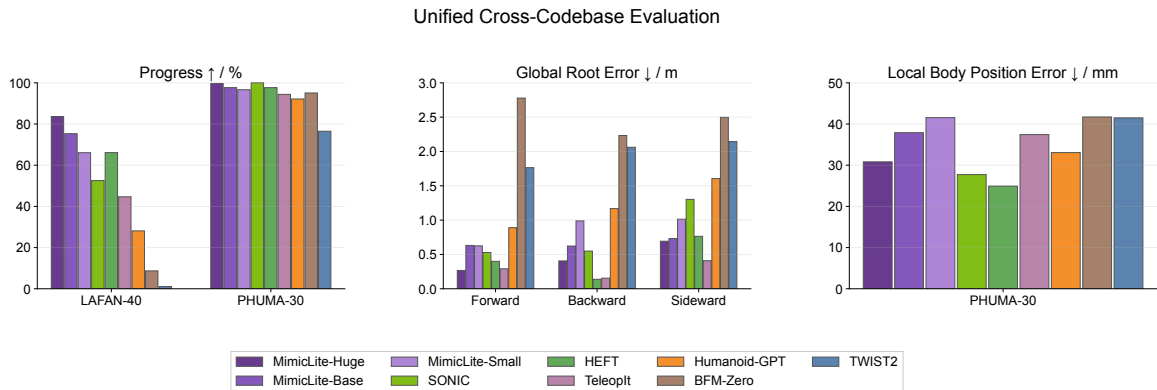


Figure 3: Unified MuJoCo evaluation of nine imported policies. Progress is reported on LAFAN-40 using complete motion lengths and on PHUMA-30 using a 2000-step horizon. Root-8 reports final XY displacement by direction. PHUMA-30 local body-position error uses each frame’s pelvis XY and projected-yaw frame while retaining world-frame height. Bars report means over the selected motion sets.

4.2 Simulation and Real-Robot Deployment Results

The shared policy interface also supports real-time teleoperation. In MuJoCo, Pico motion input is converted online into reference-motion commands; the policy then reads the robot state and reference windows and outputs low-level joint targets. Figure 4 shows representative frames from this loop, demonstrating that the policy handles streaming human input in addition to fixed offline motion files.

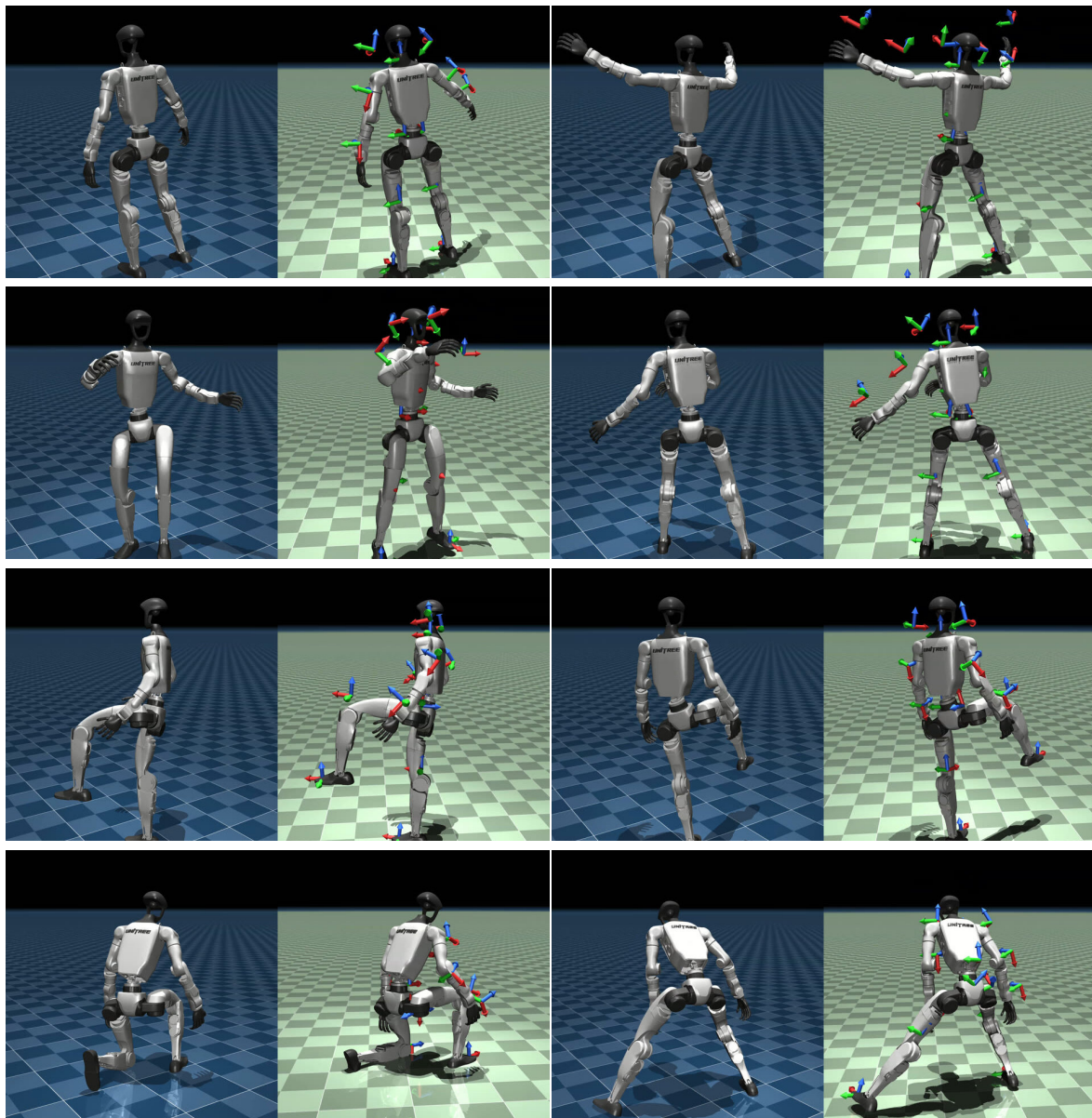


Figure 4: Eight representative frames from real-time MuJoCo teleoperation. The policy tracks streaming reference-motion input with the same interface used for offline replay and real-robot deployment.

The final policy transfers from MuJoCo validation to a real G1 execution loop without changes to its logic. Pico input is converted into reference motions, the policy produces low-level control targets, and the robot bridge sends those targets to the hardware. The resulting interface supports low-latency real-time control. Figure 5 shows representative real-robot frames for turning, side stepping, shuttle running, kneeling down, and standing up.



Figure 5: Representative frames from real-robot teleoperation. Rows show turning, side stepping, shuttle running, kneeling down, and standing up; columns show four moments within each motion.

4.3 Why MimicLite Is More Efficient

MimicLite’s efficiency has two components: system throughput and learning efficiency. We report system throughput as environment frames simulated per GPU per second and aggregate GPU-hours for a fixed 4000-update training budget. Learning efficiency is the tracking performance achieved by the resulting checkpoint under that budget.

Three implementation choices account for the high system throughput. First, MimicLite uses the mujlab backend for its main training runs. Its batched, GPU-native MuJoCo execution keeps the headless simulation loop compact and scales to thousands of environments per GPU; Figure 6 separately measures the backend contribution by running the same Huge-aligned task through mujlab and IsaacLab. Second, the tracking workload repeatedly converts reference and robot body states between world, root-yaw, and local frames. MimicLite compiles these body-transform, quaternion, future-reference, and tracking-error computations with `torch.compile`. Compilation fuses tensor operations across the environment and body dimensions, reducing Python dispatch and intermediate-kernel overhead in the per-step path.

Third, large motion libraries cannot be replicated in full on every GPU. The loader therefore partitions motions into disjoint, motion-aligned rank shards before selecting the runtime storage strategy. Each rank places only its local FP16 shard on the GPU, while the shards collectively cover the complete filtered dataset. This reduces per-rank dataset memory and avoids redundant host-to-device traffic without changing the simulator or PPO code. When even a shard cannot remain resident, the same partition stage feeds a windowed runtime with persistent GPU pools and asynchronous prefetching. Together, these choices reduce the three main sources of per-step overhead: simulation, tracking-state computation, and motion-data movement.

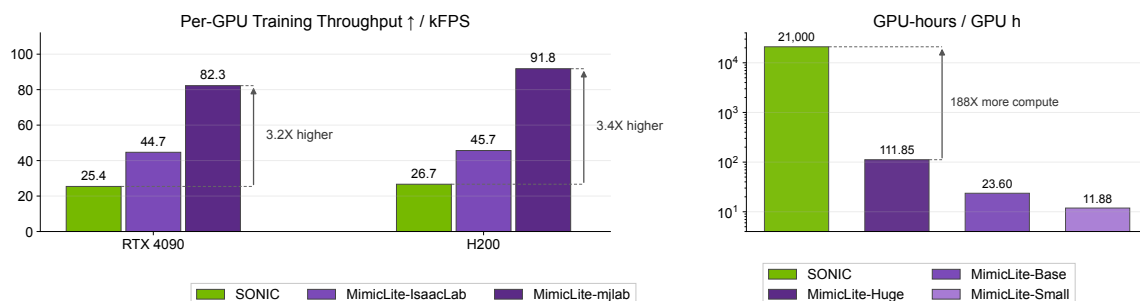


Figure 6: Training efficiency. Throughput is averaged over the final 32 of 100 updates using eight GPUs, 4096 environments per GPU, 32 steps per update, plane-terrain LAFAN motion, and Huge-aligned networks. GPU-hours use the 4000-update RTX 4090 results for Huge, Base, and Small, together with SONIC’s reported total.

A major factor affecting learning efficiency is root-drift handling. The deployable policy does not observe absolute world-frame root position, yet horizontal drift accumulates over long sequences. If drift is unconstrained, the global root reward depends on an absolute target that the policy cannot infer: similar observable states can receive different rewards as their hidden displacement changes. The bounded reward also decays toward zero at large errors, leaving too little signal to recover. This weak and ambiguous objective can interfere with otherwise valid local pose and velocity tracking.

Treating drift as failure creates the opposite problem. Horizontal displacement does not necessarily mean that the local motion has failed, but failure termination labels the trajectory as a

negative example, removes value bootstrapping, and stops exploration of highly dynamic motion. MimicLite instead truncates the sampled segment beyond 0.4 m while retaining value-function bootstrapping. Truncation keeps the global target in a meaningful range without turning a partially observable displacement into motion failure.

We compare SONIC, MimicLite, and MimicLite* on the same LAFAN+100STYLE+Real mixture. The SONIC policies here are our 4,000-update, 8×8192 -environment ablation runs, not the released policy in Figure 3. MimicLite* keeps MimicLite’s codebase, data, algorithm, and network but adopts SONIC-aligned strict local-body termination, isolating the effect of the non-root termination set. For each configuration we compare root-drift truncation, failure termination, and no constraint. Global root error measures long-horizon drift, SONIC-subset local body-position error measures pose tracking, and LAFAN progress exposes settings that obtain low error only by ending difficult trajectories early.

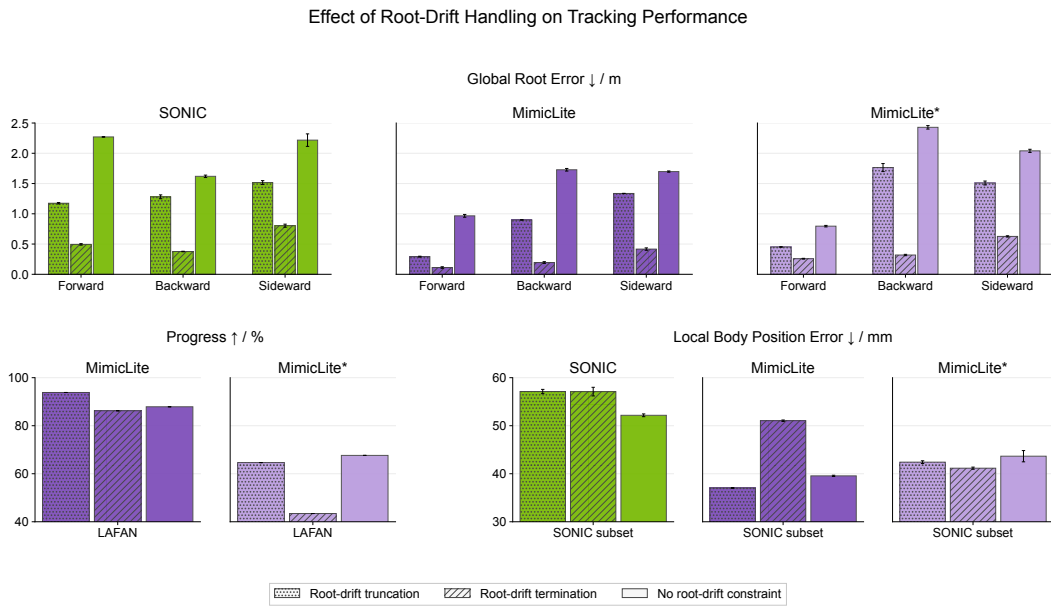


Figure 7: Root-drift handling under default and strict local-body termination. MimicLite* applies SONIC-aligned strict local-body termination to MimicLite. Each configuration varies root handling among truncation, failure termination, and no constraint; errors are lower when better, while progress is higher when better.

Figure 7 shows three consistent effects. First, removing the root constraint greatly increases global error for SONIC, MimicLite, and MimicLite* across walking directions, confirming that the saturated root reward cannot correct long-horizon drift by itself. Second, both truncation and failure termination reduce global error, but failure termination gives the default MimicLite policy a worse overall tradeoff: local error rises and LAFAN progress falls relative to truncation. This is consistent with the hard failure signal disrupting local-pose learning and dynamic-motion exploration.

Third, the MimicLite–MimicLite* control shows why local termination must be matched. The strict SONIC-aligned set removes difficult trajectories early, making local errors appear closer while reducing progress; failure termination gives the lowest MimicLite* progress. Strict local termination therefore masks part of root handling’s effect instead of improving the overall trade-off. Root-drift truncation provides the best balance of global tracking, local pose accuracy, and dynamic-motion completion.

5 Compute Ablations

The compute ablations use PPO and distinguish training dynamics from final matched evaluation. LAFAN progress is measured with a 512-environment mjlabs rollout and horizon normalization by $\min(\text{motion length}, 2000)$. SONIC-subset local body-position error follows the same mjlabs protocol. Global root error is measured by integrated sim2sim evaluation on the eight-motion root-tracking set, using final XY displacement for forward, backward, and sideward motions. Final-evaluation bars report mean metrics over the selected motion sets.

5.1 Environment Scale

The fixed-total-environment group compares 1×4096 , 2×2048 , 4×1024 , and 8×512 . Figure 8 shows consistent training dynamics across distributed configurations with 4096 total environments, indicating that distribution preserves the effective PPO recipe.

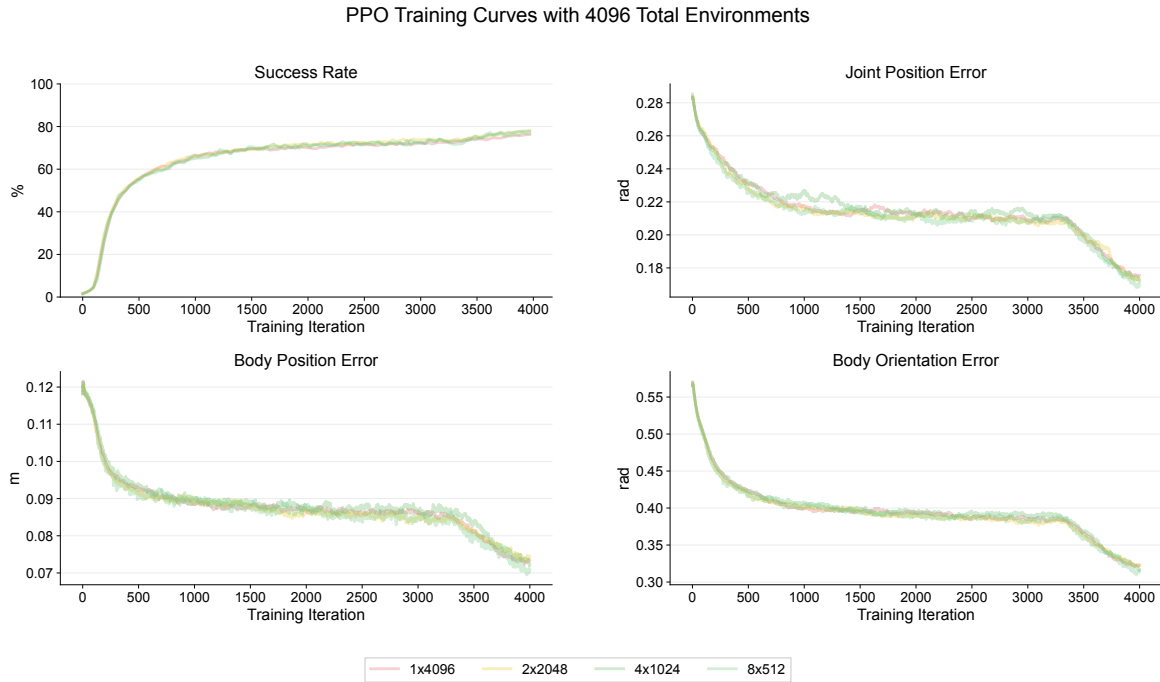


Figure 8: PPO training curves at fixed total environment count. Different parallelisms with the same 4096 total environments show consistent dynamics.

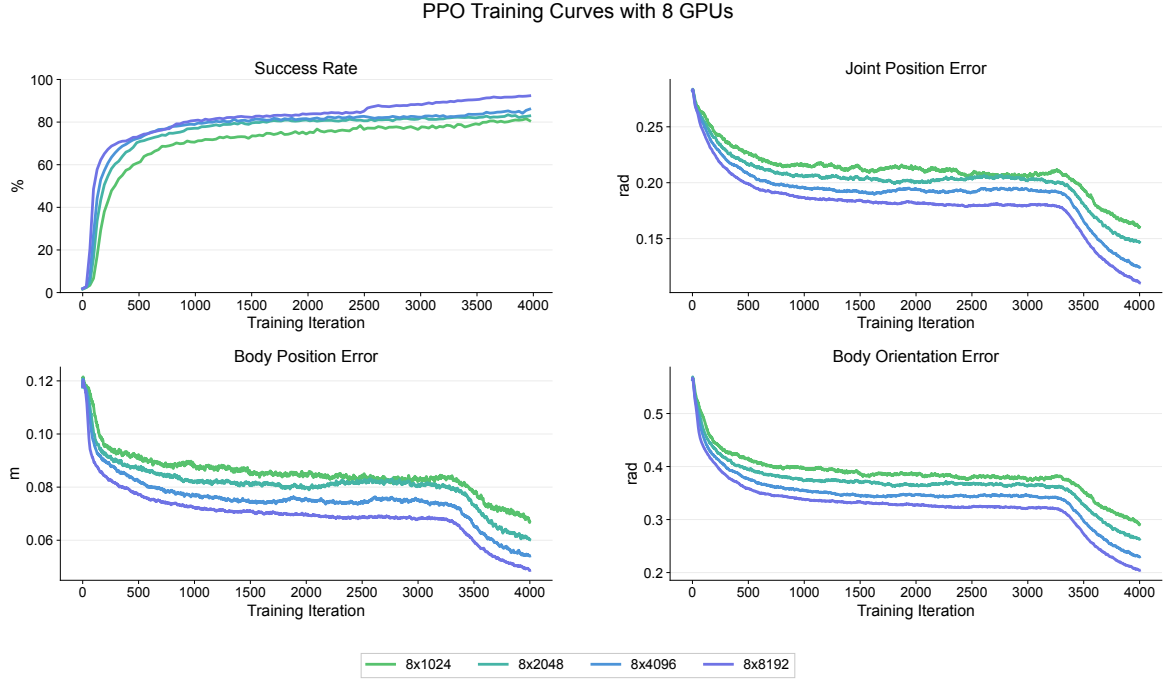


Figure 9: PPO training curves at fixed 8-GPU hardware scale. Larger per-GPU environment counts change training dynamics.

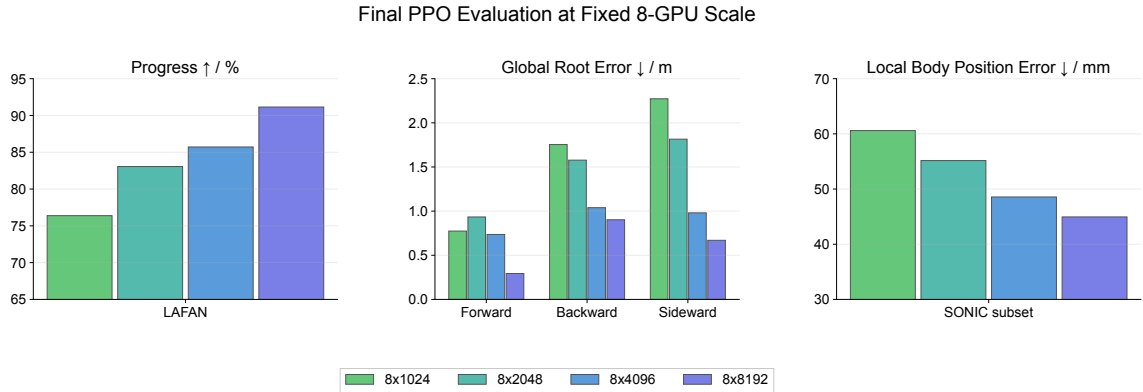


Figure 10: Final evaluation at fixed 8-GPU hardware scale. Increasing parallel environments improves LAFAN progress and direction-split global root tracking.

The fixed-8-GPU group compares 8×1024 , 8×2048 , 8×4096 , and 8×8192 . Figures 9 and 10 show that increasing the per-GPU environment count improves final tracking quality, particularly LAFAN progress. The 8×8192 setting therefore remains the practical default when final policy quality is the priority.

5.2 Architecture and Parameter Scale

The module ablation fixes training at 8×8192 . Width scaling compares small, base, large, and huge actors; it tests capacity at fixed depth, where final-quality gains are clearer than monotonic training-speed gains. Depth scaling adds base-deep and large-deep variants.

Configuration	Actor hidden dimensions	Configuration	Actor hidden dimensions
small	[128, 128, 128]	huge	[1024, 1024, 1024]
base	[256, 256, 256]	base-deep	[256, 256, 256, 256, 256]
large	[512, 512, 512]	large-deep	[512, 512, 512, 512, 512]

Table 2: Policy-network configurations in the PPO module-scale ablation.

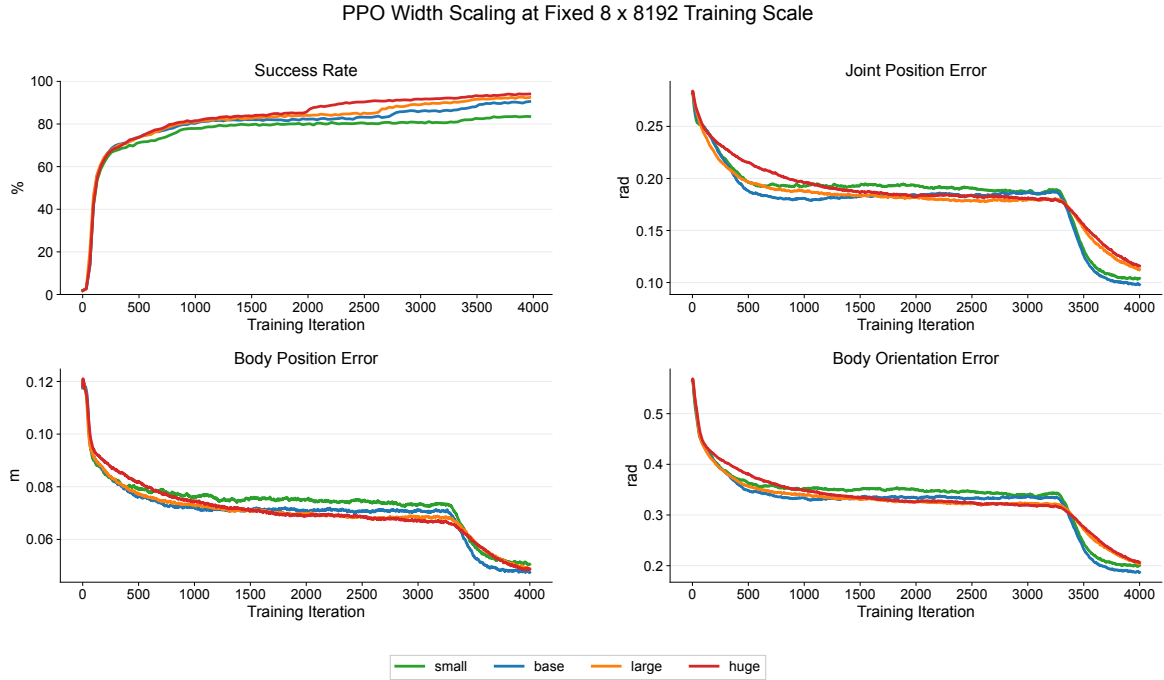


Figure 11: Training curves for width scaling. Wider actors increase model capacity but do not exhibit uniformly better optimization behavior.

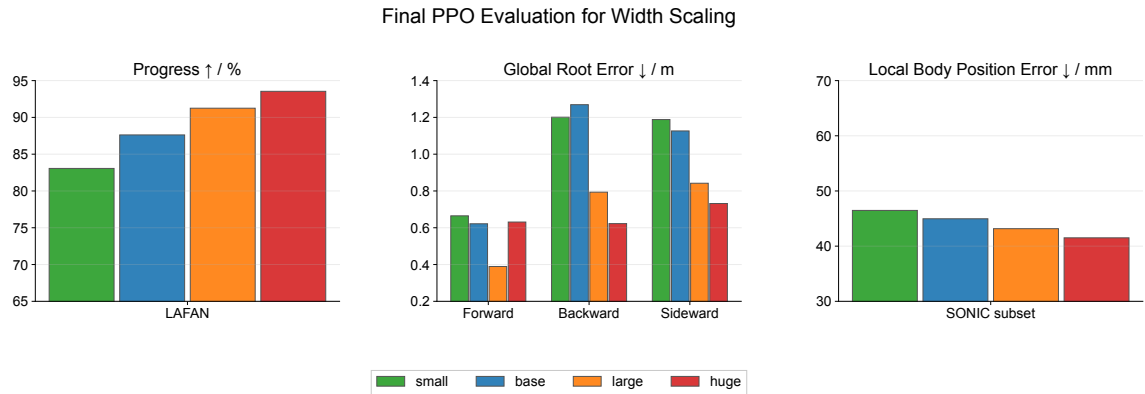


Figure 12: Final tracking quality for width scaling. Increasing actor width generally improves matched-evaluation performance.

Increasing actor width generally improves final tracking quality, although wider models do not show uniformly better optimization behavior during training. Figures 11 and 12 show that width scaling improves several matched-evaluation metrics, but the corresponding training curves do not improve monotonically with model size.



Figure 13: Training curves for depth scaling. Deeper actors can show worse reward-side tracking curves.

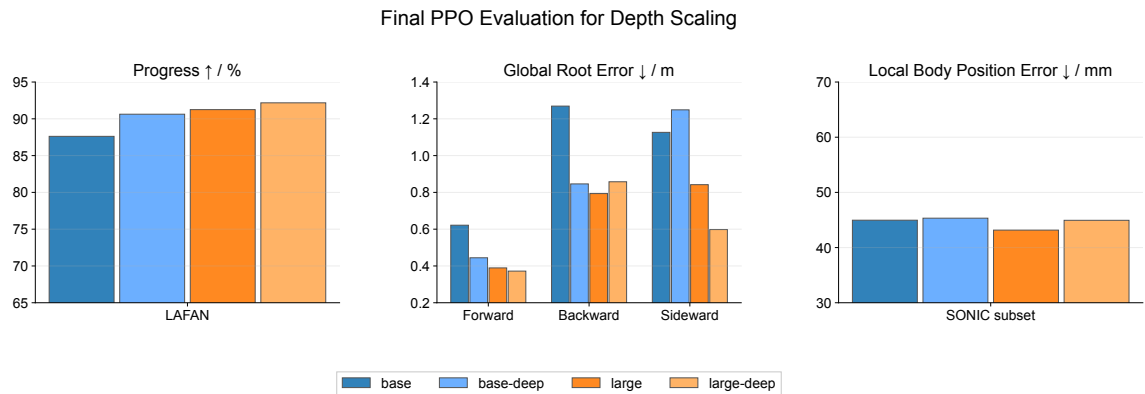


Figure 14: Final tracking quality for depth scaling. Deeper actors can still improve matched final evaluation.

Depth scaling shows that better final tracking need not coincide with better training curves. Figures 13 and 14 show worse reward-side tracking curves for deeper actors, especially local body tracking, while selected final metrics can still improve. Network depth therefore changes optimization behavior, so architecture choices should be judged by matched final evaluation rather than intermediate reward curves alone.

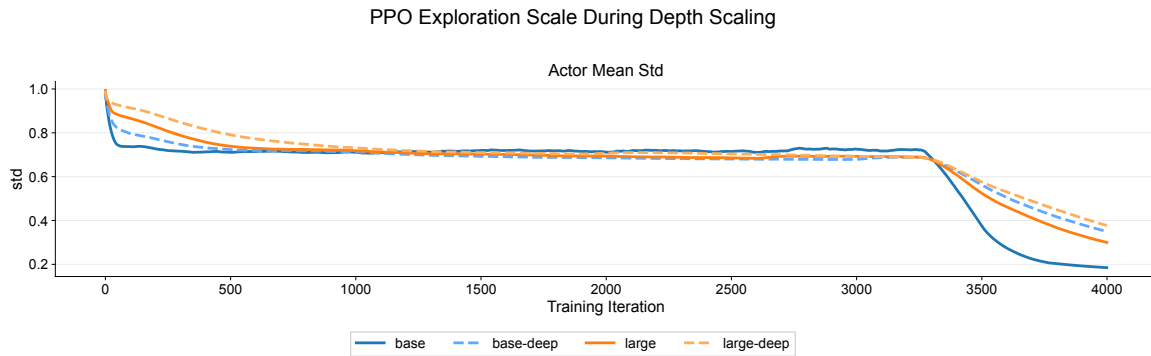


Figure 15: Actor standard deviation during depth scaling. Deeper actors maintain a larger exploration scale during PPO updates.

Figure 15 helps explain the mismatch with the reward curves: deeper actors maintain higher action standard deviation during much of training, increasing exploration noise and altering the optimization trajectory. The depth comparison must therefore be interpreted using both training dynamics and final evaluation.

5.3 Ablation Summary

Taken together, the ablations reveal four consistent trends:

- At a fixed total environment count, redistributing environments across GPUs leaves the training dynamics largely unchanged, indicating stable distributed scaling.
- On fixed eight-GPU hardware, increasing the number of environments per GPU improves final tracking quality; we therefore adopt 8×8192 as the default setting.
- Increasing actor width generally improves matched-evaluation performance, while the large three-layer actor provides the strongest balance between tracking quality and computational efficiency.
- Additional depth improves some final metrics but also increases exploration noise and makes optimization less consistent, producing no uniform benefit.

6 Discussion

System-level sources of efficiency. MimicLite’s efficiency comes from the joint design of simulation throughput, parallel rollout scale, policy capacity, and termination semantics. The mjlab backend increases headless simulation throughput, large parallel rollouts improve final tracking, and root-drift truncation preserves a consistent learning target. Together, these choices reduce training GPU-hours and shorten the iteration cycle for changes to data, observations, rewards, and termination settings while preserving strong matched-evaluation performance.

Limits of the current evidence. The evidence is strongest for simulation tracking, training efficiency, and the real-time teleoperation pipeline, but remains limited for broad quantitative real-robot evaluation. The physical-robot results demonstrate live operation and motion coverage; broader claims of hardware robustness will require standardized real-world trials, disturbance tests, and systematic failure analysis.

Architectural scope. The policy study is restricted to feed-forward MLP actors. We do not evaluate transformer-based or flow-matching architectures, so the architecture ablations characterize scaling within the MLP family rather than establishing an architecture-independent optimum.

Use as a downstream prior. MimicLite’s value extends beyond the retained policy to the ability to retrain and modify the low-level tracker efficiently as downstream projects change cameras, action interfaces, teleoperation latency, reward terms, or data distributions. It therefore provides a reusable base layer for subsequent teleoperation, data collection, and whole-body manipulation work.

7 Conclusion

MimicLite demonstrates efficient general humanoid motion tracking through system co-design. Its Small, Base, and Huge PPO policies train in a few hours; increased scale improves dynamic-motion following, and MimicLite-Huge reduces Root-8 global root error relative to SONIC and TWIST2 while remaining competitive on PHUMA-30. The modular observation interface unifies external-policy evaluation and deployment, while one G1 policy supports low-latency Pico teleoperation and highly dynamic motions.

The ablations show stable distributed scaling, gains from parallel environments and actor width, no consistent benefit from added depth, and a better learning target from root-drift truncation. These findings motivate joint design of throughput, rollout scale, capacity, and termination. The study remains limited to MLP actors and primarily qualitative hardware results; future work should evaluate transformer and flow-matching policies and standardize real-robot benchmarks.

Acknowledgments

Robotparty Lab acknowledges the shared infrastructure and robot platform support behind this technical report.

References

- [1] BotRunner64. Teleopit: Lightweight, extensible whole-body teleoperation framework for humanoid robots. <https://github.com/BotRunner64/Teleopit>, 2026. GitHub repository.
- [2] Zixuan Chen, Mazeyu Ji, Xuxin Cheng, Xuanbin Peng, Xue Bin Peng, and Xiaolong Wang. Gmt: General motion tracking for humanoid whole-body control, 2025. URL <https://arxiv.org/abs/2506.14770>.
- [3] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion, 2022. URL <https://arxiv.org/abs/2210.10044>.
- [4] Jinrui Han, Weiji Xie, Jiakun Zheng, Jiyuan Shi, Weinan Zhang, Ting Xiao, and Chenjia Bai. Kungfubot2: Learning versatile motion skills for humanoid whole-body control, 2025. URL <https://arxiv.org/abs/2509.16638>.
- [5] Tairan He, Jiawei Gao, Wenli Xiao, Yuanhang Zhang, Zi Wang, Jiashun Wang, Zhengyi Luo, Guanqi He, Nikhil Sobanbab, Chaoyi Pan, Zeji Yi, Guannan Qu, Kris Kitani, Jessica Hodgins, Linxi Fan, Yuke Zhu, Changliu Liu, and Guanya Shi. Asap: Aligning simulation and real-world physics for learning agile humanoid whole-body skills, 2025. URL <https://arxiv.org/abs/2502.01143>.
- [6] Haoran Jiang, Jin Chen, Qingwen Bu, Li Chen, Modi Shi, Yanjie Zhang, DeLong Li, Chuanzhe Suo, Chuang Wang, Zhihui Peng, and Hongyang Li. Wholebodyvla: Towards unified latent vla for whole-body loco-manipulation control, 2025. URL <https://arxiv.org/abs/2512.11047>.
- [7] Nan Jiang, Zimo He, Wanhe Yu, Lexi Pang, Yunhao Li, Hongjie Li, Jieming Cui, Yuhan Li, Yizhou Wang, Yixin Zhu, and Siyuan Huang. Uniact: Unified motion generation and action streaming for humanoid robots, 2025. URL <https://arxiv.org/abs/2512.24321>.
- [8] Yitang Li, Zhengyi Luo, Tonghe Zhang, Cunxi Dai, Anssi Kanervisto, Andrea Tirinzoni, Haoyang Weng, Kris Kitani, Mateusz Guzek, Ahmed Touati, Alessandro Lazaric, Matteo Pirotta, and Guanya Shi. Bfm-zero: A promptable behavioral foundation model for humanoid control using unsupervised reinforcement learning, 2025. URL <https://arxiv.org/abs/2511.04131>.
- [9] Yixuan Li, Le Ma, Yutang Lin, Yushi Du, Mengya Liu, Kaizhe Hu, Jieming Cui, Yixin Zhu, Wei Liang, Baoxiong Jia, and Siyuan Huang. Omniclone: Engineering a robust, all-rounder whole-body humanoid teleoperation system, 2026. URL <https://arxiv.org/abs/2603.14327>.
- [10] Qiayuan Liao, Takara E. Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C. Karen Liu. Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion, 2025. URL <https://arxiv.org/abs/2508.08241>.
- [11] Chenxin Liu, Qingzhou Lu, Guangxiao Yang, Xuanyang Shi, Chenghan Yang, Yanjiang Guo, and Jianyu Chen. Heft: Heavy-payload full-size humanoid teleoperation with privileged motion guidance and windowed payload curriculum, 2026. URL <https://arxiv.org/abs/2607.02332>.
- [12] Zhengyi Luo, Jinkun Cao, Alexander W. Winkler, Kris Kitani, and Weipeng Xu. Learning human-to-humanoid real-time whole-body teleoperation, 2024. URL <https://arxiv.org/abs/2403.04436>.

- [13] Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Sirui Chen, Fernando Castañeda, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, Xingye Da, Runyu Ding, Cyrus Hogg, Lina Song, Edy Lim, Eugene Jeong, Tairan He, Haoru Xue, Wenli Xiao, Zi Wang, Simon Yuen, Jan Kautz, Yan Chang, Umar Iqbal, Linxi "Jim" Fan, and Yuke Zhu. Sonic: Supersizing motion tracking for natural humanoid whole-body control, 2025. URL <https://arxiv.org/abs/2511.07820>.
- [14] Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation, 2017. URL <https://arxiv.org/abs/1707.02201>.
- [15] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills, 2018. URL <https://arxiv.org/abs/1804.02717>.
- [16] Zekun Qi, Xuchuan Chen, Dairu Liu, Chenghuai Lin, Yunrui Lian, Sikai Liang, Zhikai Zhang, Yu Guan, Jilong Wang, Wenyao Zhang, Xinqiang Yu, He Wang, and Li Yi. Humanoid-gpt: Scaling data and structure for zero-shot motion tracking, 2026. URL <https://arxiv.org/abs/2606.03985>. Code: <https://github.com/GalaxyGeneralRobotics/Humanoid-GPT>.
- [17] Yinhuai Wang, Qihan Zhao, Yuen Fui Lau, Runyi Yu, Hok Wai Tsui, Qifeng Chen, Jingbo Wang, Jiangmiao Pang, and Ping Tan. Humanx: Toward agile and generalizable humanoid interaction skills from human videos, 2026. URL <https://arxiv.org/abs/2602.02473>.
- [18] Yuxuan Wang, Ming Yang, Ziluo Ding, Yu Zhang, Weishuai Zeng, Xinrun Xu, Haobin Jiang, and Zongqing Lu. From experts to a generalist: Toward general whole-body control for humanoid robots, 2025. URL <https://arxiv.org/abs/2506.12779>.
- [19] Haoyang Weng, Yitang Li, Nikhil Sobanbabu, Zihan Wang, Zhengyi Luo, Tairan He, Deva Ramanan, and Guanya Shi. Hdmi: Learning interactive humanoid whole-body control from human videos, 2025. URL <https://arxiv.org/abs/2509.16757>.
- [20] Yanjie Ze, Zixuan Chen, Wenhao Wang, Tianyi Chen, Xuanbin Peng, Yuying Ge, Yunchu Zhang, Zerong Zheng, Pieter Abbeel, Angjoo Kanazawa, C. Karen Liu, and Jiajun Wu. Twist: Teleoperated whole-body imitation system, 2025. URL <https://arxiv.org/abs/2505.02833>.
- [21] Yanjie Ze, Siheng Zhao, Weizhuo Wang, Angjoo Kanazawa, Rocky Duan, Pieter Abbeel, Guanya Shi, Jiajun Wu, and C. Karen Liu. Twist2: Scalable, portable, and holistic humanoid data collection system, 2025. URL <https://arxiv.org/abs/2511.02832>.
- [22] Zhikai Zhang, Jun Guo, Chao Chen, Jilong Wang, Chenghuai Lin, Yunrui Lian, Han Xue, Zhenrong Wang, Maoqi Liu, Jiangran Lyu, Huaping Liu, He Wang, and Li Yi. Track any motions under any disturbances, 2025. URL <https://arxiv.org/abs/2509.13833>.