

---

# FastTD3: Simple, Fast, and Capable Reinforcement Learning for Humanoid Control

---

Younggyo Seo<sup>1</sup> Carmelo Sferrazza<sup>1</sup> Haoran Geng<sup>1</sup>

Michal Nauman<sup>1,2</sup> Zhao-Heng Yin<sup>1</sup> Pieter Abbeel<sup>1</sup>

<sup>1</sup>University of California, Berkeley <sup>2</sup>University of Warsaw

[https://younggyo.me/fast\\_td3](https://younggyo.me/fast_td3)

## Abstract

Reinforcement learning (RL) has driven significant progress in robotics, but its complexity and long training times remain major bottlenecks. In this report, we introduce FastTD3, a simple, fast, and capable RL algorithm that significantly speeds up training for humanoid robots in popular suites such as HumanoidBench, IsaacLab, and MuJoCo Playground. Our recipe is remarkably simple: we train an off-policy TD3 agent with several modifications – parallel simulation, large-batch updates, a distributional critic, and carefully tuned hyperparameters. FastTD3 solves a range of HumanoidBench tasks in under 3 hours on a single A100 GPU, while remaining stable during training. We also provide a lightweight and easy-to-use implementation of FastTD3 to accelerate RL research in robotics.



Figure 1: **Sim-to-real reinforcement learning with FastTD3.** We successfully transfer the FastTD3 policy trained in MuJoCo Playground (Zakka et al., 2025) to Booster T1. This represents, to the best of our knowledge, the first documented instance of a successful deployment of an off-policy RL policy on a full-size humanoid robot in the real world. See our [project page](#) for videos.

## 1 Introduction

Reinforcement learning (RL) has been a key driver behind recent successes in robotics, enabling the successful transfer of robust simulation policies to real-world environments (Hwangbo et al., 2019; Kaufmann et al., 2023). However, progress is often bottlenecked by slow training times in complex tasks. For example, in the recently proposed benchmark HumanoidBench (Sferrazza et al., 2024), even state-of-the-art RL algorithms failed to solve many tasks after 48 hours of training.

This slow training remains a major bottleneck for practitioners aiming to unlock new behaviors in humanoid robots using RL. In particular, the iterative nature of reward design in robotics – where multiple rounds of reward shaping and policy retraining are often necessary – demands RL algorithms that are not only *capable* but also significantly *faster*. These algorithms must support rapid iteration and reliably solve tasks when given well-designed rewards.

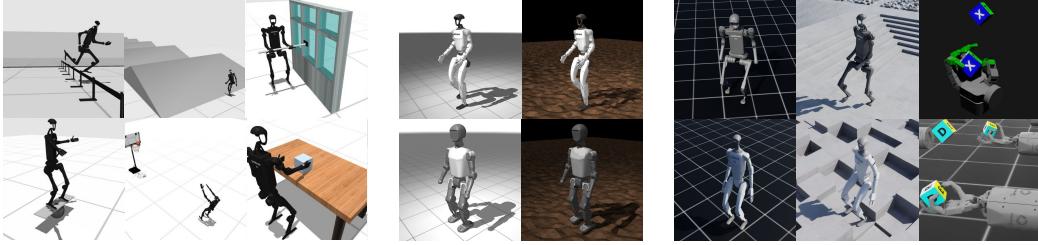


Figure 2: **Tasks.** We consider a range of challenging humanoid and dexterous control tasks from HumanoidBench (left), MuJoCo Playground (middle), and IsaacLab (right).

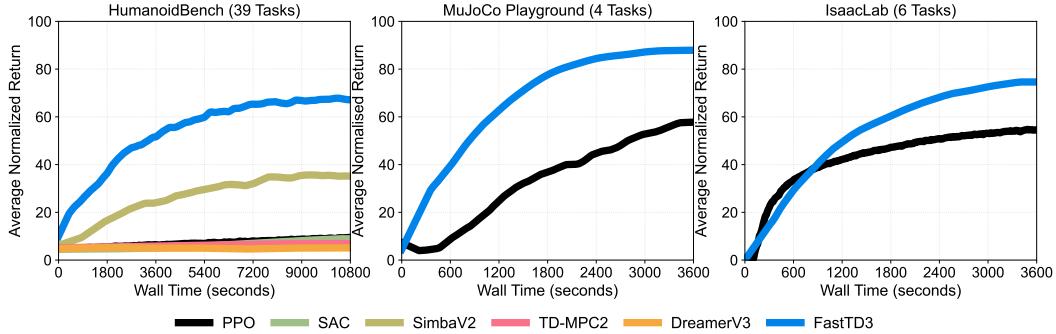


Figure 3: **Summary of results.** FastTD3 is a simple, fast, and capable RL algorithm that significantly speeds up training for humanoid robots on tasks from popular suites such as HumanoidBench (Sferrazza et al., 2024), IsaacLab (Mittal et al., 2023), and MuJoCo Playground (Zakka et al., 2025). To accelerate RL research in robotics, we provide an easy-to-use open-source implementation of FastTD3, enabling users to easily reproduce these results or build upon our work.

For that purpose, practitioners have mostly used Proximal Policy Optimization (PPO; Schulman et al. 2017) for training deployable policies in simulation, as PPO learns behaviors very fast with massively parallel simulation (Heess et al., 2017; Hwangbo et al., 2019). However, PPO is an on-policy algorithm and not sample-efficient, making it difficult to fine-tune it during real-world deployment or initialize training with demonstrations (Hester et al., 2018).

Meanwhile, recent off-policy RL research has made significant progress in improving sample-efficiency (D’Oro et al., 2023; Nauman et al., 2024b; Hansen et al., 2024). However, this line of work often suffers from increased algorithmic complexity and long wall-clock training times, making them difficult to be widely used for learning deployable policies in robotics.

On the other hand, Parallel Q-Learning (PQL; Li et al. 2023b) have shown that off-policy RL can be both fast and sample-efficient by scaling up through massively parallel simulation, large batch sizes, and a distributional critic (Bellemare et al., 2017). However, the core contribution of PQL – its use of asynchronous parallel processes that cuts wall-clock time – unfortunately comes with high implementation complexity, which has hindered its widespread adoption. In this work, we do not aim to claim novelty over PQL, but rather focus on developing a simple yet highly-optimized algorithm without asynchronous processes, highlighting its effectiveness on popular humanoid control suites, and providing the easy-to-use implementation to accelerate future RL research on robotics.

**FastTD3** We introduce FastTD3, a simple, fast, and capable RL algorithm that significantly speeds up training for humanoid robots on tasks from popular suites such as HumanoidBench (Sferrazza et al., 2024), IsaacLab (Mittal et al., 2023), and MuJoCo Playground (Zakka et al., 2025). Our recipe is remarkably simple: by training an off-policy TD3 agent (Fujimoto et al., 2018) with parallel simulation, large-batch updates, a distributional critic (Bellemare et al., 2017), and carefully tuned hyperparameters, FastTD3 solves a range of HumanoidBench tasks in under 3 hours on a single GPU. Compared to PPO (Schulman et al., 2017), FastTD3 trains humanoid locomotion policies faster in IsaacLab and MuJoCo Playground, particularly on rough terrain with domain randomization.

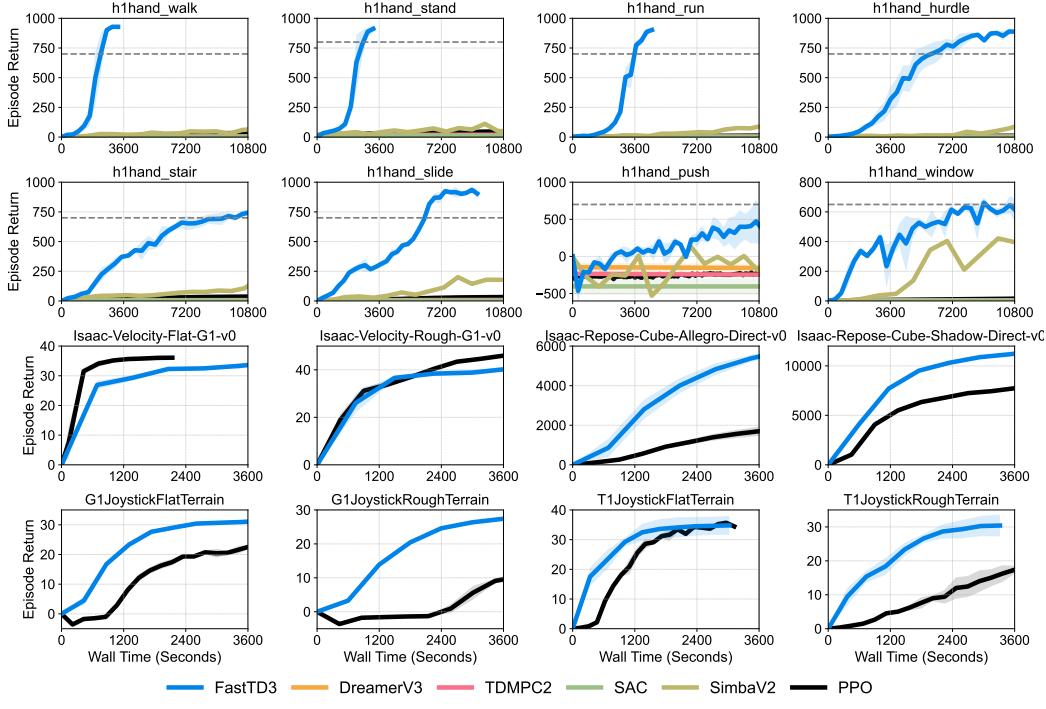


Figure 4: **Results on a selected set of tasks.** Learning curves on selected individual tasks from HumanoidBench (first two rows), IsaacLab (third row), and MuJoCo Playground (fourth row). The solid line and shaded regions represent the mean and standard deviation across three runs. The dashed lines indicate success thresholds in HumanoidBench tasks.

**Open-source implementation** We provide an open-source implementation of FastTD3 based on PyTorch (Paszke et al., 2019) – a lightweight codebase that enables users to easily build new ideas on top of FastTD3. Our implementation is easy-to-install and versatile – users can easily train FastTD3 agents on HumanoidBench, IsaacLab, and MuJoCo Playground, and the codebase also supports several user-friendly features, such as preconfigured hyperparameters, rendering support, logging, and loading checkpoints. We note that our work is orthogonal to latest RL research, such as SR-SAC (D’Oro et al., 2023), BBF (Schwarzer et al., 2023), BRO (Nauman et al., 2024b), Simba (Lee et al., 2024), NaP (Lyle et al., 2024), TDMPC2 (Hansen et al., 2024), TDMPBC (Zhuang et al., 2025), SimbaV2 (Lee et al., 2025), MAD-TD (Voelcker et al., 2025), and MR.Q (Fujimoto et al., 2025), we expect various improvements in these works to be also useful when incorporated into FastTD3.

Our key contributions can be summarized as follows:

- We introduce FastTD3, a simple, fast and capable RL algorithm that solves a variety of locomotion and manipulation tasks that prior RL algorithms take tens of hours to complete or fail to solve. We demonstrate that this performance can be achieved using a remarkably simple recipe: training a TD3 agent (Fujimoto et al., 2018) with large-batch updates, parallel simulation, distributional RL, and well-tuned hyperparameters.
- We provide experimental results that show the effectiveness of the various design choices.
- We release an easy-to-use open-source implementation of FastTD3 to accelerate RL research on robotics. This implementation supports popular suites such as HumanoidBench (Sferrazza et al., 2024), IsaacLab (Mittal et al., 2023), and MuJoCo Playground (Zakka et al., 2025).

## 2 FastTD3: Simple, Fast, Capable RL for Humanoid Control

FastTD3 is a high-performance variant of the Twin Delayed Deep Deterministic Policy Gradient (TD3; Fujimoto et al. 2018) algorithm, optimized for complex robotics tasks. These optimizations are based on the observations of Li et al. (2023b) that found parallel simulation, large batch sizes, and a distributional critic are important for achieving strong performance with off-policy RL algorithms.

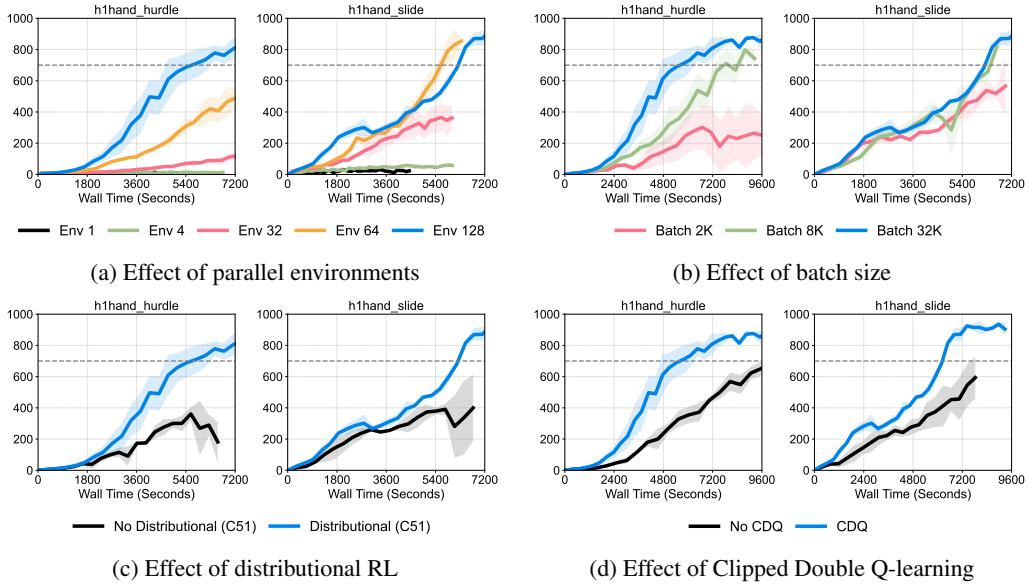


Figure 5: **Effect of design choices (1 / 2).** We investigate the effect of (a) parallel environments, (b) batch size, (c) distributional RL, and (d) Clipped Double Q-learning. The solid line and shaded regions represent the mean and standard deviation across three runs.

## 2.1 Design Choices

In this section, we describe the key design choices made in the development of FastTD3 and their impact on performance. For details of TD3, we refer the reader to [Fujimoto et al. \(2018\)](#).

**Parallel environments** Similar to observations in [Li et al. \(2023a\)](#), we find that using massively parallel environments significantly accelerates TD3 training. We hypothesize that combining deterministic policy gradient algorithms ([Silver et al., 2014](#)) with parallel simulation is particularly effective, because the randomness from parallel environments increases diversity in the data distribution. This enables TD3 to leverage its strength – efficient exploitation of value functions – while mitigating its weakness in exploration.

**Large-batch training** We find that using an unusually large batch size of 32,768 for training the FastTD3 agent is highly effective. We hypothesize that, with massively parallel environments, large-batch updates provide a more stable learning signal for the critic by ensuring high data diversity in each gradient update. Otherwise, unless with high update-to-data ratio, a large portion of data will never be seen by the agent. While increasing the batch size incurs a higher per-update wall-clock time, it often reduces overall training time due to improved training efficiency.

**Distributional RL** We also find that using the distributional critic ([Bellemare et al., 2017](#)) is helpful in most cases, similar to the observation of [Li et al. \(2023b\)](#). However, we note that this comes at the cost of additional hyperparameters –  $v_{\min}$  and  $v_{\max}$ . Although we empirically find that they are not particularly difficult to tune<sup>1</sup>, one may be able to consider incorporating the reward normalization for the distributional critic proposed in SimbaV2 ([Lee et al., 2025](#)) into FastTD3.

**Clipped Double Q-learning (CDQ)** While [Nauman et al. \(2024a\)](#) report that using the average of Q-values rather than the minimum employed in CDQ leads to better performance when combined with layer normalization, our findings indicate a different trend in the absence of layer normalization. Specifically, without layer normalization, CDQ remains a critical design choice and using minimum generally performs better across a range of tasks. This suggests that CDQ continues to be an important hyperparameter that must be tuned per task to achieve optimal reinforcement learning performance.

**Architecture** We use an MLP with a descending hidden layer configuration of 1024, 512, and 256 units for the critic, and 512, 256, and 128 units for the actor. We find that using smaller models

<sup>1</sup>We provide tuned hyperparameters in our open-source implementation.

tends to degrade both time-efficiency and sample-efficiency in our experiments. We also experiments with residual paths and layer normalization (Ba et al., 2016) similar to BRO (Nauman et al., 2024b) or Simba (Lee et al., 2024), but they tend to slow down training without significant gains in our experiments. We hypothesize that this is because the data diversity afforded by parallel simulation and large-batch training reduces the effective off-policyness of updates, thereby mitigating instability often associated with the deadly triad of bootstrapping, function approximation, and off-policy learning (Sutton & Barto, 2018). As a result, the training process remains stable even without additional architectural stabilizers like residual connections or layer normalization.

**Exploration noise schedules** In contrast to PQL (Li et al., 2023b) which found the effectiveness of mixed noise – using different Gaussian noise scales for each environment sampled from  $[\sigma_{\min}, \sigma_{\max}]$ , we find no significant gains from the mixed noise scheme. Nonetheless, we used mixed noise schedule, as it allows for flexible noise scheduling with only a few lines of additional code. But we find that using large  $\sigma_{\max} = 0.4$  is helpful for FastTD3 as shown in Li et al. (2023b).

**Update-to-data ratio** In contrast to prior work showing that increasing the update-to-data (UTD) ratio – that is, the number of gradient updates per environment step – typically requires additional techniques (D’Oro et al., 2023; Schwarzer et al., 2023) or architectural changes (Nauman et al., 2024b; Lee et al., 2024, 2025), we find that FastTD3 does not require such modifications. Using a standard 3-layer MLP without normalization, FastTD3 scales favorably with higher UTDs in terms of sample efficiency. In particular, we find sample-efficiency tends to improve with higher UTDs, but at the cost of increased wall-time for training. We hypothesize that this is because FastTD3 operates at extremely low UTDs – typically 2, 4, 8 updates per 128 to 4096 (parallel) environment steps – reducing the risk of early overfitting often associated with high UTDs.

**Replay buffer size** Instead of defining a *global* replay buffer size, we set the size as  $N \times \text{num\_envs}$  (see Section 2.2 for more details on replay buffer design). In practice, we find that using a larger  $N$  improves performance, though it comes at the cost of increased GPU memory usage, as we store entire buffer on the GPU.

## 2.2 Implementation Details

**Parallel environments** For IsaacLab and MuJoCo Playground, we use their native support for parallel simulation. However, for HumanoidBench that does not support GPU-based parallelization, we use SubprocVecEnv from Stable Baselines3 library (Raffin et al., 2021). We find that HumanoidBench’s default configuration launches a GPU-based renderer for each simulation, which makes it difficult to run more than 100 environments. We have submitted a pull request to HumanoidBench that adds support for disabling the default GPU-based renderer, which is merged into the main branch.

**Environment wrappers** To build an easy-to-use codebase that supports different suites that assume different configurations, we built or used wrappers for each suite.

- For MuJoCo Playground, we use the native RSLRLBraxWrapper that converts Jax tensors to Torch tensors and follows the API of RSL-RL (Rudin et al., 2022). Because this wrapper does not support saving *final* observations before resetting each environment, we implemented this in a separate fork<sup>2</sup>. We will get these changes to be merged into the main repository.
- As IsaacLab natively supports PyTorch, we implemented a simple wrapper that conforms to the RSL-RL API. Currently, our implementation does not support rendering during IsaacLab training, as IsaacLab does not allow for multiple simulations to run concurrently.
- For HumanoidBench, we developed a wrapper that follows the RSL-RL API and converts NumPy arrays to PyTorch tensors.

**Asymmetric actor-critic** For IsaacLab and MuJoCo Playground, which often provide privileged states for the critic network, we implement support for the asymmetric actor-critic (Pinto et al., 2017).

**AMP and torch.compile** While JAX-based RL implementations have become popular in recent days for its speed, we build our implementation upon PyTorch (Paszke et al., 2019) for its simplicity and flexibility. We find that mixed-precision training with AMP and bfloat16 accelerates training

---

<sup>2</sup>See [https://github.com/younggyoseo/mujoco\\_playground](https://github.com/younggyoseo/mujoco_playground).

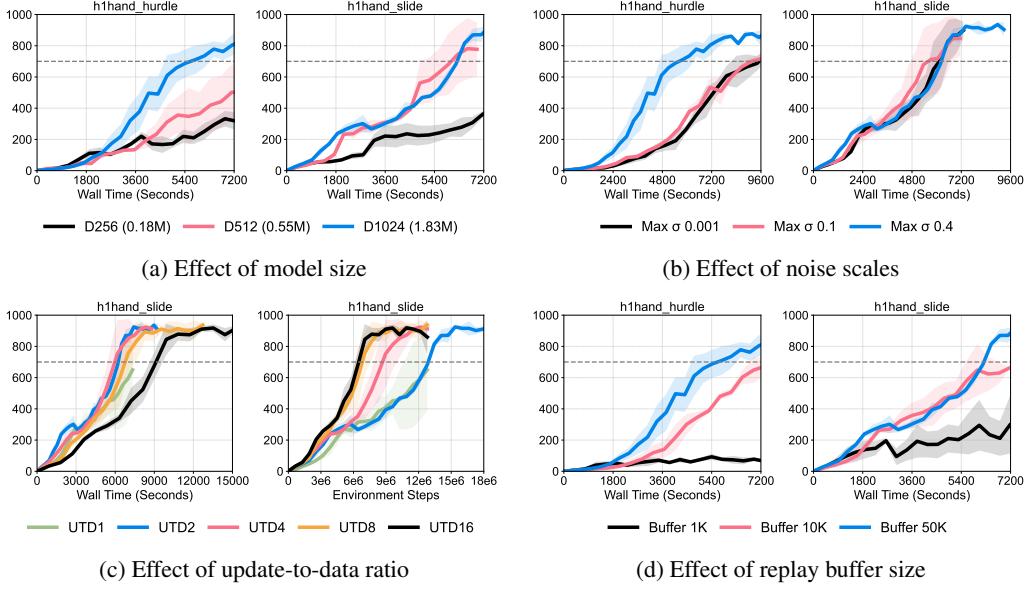


Figure 6: **Effect of design choices (2 / 2).** We investigate the effect of (a) model size, (b) noise scales, (c) update-to-data ratio, and (d) replay buffer size. The solid line and shaded regions represent the mean and standard deviation across three runs.

by up to 40% in our setup using a single A100 GPU, with no observed instability. We also support `torch.compile` by building our codebase on LeanRL (Huang et al., 2022)<sup>3</sup>, which provides up to a 35% speedup in our experiments. When using both AMP and `torch.compile`, we observe a combined training speedup of up to 70%.

**Replay buffer** Instead of defining a *global* replay buffer size, we set the size as  $N \times \text{num\_envs}$ . We find that it better decouples the effects of replay buffer size from the number of parallel environments. For instance, if the global buffer size is fixed at one million, the task episode length is 1000, and the user increases the number of environments from 1000 to 2000, the replay buffer will only be able to save half of each trajectory from each environment (it will start discarding early samples after 500 timesteps), which may negatively affect performance. On the other hand, if we specify  $N = 1000$  for each environment, the buffer can save the whole trajectory without being affected by the number of parallel environments. Because we focus on non-vision domains, we store the entire buffer on the GPU to avoid the overhead of data transfer between CPU and GPU.

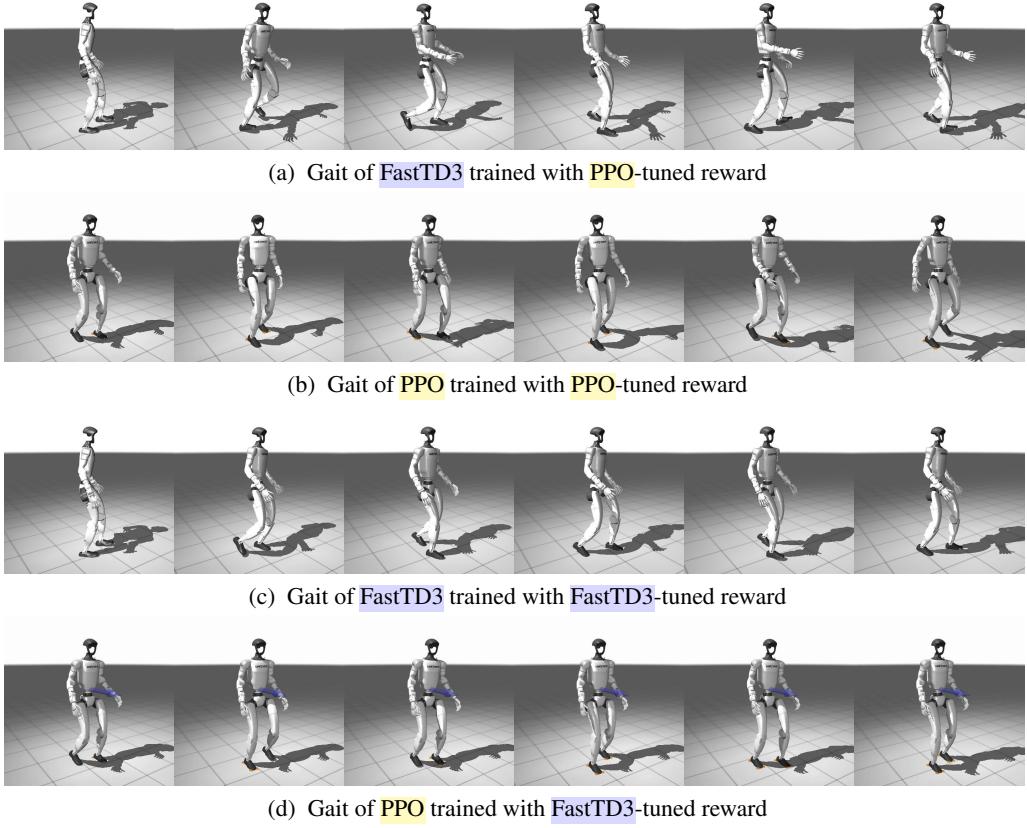
### 3 Experiments

**Setups** For the DreamerV3 (Hafner et al., 2023), SAC (Haarnoja et al., 2018), and TDMPC2 (Hansen et al., 2024) baselines, we use learning curves from three runs available in the Humanoid-Bench repository. Given that each run is trained for 48 hours, we use interpolated wall-clock timestamps to plot the curves. For SimbaV2 (Lee et al., 2025), we use the official codebase to conduct experiments on tasks involving dexterous hands<sup>4</sup>. We report single-seed results for SimbaV2 but plan to include additional runs in the future. All FastTD3 results are aggregated over three runs. Experiments are conducted on a cloud instance with a single NVIDIA A100 GPU and 16 CPU cores.

**Results** We provide the aggregate results over all tasks for each suite in Figure 3, individual results on a selected set of tasks in Figure 4, and individual results on full set of tasks in Appendix A. We provide extensive experimental results that investigate the effect of various design choices (described in Section 2.1) in Figure 5 and Figure 6.

<sup>3</sup><https://github.com/pytorch-labs/LeanRL>

<sup>4</sup>SimbaV2 uses action repeat of 2 for HumanoidBench in their paper, which is unusual for joint position control. We thus run our SimbaV2 experiments with action repeat of 1.



**Figure 7: Different RL algorithms may need different reward functions.** We find that FastTD3 trained with the PPO-tuned reward function (a) results in an undeployable gait, as the G1 robot exhibits abrupt arm movements. (b) This is notable because the same (PPO-tuned) reward function successfully produces a natural walking gait when used with PPO. (c) By tuning the reward function specifically for FastTD3 – adding stronger penalty terms – we were able to train a smoother gait with FastTD3. (d) On the other hand, training PPO with FastTD3-tuned reward is slow to train and results in a slowly-walking gait because of stronger penalty terms. For clarity, we use PPO-tuned reward for all experiments in other experiments except this analysis and sim-to-real experimental results.

**Different RL algorithms may need different reward functions** While training humanoid locomotion policies in MuJoCo Playground, we observed that PPO and FastTD3 produced notably different gaits, despite being trained with the same reward function (see Figure 7a and Figure 7b). We hypothesize that this is because existing reward functions are typically tuned for PPO, and different algorithms may require different reward structures to produce desirable behaviors. To address this, we tuned the reward function specifically for FastTD3 – with stronger penalty terms. This process was efficient due to FastTD3’s short training time. As shown in Figure 7c, the tuned reward enabled FastTD3 to learn a stable and visually appealing gait compared to the one in Figure 7a. On the other hand, we observe that training PPO with FastTD3-tuned reward results also results in undeployable gait that walks too slowly (see Figure 7d). This observation suggests that standard metric – episode return – may not capture the practical usefulness of learned policies.

**FastSAC Experiments** To investigate whether our recipe generalizes to other model-free RL algorithms, we develop FastSAC, which incorporates our FastTD3 recipe into SAC (Haarnoja et al., 2018). We find that FastSAC trains significantly faster than vanilla SAC (see Figure 8). However, we also observe that FastSAC tends to be unstable during training, which we hypothesize is due to the difficulty of maximizing action entropy in high-dimensional action spaces. Given that SimbaV2 (Lee et al., 2025) is notably faster than vanilla SAC in our main experiments, incorporating such recent advancements in off-policy RL into FastTD3 or FastSAC may be a promising future direction.

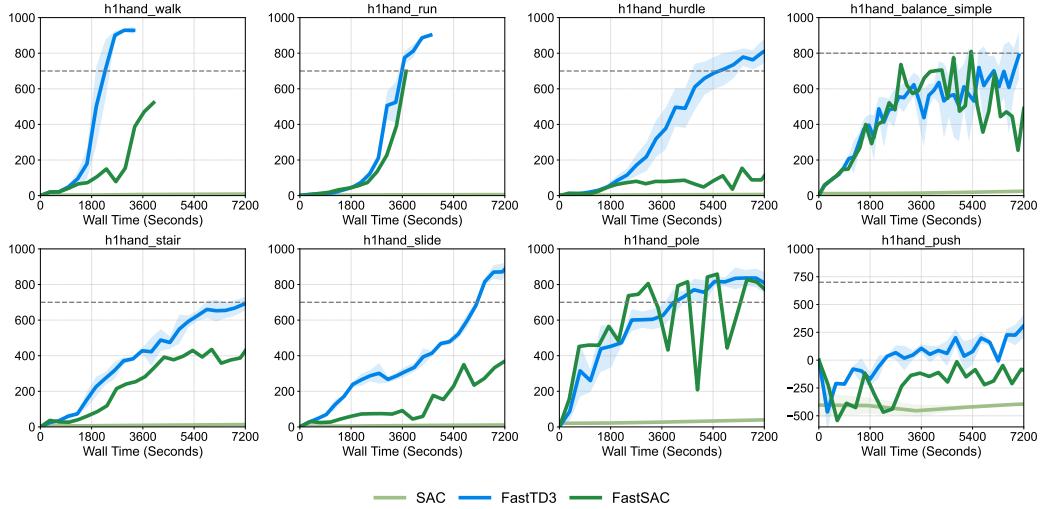


Figure 8: **FastSAC.** We develop FastSAC, a variant of SAC that incorporates our FastTD3 recipe. We find that FastSAC is significantly faster than vanilla SAC, though still slower than FastTD3.

**Sim-to-real RL with FastTD3** For the sim-to-real experiments in Figure 1, we use Booster Gym<sup>5</sup>, which supports 12-DOF control for a Booster T1 humanoid robot with fixed arms, waist, and heading. For convenience, we ported Booster Gym’s robot configuration and reward functions into MuJoCo Playground, which originally only supports 23-DOF T1 control with all joints enabled. We find that training FastTD3 in MuJoCo Playground significantly simplifies and accelerates iteration cycle compared to training with Booster Gym based on IsaacGym (Makoviychuk et al., 2021).

## 4 Discussion

We have presented FastTD3, a simple, fast, and capable RL algorithm that efficiently solves a variety of locomotion and manipulation tasks from HumanoidBench (Sferrazza et al., 2024), IsaacLab (Mittal et al., 2023), and MuJoCo Playground (Zakka et al., 2025). We have demonstrated that a simple algorithm, combined with well-tuned hyperparameters and without introducing new architectures or training techniques, can serve as a surprisingly strong baseline for complex robotics tasks. Along with this report, we provide an open-source implementation of FastTD3 – a lightweight and easy-to-use codebase featuring user-friendly features and pre-configured hyperparameters.

Here, we would like to emphasize the goal of this work is not to claim novelty or superiority over prior algorithms. Our approach builds directly on insights already established in the research community. Parallel Q-Learning (PQL; Li et al. 2023b) demonstrated how off-policy RL can be effectively scaled using massive parallel simulation, followed by Parallel Q-Network (PQN; Gallici et al. 2024) that made a similar observation for discrete control. Similarly, Raffin (2025) and Shukla (2025) showed that SAC can also be scaled successfully through parallel simulation and carefully tuned hyperparameters. The aim of this work is to distill those insights into a simple algorithm, provide extensive experimental analysis of various design choices, and release an easy-to-use implementation.

We are excited about several future directions. Importantly, our work is orthogonal to many recent advances in RL, and these improvements can be readily incorporated into FastTD3 to further advance the state of the art. We expect this integration process to be both straightforward and effective. We also look forward to applications of FastTD3 in real-world RL setups. As an off-policy RL algorithm, FastTD3 is well-suited for demo-driven RL setups for humanoid control (Chernyadev et al., 2024; Seo & Abbeel, 2024), as well as for fine-tuning simulation-trained policies through real-world interactions. Finally, the faster iteration cycles of FastTD3 could be useful in iterative inverse RL setups that leverage language models as reward generators (Ma et al., 2023), offering a promising approach to address the longstanding challenge of reward design in humanoid control.

We hope that our work and implementation help accelerate future RL research in robotics.

<sup>5</sup>[https://github.com/BoosterRobotics/booster\\_gym](https://github.com/BoosterRobotics/booster_gym)

## Acknowledgements

This work is supported in part by Multidisciplinary University Research Initiative (MURI) award by the Army Research Office (ARO) grant No. W911NF-23-1-0277 and the ONR Science of Autonomy Program N000142212121, and ONR MURI N00014-22-1-2773. Pieter Abbeel holds concurrent appointments as a Professor at UC Berkeley and as an Amazon Scholar. This paper describes work performed at UC Berkeley and is not associated with Amazon. We thank NVIDIA for providing compute resources through the NVIDIA Academic DGX Grant.

## References

- Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- Bellemare, Marc G, Dabney, Will, and Munos, Rémi. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, 2017.
- Chernyadev, Nikita, Backshall, Nicholas, Ma, Xiao, Lu, Yunfan, Seo, Younggyo, and James, Stephen. Bigym: A demo-driven mobile bi-manual manipulation benchmark. In *Conference on Robot Learning*, 2024.
- D’Oro, Pierluca, Schwarzer, Max, Nikishin, Evgenii, Bacon, Pierre-Luc, Bellemare, Marc G, and Courville, Aaron. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *International Conference on Learning Representations*, 2023.
- Fujimoto, Scott, Hoof, Herke, and Meger, David. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.
- Fujimoto, Scott, D’Oro, Pierluca, Zhang, Amy, Tian, Yuandong, and Rabbat, Michael. Towards general-purpose model-free reinforcement learning. In *International Conference on Learning Representations*, 2025.
- Gallici, Matteo, Fellows, Mattie, Ellis, Benjamin, Pou, Bartomeu, Masmitja, Ivan, Foerster, Jakob Nicolaus, and Martin, Mario. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.
- Haarnoja, Tuomas, Zhou, Aurick, Hartikainen, Kristian, Tucker, George, Ha, Sehoon, Tan, Jie, Kumar, Vikash, Zhu, Henry, Gupta, Abhishek, Abbeel, Pieter, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.
- Hafner, Danijar, Pasukonis, Jurgis, Ba, Jimmy, and Lillicrap, Timothy. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.
- Hansen, Nicklas, Su, Hao, and Wang, Xiaolong. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024.
- Heess, Nicolas, Tb, Dhruva, Sriram, Srinivasan, Lemmon, Jay, Merel, Josh, Wayne, Greg, Tassa, Yuval, Erez, Tom, Wang, Ziyu, Eslami, SM, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.
- Hester, Todd, Vecerik, Matej, Pietquin, Olivier, Lanctot, Marc, Schaul, Tom, Piot, Bilal, Horgan, Dan, Quan, John, Sendonaris, Andrew, Osband, Ian, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.
- Huang, Shengyi, Dossa, Rousslan Fernand Julien, Ye, Chang, Braga, Jeff, Chakraborty, Dipam, Mehta, Kinal, and Araújo, João G.M. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL <http://jmlr.org/papers/v23/21-1342.html>.
- Hwangbo, Jemin, Lee, Joonho, Dosovitskiy, Alexey, Bellicoso, Dario, Tsounis, Vassilios, Koltun, Vladlen, and Hutter, Marco. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 2019.

- Kaufmann, Elia, Bauersfeld, Leonard, Loquercio, Antonio, Müller, Matthias, Koltun, Vladlen, and Scaramuzza, Davide. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.
- Lee, Hojoon, Hwang, Dongyoong, Kim, Donghu, Kim, Hyunseung, Tai, Jun Jet, Subramanian, Kaushik, Wurman, Peter R, Choo, Jaegul, Stone, Peter, and Seno, Takuma. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. In *International Conference on Learning Representations*, 2024.
- Lee, Hojoon, Lee, Youngdo, Seno, Takuma, Kim, Donghu, Stone, Peter, and Choo, Jaegul. Hyper-spherical normalization for scalable deep reinforcement learning. In *International Conference on Machine Learning*, 2025.
- Li, Tianhong, Chang, Huiwen, Mishra, Shlok Kumar, Zhang, Han, Katabi, Dina, and Krishnan, Dilip. Mage: Masked generative encoder to unify representation learning and image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023a.
- Li, Zechu, Chen, Tao, Hong, Zhang-Wei, Ajay, Anurag, and Agrawal, Pulkit. Parallel  $q$ -learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *International Conference on Machine Learning*, pp. 19440–19459. PMLR, 2023b.
- Lyle, Clare, Zheng, Zeyu, Khetarpal, Khimya, Martens, James, van Hasselt, Hado P, Pascanu, Razvan, and Dabney, Will. Normalization and effective learning rates in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:106440–106473, 2024.
- Ma, Yecheng Jason, Liang, William, Wang, Guanzhi, Huang, De-An, Bastani, Osbert, Jayaraman, Dinesh, Zhu, Yuke, Fan, Linxi, and Anandkumar, Anima. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.
- Makoviychuk, Viktor, Wawrzyniak, Lukasz, Guo, Yunrong, Lu, Michelle, Storey, Kier, Macklin, Miles, Hoeller, David, Rudin, Nikita, Allshire, Arthur, Handa, Ankur, and State, Gavriel. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.
- Mittal, Mayank, Yu, Calvin, Yu, Qinx, Liu, Jingzhou, Rudin, Nikita, Hoeller, David, Yuan, Jia Lin, Singh, Ritvik, Guo, Yunrong, Mazhar, Hammad, Mandlekar, Ajay, Babich, Buck, State, Gavriel, Hutter, Marco, and Garg, Animesh. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.
- Nauman, Michał, Bortkiewicz, Michał, Miłoś, Piotr, Trzcinski, Tomasz, Ostaszewski, Mateusz, and Cygan, Marek. Overestimation, overfitting, and plasticity in actor-critic: the bitter lesson of reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, 2024a. URL <https://arxiv.org/pdf/2403.00514.pdf>. PMLR 235:37342–37364.
- Nauman, Michał, Ostaszewski, Mateusz, Jankowski, Krzysztof, Miłoś, Piotr, and Cygan, Marek. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. In *Advances in Neural Information Processing Systems*, 2024b.
- Paszke, Adam, Gross, Sam, Massa, Francisco, Lerer, Adam, Bradbury, James, Chanan, Gregory, Killeen, Trevor, Lin, Zeming, Gimelshein, Natalia, Antiga, Luca, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- Pinto, Lerrel, Andrychowicz, Marcin, Welinder, Peter, Zaremba, Wojciech, and Abbeel, Pieter. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.
- Raffin, Antonin. Getting sac to work on a massive parallel simulator: An rl journey with off-policy algorithms. *araffin.github.io*, Feb 2025. URL <https://araffin.github.io/post/sac-massive-sim/>.
- Raffin, Antonin, Hill, Ashley, Gleave, Adam, Kanervisto, Anssi, Ernestus, Maximilian, and Dormann, Noah. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL <http://jmlr.org/papers/v22/20-1364.html>.

- Rudin, Nikita, Hoeller, David, Reist, Philipp, and Hutter, Marco. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 91–100. PMLR, 2022. URL <https://proceedings.mlr.press/v164/rudin22a.html>.
- Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Schwarzer, Max, Ceron, Johan Samir Obando, Courville, Aaron, Bellemare, Marc G, Agarwal, Rishabh, and Castro, Pablo Samuel. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, 2023.
- Seo, Younggyo and Abbeel, Pieter. Reinforcement learning with action sequence for data-efficient robot learning. *arXiv preprint arXiv:2411.12155*, 2024.
- Sferrazza, Carmelo, Huang, Dun-Ming, Lin, Xingyu, Lee, Youngwoon, and Abbeel, Pieter. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation. In *Robotics: Science and Systems*, 2024.
- Shukla, Arth. Speeding up sac with massively parallel simulation. <https://arthshukla.substack.com>, Mar 2025. URL <https://arthshukla.substack.com/p-speeding-up-sac-with-massively-parallel>.
- Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014.
- Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press, 2018.
- Voelcker, Claas A, Hussing, Marcel, Eaton, Eric, Farahmand, Amir-massoud, and Gilitschenski, Igor. Mad-td: Model-augmented data stabilizes high update ratio rl. In *International Conference on Learning Representations*, 2025.
- Zakka, Kevin, Tabanpour, Baruch, Liao, Qiayuan, Haiderbhai, Mustafa, Holt, Samuel, Luo, Jing Yuan, Allshire, Arthur, Frey, Erik, Sreenath, Koushil, Kahrs, Lueder A, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.
- Zhuang, Zifeng, Shi, Diyuan, Suo, Runze, He, Xiao, Zhang, Hongyin, Wang, Ting, Lyu, Shangke, and Wang, Donglin. Tdmpbc: Self-imitative reinforcement learning for humanoid robot control. *arXiv preprint arXiv:2502.17322*, 2025.

## A Additional Results

### A.1 HumanoidBench

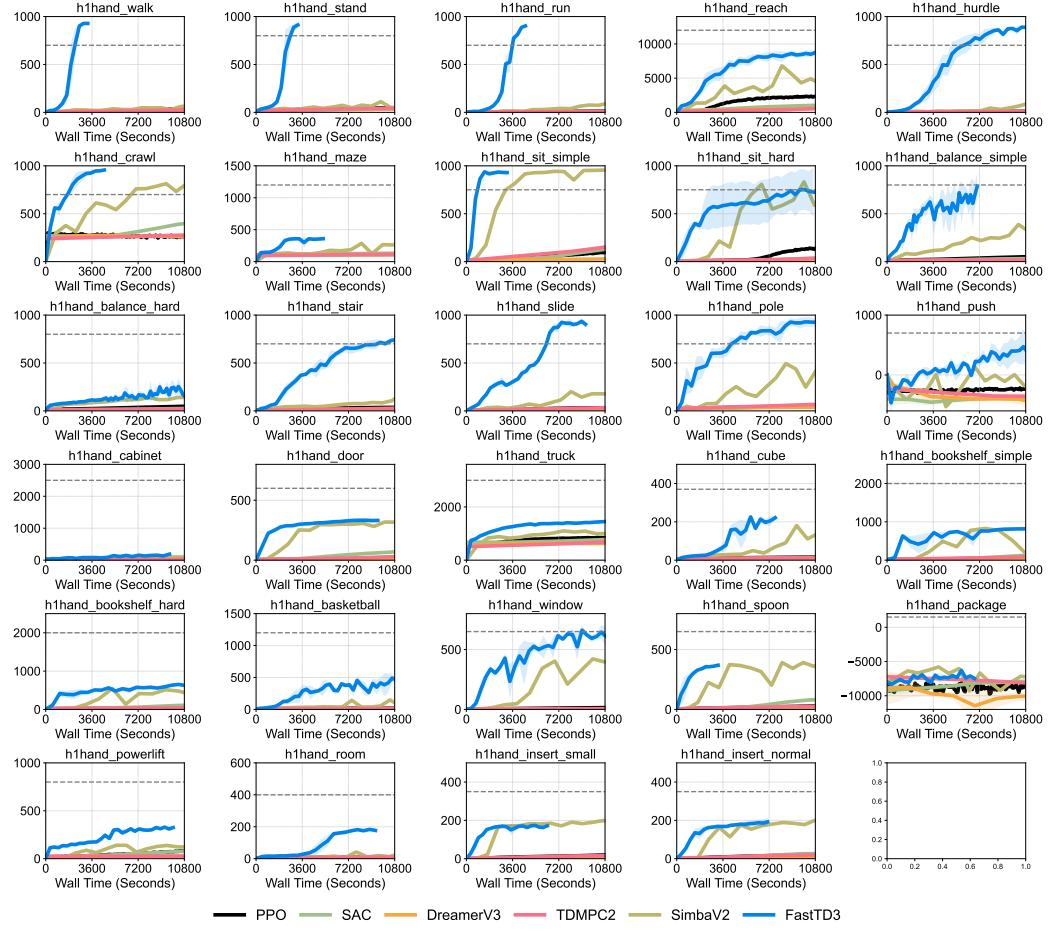


Figure 9: **HumanoidBench results.** We provide learning curves on a 39 tasks from HumanoidBench (Sferrazza et al., 2024) The solid line and shaded regions represent the mean and standard deviation across three runs.

## A.2 IsaacLab

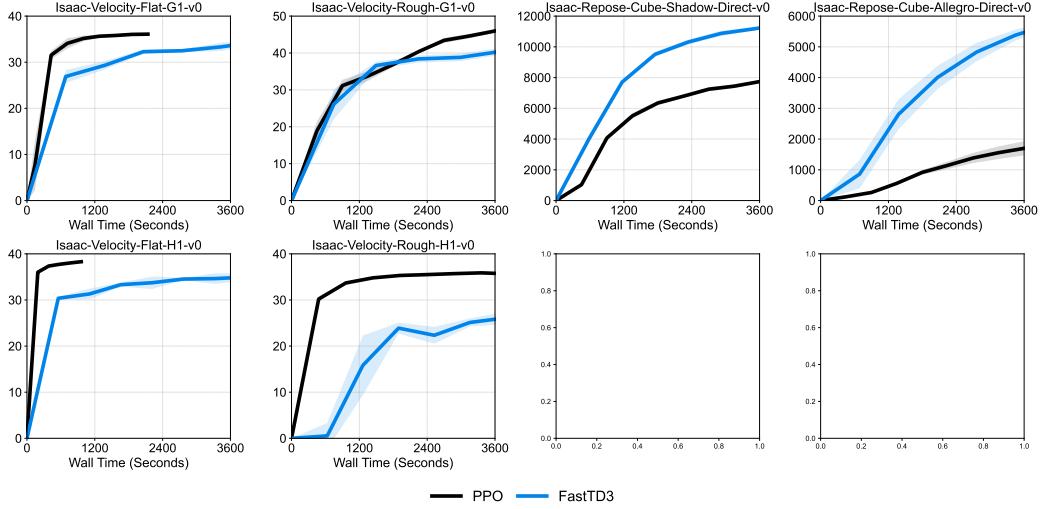


Figure 10: **IsaacLab results.** We provide learning curves on six tasks from IsaacLab (Mittal et al., 2023). The solid line and shaded regions represent the mean and standard deviation across three runs.

## A.3 MuJoCo Playground

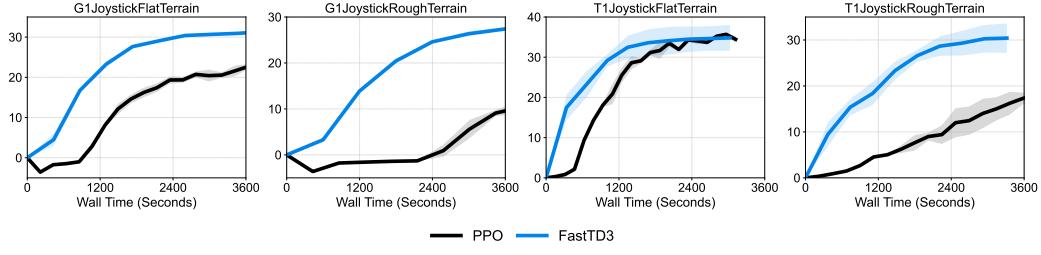


Figure 11: **MuJoCo Playground results.** We provide learning curves on four tasks from MuJoCo Playground (Zakka et al., 2025). The solid line and shaded regions represent the mean and standard deviation across three runs.

## B Sample Efficiency Curves

### B.1 HumanoidBench

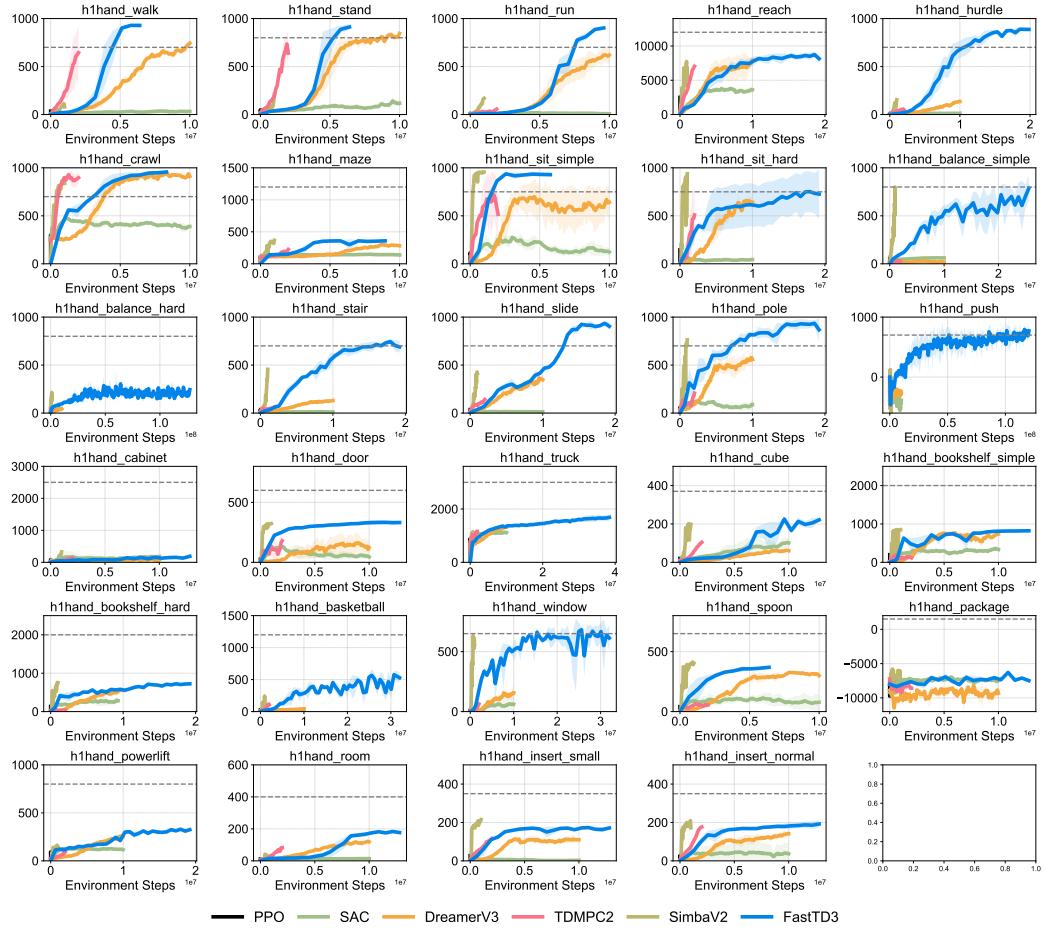


Figure 12: **HumanoidBench results (sample-efficiency).** We provide learning curves on a 39 tasks from HumanoidBench (Sferrazza et al., 2024) The solid line and shaded regions represent the mean and standard deviation across three runs.

## B.2 IsaacLab

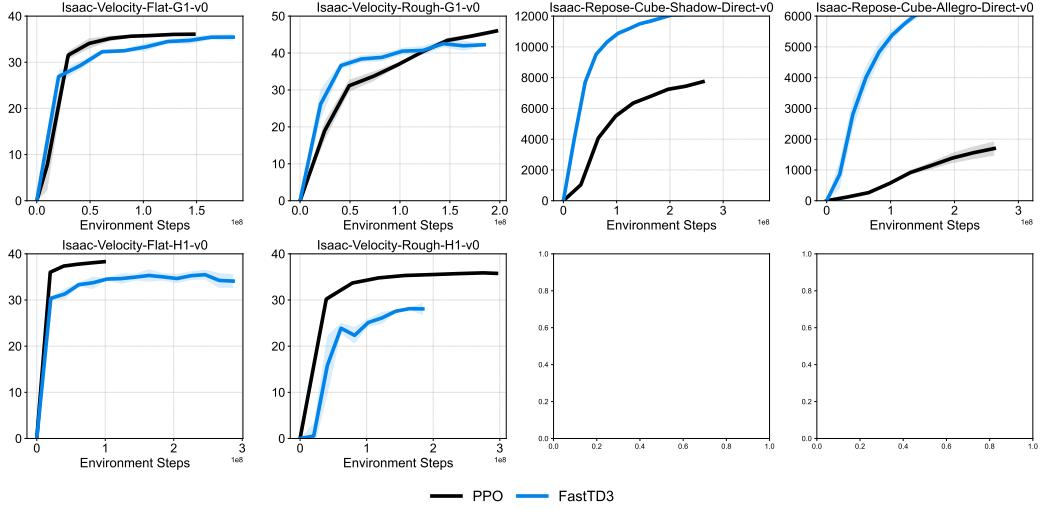


Figure 13: **IsaacLab results (sample-efficiency).** We provide learning curves on six tasks from IsaacLab (Mittal et al., 2023). The solid line and shaded regions represent the mean and standard deviation across three runs.

## B.3 MuJoCo Playground

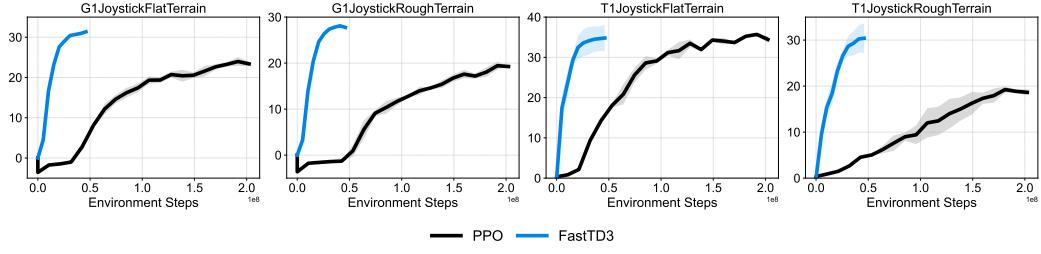


Figure 14: **MuJoCo Playground results (sample-efficiency).** We provide learning curves on four tasks from MuJoCo Playground (Zakka et al., 2025). The solid line and shaded regions represent the mean and standard deviation across three runs.