∇Mimic: Ultra-Efficient Humanoid Control with Differentiable Simulation

Weyl Lu Yubei Chen
UC Davis*

Abstract

Robot policy learning is dominated by zeroth-order methods such as PPO, which treat the simulator as a black box and pay for it with enormous sample counts. First-order policy learning differentiates through the dynamics, but on contact-rich whole-body control it has rarely worked: its gradients explode. A recent world-model method showed that derivative information can make humanoid motion tracking far more efficient, using surrogate gradients of a learned model. We show that exact gradients of the simulator itself do better, once two obstacles are removed. Exact gradients contain rare unbounded samples, because contact dynamics is not differentiable everywhere and its derivative diverges near some events; and they are expensive, because the simulator's adjoint has to replay the contact solve.

∇Mimic bounds these samples with median-relative per-sample gradient clipping at every step of the backward pass, and reuses each costly simulated window by multistep regression onto gradient-derived action targets, which bounds every action target independently of the gradient scale. **NablaEngine**, a differentiable GPU simulation engine for the Unitree G1, removes the cost: it computes MuJoCo's full contact dynamics with fused per-world kernels and the adjoint of a whole control step as a single megakernel, with analytic gradients that match float64 finite differences, 11.5× faster than the available MuJoCo Warp adjoint.

On whole-body tracking of all 40 LAFAN1 clips with one policy on one GPU, ∇Mimic reaches the score reported by the state-of-the-art world-model method, 0.82, in 8 minutes, where that method reports reaching it within 1.2 h, and levels off at about 0.91, above the 0.866 that method reports at 3 h. SHAC, the standard first-order baseline, stops on exploding gradients within minutes; with its guard relaxed it needs 2.6 h to reach the same score. We release NablaEngine and ∇Mimic as open-source infrastructure for first-order humanoid policy learning.

1 Introduction

Motion tracking has become a central task in humanoid learning. A tracking policy imitates reference motions in physics, so any human motion that can be retargeted to a robot becomes a behavior it can execute. The idea goes back to DeepMimic and its successors in character animation [42, 59, 72], and has moved to real humanoids [25, 27, 31, 79, 88]. Two developments made it a foundation rather than a single skill. Large motion datasets of better quality, from AMASS and LAFAN1 to BONES-SEED, together with robot retargets of them, now cover hours of diverse motion [7, 24, 44, 47]. And a single general tracker trained on such data can serve as the low-level policy of larger systems: its encoder–decoder structure provides a motion interface for teleoperation, generation and downstream tasks [9, 41, 43].

Almost all of these trackers are trained with zeroth-order policy gradient methods, chiefly PPO [64], which need only rewards from a black-box simulator and pay for it with enormous sample counts. Massively parallel GPU simulation [21, 22, 48] made the samples cheap, yet a whole-body tracking policy still takes hours to days of GPU time, and this cost bounds how far the approach can scale.

First-order policy learning differentiates the return through the dynamics, so every simulated step contributes an update direction instead of a scalar reward [12, 15, 71, 82]. On contact-rich whole-body control this promise has rarely held: gradients through stiff contact dynamics explode [50, 71]. SHAC [82], the standard first-order method, is no exception. On whole-body tracking with the Unitree G1, every SHAC run hits exploding gradients and stops within minutes (Section 4.1).

Policy learning through the derivatives of a learned dynamics model has a long history, from stochastic value gradients and model-augmented actor-critic to Dreamer and to tracking controllers for physics-based characters

*Project page: https://sailab.ai/nablamimic/. Code: https://github.com/OamIcs/NablaMimic (∇Mimic) and https://github.com/OamIcs/NablaEngine (NablaEngine).

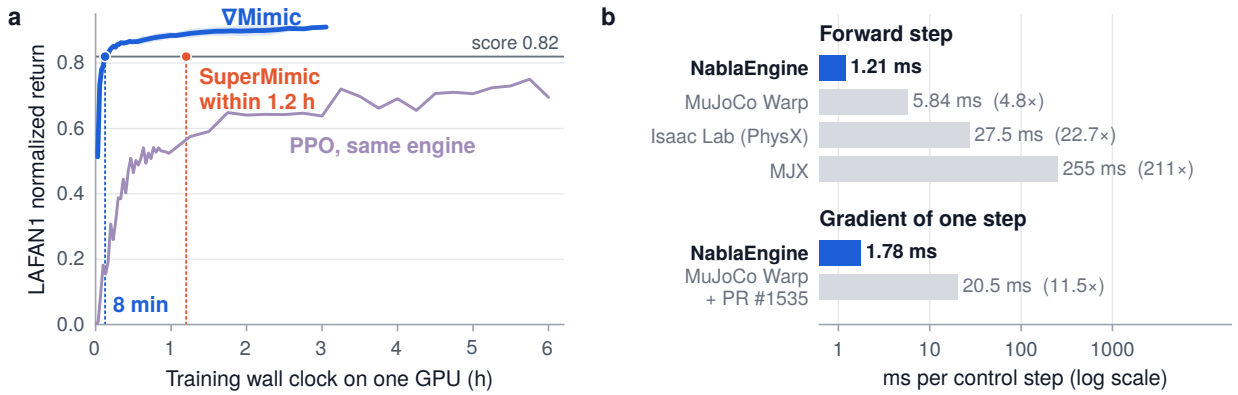


Figure 1. ∇Mimic trains a high-quality tracking policy in minutes. (a) LAFAN1 (all 40 clips): normalized return against training wall clock (linear) on one RTX PRO 6000, mean ± sd over 3 seeds. ∇Mimic reaches 0.82, the score of SuperMimic, in 8 min; SuperMimic reports reaching it within 1.2 h on one H100 [40, §4.1]. PPO trained on the same engine stays below it after 6 h. (b) Simulation time per 20 ms control step for 4096 worlds on the same GPU (log scale): forward step, with Isaac Lab on PhysX (its own G1 asset and task defaults) and MJX (the same model), and gradient of one step; multiples relative to NablaEngine.

[10, 16, 23, 28, 38, 83]. SuperMimic [40] recently brought this approach to humanoid whole-body tracking at scale: it differentiates through a learned world model instead of a simulator and trains a tracking policy far faster than PPO. We read SuperMimic's result differently: not as evidence for world models, but as evidence for derivatives. The gain comes from using the dynamics' derivatives at all; the world model is the price SuperMimic pays to make those derivatives bounded and cheap. This raises the question of what that price buys, and whether it has to be paid. A world model supplies *surrogate gradients*: smooth, bounded and cheap, but derivatives of a different system. A simulator supplies *exact gradients*: derivatives of the simulated system itself, but with rare unbounded samples, because contact dynamics is not differentiable everywhere and its derivative diverges near some events, and expensive, because the simulator's adjoint has to replay the contact solve (Section 4).

The question is therefore not whether exact gradients beat surrogate ones as they are, but what it takes to use them. This paper removes both obstacles. **∇Mimic** handles the samples: it bounds the rare extreme gradients per sample and turns the rest into action targets whose step size is fixed in physical units, so that no single sample can take over an update (Section 5). **NablaEngine** handles the cost: a differentiable GPU engine specialized to the G1, with the adjoint of a whole control step as a single megakernel, whose exact gradient costs less than an extra forward step (Section 6).

On all 40 LAFAN1 clips, with one policy and one GPU, ∇Mimic reaches SuperMimic's score of 0.82 in 8 minutes, where SuperMimic reports 1.2 h, and levels off at about 0.91, with no seed falling on any clip from 31 minutes on (Fig. 1). An open-source world-model tracker run in the same physics on the same GPU takes twice as long to reach 0.82 and ends at 0.882. PPO on the same engine stays below 0.82 for 6 h. The engine behind this simulates 18.8 h of robot motion per second of wall clock (Fig. 1b). Motion tracking suits exact gradients, since its reward is dense and points toward the reference almost everywhere; for sparse rewards, a longer gradient window trades reliance on the critic against accumulated extreme gradients (Section 4.2).

Contributions.

1. A paired measurement of exact and world-model gradients along the same trajectories: the two agree on the bulk of every batch and part ways only in its largest samples, where the exact derivative can be unbounded. This locates the failure of prior first-order methods on whole-body control in single samples, not in the gradient as a whole.
2. **∇Mimic**, a first-order policy learning method that is stable on exact contact gradients at the scale of thousands of parallel environments, with each component ablated.
3. **NablaEngine**, an open-source differentiable simulation engine for the Unitree G1 whose analytic gradients match float64 finite differences.
4. Humanoid motion tracking at the level of the world-model state of the art without a world model, on one GPU: the LAFAN1 score of 0.82 in 8 minutes, where SuperMimic reports 1.2 h.

∇Mimic and NablaEngine are open source; the links are on the first page.

2 Related work

Zeroth-order motion tracking. Physics-based motion imitation with RL was established by DeepMimic [59], and universal trackers such as PHC [42] imitate thousands of clips from datasets such as AMASS [47] with one policy. On real humanoids, H2O and OmniH2O [25, 26], ASAP [27], GMT [9] and others [79, 84] track retargeted human motion on the Unitree G1. BeyondMimic [41] tracks agile LAFAN1 motions [24, 44] with one compact PPO formulation and shared hyperparameters, and SONIC [43] scales PPO tracking to 700 h of motion capture and 21k GPU-hours (128 GPUs for 7 days). These trackers use zeroth-order methods, chiefly PPO [64]. We use BeyondMimic's PPO configuration as our zeroth-order baseline.

First-order policy learning in differentiable simulators. Full-horizon BPTT through differentiable simulators [12, 15, 30] suffers from chaotic, exploding gradients [50], and gradients through stiff contact can be biased or high-variance [56, 70, 71]. PODS [51] regresses the policy onto gradient-improved actions. SHAC [82] introduced short windows [78] with a TD(λ) critic; successors adapt the horizon (AHAC; 18), add maximum entropy (SAPO; 80), combine analytic gradients with PPO (GI-PPO, RPO; 68, 92) or decouple rendering from the gradient (D.Va; 85). On legged robots, policies transferred to hardware were trained with smoothed foot contact [6, 65], a simplified surrogate for the backward pass [69], or an exact forward pass with a smoothed contact sensitivity in the backward pass [32]. DiffMimic [62] tracks single clips in Brax without a critic. BCG [3] combines SHAC and an ADD imitation reward [91] on the smoothed-contact simulator of Schwarke et al. with contact-local randomized bundling, and trains G1 trackers for four LAFAN1 clips with 64 environments. These works use 32–1024 environments; we use 4096 BPTT windows per update on full contact dynamics.

Gradients through learned models. SVG, MAAC and Dreamer optimize policies through learned dynamics [10, 23, 28]. PWM [19] and Ma et al. [45] report that learned models can give smoother optimization landscapes than the true dynamics, and Amigo et al. [4] roll out in the simulator and backpropagate through a learned model. For physics-based characters, SuperTrack [16], ControlVAE [83] and Lei et al. [38] train tracking controllers through learned dynamics. SuperMimic [40] trains one world model online with the policy and reports the state of the art on LAFAN1 and BONES-SEED [7]. Their gradients are those of a learned surrogate; ours are of the simulator itself.

Differentiable and GPU simulators. GPU simulation [48, 55, 63, 86, 87] made samples cheap for zeroth-order methods. Among differentiable engines, DiffTaichi and Warp generate adjoint kernels [30, 46], and dflex [81] and Rewarped [80] use penalty contacts. Brax and MJX [15, 21] use JAX autodiff; MJX is forward mode only on our model [53]. Nimble [77] differentiates LCP contact analytically, and Dojo, Simple and Ostrich [29, 35, 37] apply the implicit function theorem at the contact solution. DiffMJX [58] studies gradients of MuJoCo's soft contacts [74, 75]. Analytic rigid-body derivatives [8] and articulated-body adjoints [61] avoid generic autodiff. MuJoCo Warp [22] is not differentiable in any release; an adjoint exists in an open pull request [14]. Newton [73] and Genesis [17] have differentiable modes, which hold the contact normal fixed [54] or leave some collision operations without gradients, respectively. Kernel fusion for RL simulation [20, 66] and the run-to-run non-determinism of GPU simulators [89] have been studied before. NablaEngine gives exact, deterministic adjoints of MuJoCo's dynamics for one robot at high throughput.

Clipping and regression onto action targets. Norm clipping of BPTT gradients goes back to Pascanu et al. [57]. Per-sample clipping is standard in private SGD [1], with median-relative [5] and quantile thresholds [49]; clipping helps under heavy-tailed noise and gradient-dependent smoothness [67, 90] but is biased in general [34]. We apply it to the action cotangents inside BPTT, as a heuristic outside these guarantees. Regressing a policy onto improved actions follows trajectory-optimization teachers [39, 52], MPO [2], AWR [60] and target-based surrogates [36, 76]. We keep PODS's target regression and add what whole-body contact at thousands of parallel environments requires: per-sample median-relative clipping inside BPTT, without which our training diverges or stalls, and a step size fixed in radians by the batch RMS of the clipped gradients (Sections 5.1 and 5.2).

3 Background

Task. Whole-body motion tracking on the Unitree G1 (29 actuated joints), mjlab model [87]. Control runs at 50 Hz with 4 physics substeps of 5 ms. The policy observes a 10-frame history of noisy proprioception (joint positions and velocities, base angular velocity, gravity direction, previous action), the next 10 reference frames, and the orientation of the reference anchor body (torso) relative to the robot's, which needs the IMU but no global position. The reward is BeyondMimic's tracking reward in SuperMimic's setting [40, 41]: exponential kernels on the position, orientation and

Table 1. Exact and surrogate gradients: bias, unbounded derivatives and cost.

	Exact gradient (simulator)	Surrogate gradient (world model)
Bias	no model error: derivative of the simulated system	derivative of a different, learned system
Unbounded derivatives	near non-differentiable events	none (a smooth network)
Cost per update	high (simulator adjoint)	low (one network pass)

velocities of 14 bodies and on the anchor pose, with action-rate and joint-limit penalties. An episode terminates when the root height error exceeds 0.25 m or the tilt error exceeds 0.8. Appendix A gives the full definitions.

First-order policy learning. The return of a window of H steps starting from state s_0 is

$$J(\theta) = \sum_{t<H} \gamma^t \, r(s_t, a_t) + \gamma^H V_\phi(s_H), \qquad a_t = \pi_\theta(o_t), \quad s_{t+1} = f(s_t, a_t). \tag{1}$$

When f is differentiable, $\nabla_\theta J$ follows by backpropagation through time, a pathwise gradient. Unlike the likelihood-ratio gradient of PPO, it uses the derivatives of f, so its quality depends on them (Section 4).

4 Exact and surrogate gradients

A first-order method uses the derivatives of the dynamics, which a zeroth-order method ignores. It is therefore commonly expected to extract more information from each sample, and to be the better choice whenever these derivatives are available. The record says otherwise: methods that use the simulator's own derivatives have failed on whole-body control, while a method that uses the derivatives of a learned imitation of the simulator succeeded. This section explains the paradox. The derivatives of contact-rich dynamics are unusable as they are, the systems that make them well behaved describe something other than the target dynamics, and computing them costs time. We discuss these trade-offs and the challenges they pose; the rest of the paper addresses them.

Two sources of gradient. To discuss these trade-offs we introduce two terms (a surrogate gradient here is the gradient of a surrogate dynamics, not of a surrogate loss). We call a policy gradient *exact* when it contains the analytic derivatives of the system the policy is trained and evaluated in: here, when the dynamics f in Eq. (1) is the simulator itself. We call it *surrogate* when f is replaced by another, more tractable system, so that the gradient contains that system's derivatives. The surrogate can be a learned dynamics model $\hat{f}$, as in model-based policy optimization [10, 16, 23, 28, 40], or a simplified differentiable simulator that softens contact, for example by penalty springs of limited stiffness in place of rigid contact [15, 81, 82]. The two are the ends of an axis rather than a dichotomy: a simulator with softened contact lies between them, and at the two geometric events where the derivative of the step does not exist, NablaEngine's training gradient holds the undefined term fixed; there is no exact value to depart from there, and everywhere else the two are bit for bit the same (Section 6). Below we compare the two ends: the exact gradients of NablaEngine and the surrogate gradients of a learned world model, the surrogate SuperMimic showed to work for humanoid tracking.

Neither exactness nor boundedness is what learning needs. What it needs is *learning signal*: the part of a gradient that improves the return at the scale of one policy update. Two trade-offs decide how much of it each source delivers per unit of compute (Table 1).

Unbounded exact, bounded surrogate. An exact gradient has no bias, but contact-rich dynamics is not differentiable everywhere. Where contacts appear or vanish, actuators saturate or the solver's active set changes, the derivative jumps or is undefined. Near some of these events it is unbounded. Computed faithfully, it can be infinite. We measured this on the G1 (Appendix B):

- when the feet nearly touch, the derivative of the self-contact normal grows as the inverse of their distance; on training windows the gradient grew by up to 4.9×10^{21} through a window, and 32 rows were not finite;
- for nearly parallel capsules (crossed feet), one control step amplified the gradient by up to 3.6×10^{8}, and repeated steps overflowed to non-finite values in 40 of 172 such states.

A surrogate gradient, in contrast, is bounded everywhere. For a world model that is a multilayer perceptron with weight matrices $W_1, \ldots, W_n$ and an activation with Lipschitz constant c (SiLU, as in SuperMimic, has $c \approx 1.1$), the Jacobian of one predicted step satisfies

$$\left\| \frac{\partial \hat{f}}{\partial (s, a)} \right\|_2 \le L = c^{\,n-1} \prod_{l=1}^{n} \|W_l\|_2 \tag{2}$$

at every state, and the action gradient of an H-step window is bounded by a constant that depends only on L, H and the reward. The bound can be loose and grows with H, but it is uniform: no state makes it infinite. A simulator's step Jacobian has no such bound. The surrogate gradient, however, describes a different system than the one being simulated, so it is biased.

Same bulk, different tail. The two gradients agree on the bulk of the batch and part ways only at the extremes. Each on its own rollouts, they have similar tails. With the policy of a SuperMimic reproduction, the same objective and 32-step windows, the 99.9th percentile of the per-sample action-gradient norm is 25–37 times the median on both routes, and the maximum a few hundred to about 1,350 times, with no consistent ordering between them. A paired comparison along the same trajectory shows where the two differ. We ran the same policy in NablaEngine and, at every recorded state and action, computed the action gradient once with the engine's Jacobian and once with the world model's Jacobian at that state, everything else identical. The two agree closely on most samples (Spearman correlation of the norms 0.94–0.95, median cosine 0.90–0.91, at two checkpoints) and almost perfectly on small gradients (cosine 0.96–0.98 for the smallest 10%). Agreement falls as the exact gradient grows: for the largest 0.1% of exact gradients, the world model's gradient has only 0.35–0.43 of the norm (median), a median cosine of 0.39–0.67, and in some samples points the opposite way. The world model does not create large gradients of its own: where its gradient is largest, the exact one is large as well.

The difference is therefore one of tails, not of bias. Along the same trajectory the world model damps the exact tail (99.9th percentile 16–21 against 24–31 times the median). And with exact derivatives the unbounded events reappear: without NablaEngine's rules for them, single samples reach 6.5×10^{24} times the median and some are not finite. Such samples are rare, but one of them dominates a batch-mean gradient. A few samples carry huge or infinite values and no learning signal; the rest of the batch is as good as the surrogate's, and unbiased.

More exactness is not more signal. Making the derivative of force-saturated actuators exact changed about 20% of per-state gradients but did not improve learning: -0.023 after 1 h (3 seeds, within two seed standard deviations), and no difference together with the self-contact rule below or after 3 h. Training with the plain exact derivative everywhere, without either of NablaEngine's rules and with MuJoCo Warp's actuator convention, gives the same result as the rules: 0.867 against 0.869 after 1.5 h, the same time to 0.82 and the same update rate (residual recipe, 3 seeds).

Bias versus cost. A world model is cheap to evaluate and to differentiate: one network pass, independent of contact complexity. SuperMimic reports that its world model runs more than 10× faster than MuJoCo Warp at the batch sizes used in practice and 62× faster at 64k parallel environments, and that its training needs over 1000× fewer GPU-hours than SONIC's cluster-scale PPO pipeline [40]. The simulators that robot learning commonly uses are slower still on this robot. One control step of 4096 G1 worlds takes 5.8 ms in MuJoCo Warp, 12.9 ms in Isaac Lab with Newton and 27.5–54.1 ms with PhysX, 34.7 ms in MuJoCo C on a 16-thread CPU and 255 ms in MJX, all forward only (one RTX PRO 6000; Table 5). Few of them provide a gradient of the full contact dynamics at all: PhysX does not, MuJoCo C offers finite differences, MJX only forward-mode derivatives on this model, and MuJoCo Warp only through an open pull request.

Per policy update, one SuperMimic iteration takes about 0.1 s on an H100 (estimated from its paper, Appendix E). The same first-order recipe as ours on the MuJoCo Warp adjoint takes 1.63 s per update, 16× more. Even on NablaEngine an update takes 0.37 s, still more than the world model. Exact gradients make up for it in the number of updates: ∇Mimic reaches 0.82 after 3.2k updates, about 1/13 of the roughly 41k iterations SuperMimic needs (estimated from its paper; Appendix E gives the estimate and an alternative reading). This is how we read SuperMimic's result. Per update, training on exact gradients needs far fewer: 3.2k updates against about 41k. Counted in transitions, ∇Mimic reaches 0.82 after 8.8×10^8 simulated transitions, while SuperMimic's iterations of 4096×32 imagined steps amount to 5.4×10^9, or 2.1×10^9 under the alternative reading: a factor of 2.5 to 6 fewer, each of ours simulated rather than imagined. Per second, at the cost of the simulators available at the time, the surrogate made faster progress. The world model won on price.

What exact gradients need. Both trade-offs point at the same two requirements, and at how to meet them differently from a world model. A surrogate pays one bias, on every sample, to buy two things at once: bounded derivatives and a cheap update. We buy them separately. Boundedness costs a bias confined to the norms of the samples that are cut, with their directions and the rest of the batch left exact; this is the job of ∇Mimic (Section 5). Cheapness costs engineering, not bias: the adjoint has to come down to where the higher learning signal per update pays off in wall clock; this is the job of NablaEngine (Section 6). Applied to the world-model gradients of a public SuperMimic reproduction, ∇Mimic's components did not give a lasting gain: with per-sample clipping the return after 2.7 h was 0.80 against 0.79 without it, and with clipping and action targets together training collapsed after 1.5 h (one seed each).

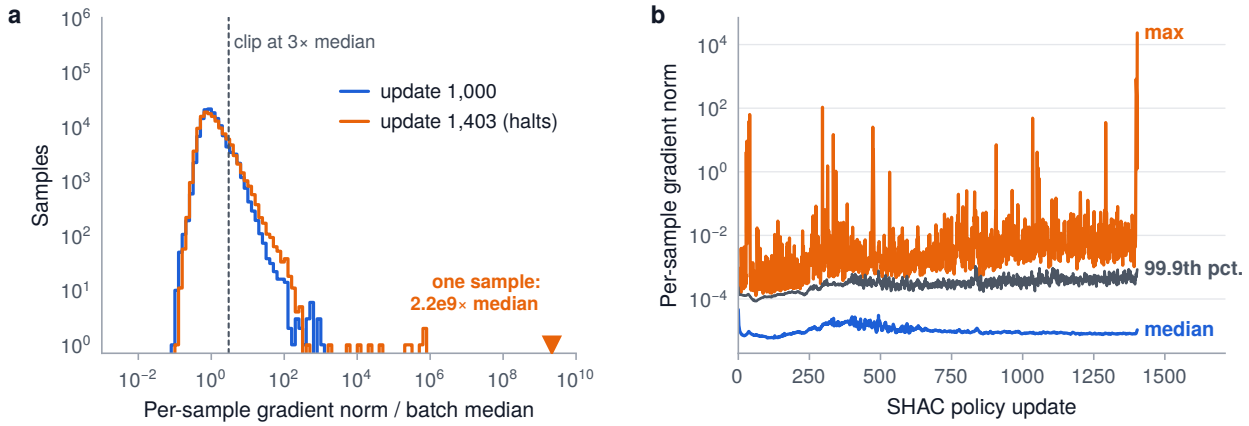


Figure 2. Exploding gradients sit in single samples. Per-sample action-gradient norms of SHAC on exact gradients (one seed, raw, before any clipping). (a) Distribution within one update batch of 131,072 samples, relative to the batch median: an ordinary update and the update in which SHAC's guard stops training, where one sample lies at 2.2×10^9 times the median. (b) Median, 99.9th percentile and maximum per update: the bulk stays flat, only the maximum explodes.

4.1 Prior first-order methods explode

SHAC [82] uses exact gradients the way surrogate-gradient methods use theirs: one global gradient-norm clip and a guard that stops training when the norm exceeds 10^6. On whole-body G1 tracking at our scale, every SHAC run hits exploding gradients and trips this guard: all 3 seeds on NablaEngine within 0.06 h (746–1568 updates), at a return of 0.37 ± 0.09, and on MuJoCo Warp within 0.36 h. The explosion comes from a single sample, not from the batch, and this is why a global norm clip cannot save it: one sample at 10^9 times the median either trips the guard or, if clipped globally, scales every other sample's gradient to nothing. Up to the halt, the median, 99th and 99.9th percentiles of the per-sample action gradient norms stay flat; in the halting update the largest per-sample gradient is 2.2×10^9 to 1.8×10^{10} times the median in each of three instrumented runs, and this one sample trips the guard. When it appears is random: the instrumented runs halted after 326 to 8361 updates. Figure 2 shows this for one seed.

4.2 When first-order methods fit

Dense rewards. Exact gradients are most useful when the reward is dense and smooth in the state. Motion tracking is such a task: the reward at every step is a sum of exponential kernels of tracking errors, so its gradient points toward the reference everywhere except at contact events, and those are what per-sample clipping handles.

Horizon trade-off. With sparse rewards and long-range dependencies the picture changes, and the window length H sets a second trade-off. A long window lets the gradient reach a distant reward and relies less on the critic V_ϕ, whose estimate is biased. But the tail of the gradient grows with every step it is backpropagated through (Fig. 3): over 32 steps the 99.9th percentile grows several-fold and, for SHAC's raw gradients, the maximum by eight to nine orders of magnitude. With the same policy, exact and world-model gradients grow alike, so the accumulation is a property of long chains of Jacobians, not of contact alone. A world model does not escape this trade-off; it only starts from a lighter tail.

A short window keeps the gradient clean but leaves more to the critic. Section 5.4 measures this trade-off for tracking. For tasks whose reward lies seconds away, whether differentiability can be exploited depends on this trade-off, and first-order methods may not pay off.

5 ∇Mimic

We design ∇Mimic to learn from exact gradients as they are, with their rare extreme samples and a scale that varies by orders of magnitude (Section 4). It is a windowed first-order actor-critic built around two components: per-sample clipping and multistep regression onto gradient-derived action targets. The two are a pair. The action targets fix the size of every update in radians, and clipping keeps any single sample from spending that budget alone: without clipping, one extreme sample moves its target by about 15 rad and every other sample's by almost nothing; with it, no sample moves more than 0.13 rad (Section 5.3). Each update simulates a batch of BPTT training windows of H steps on NablaEngine,

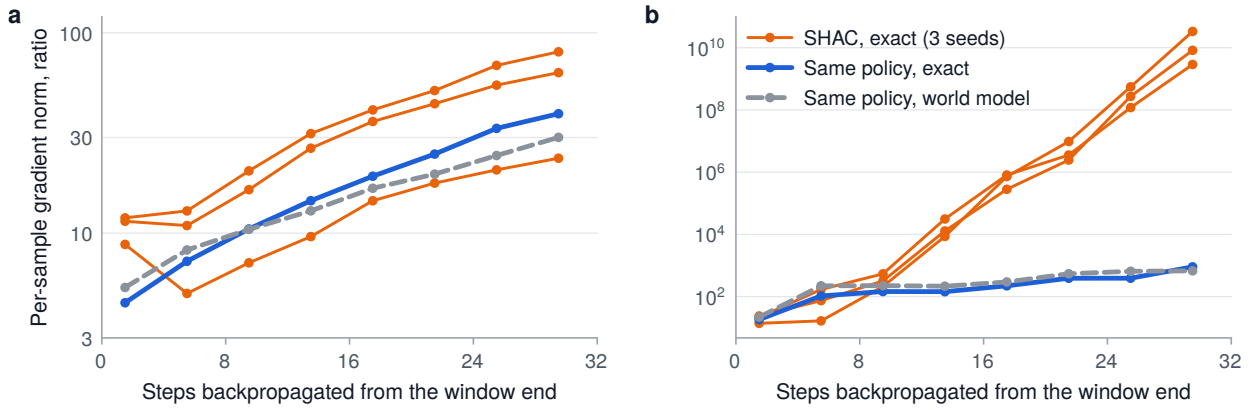


Figure 3. The tail of the per-sample action gradient grows with the number of steps it is backpropagated through, while its median stays flat: (a) 99.9th percentile and (b) maximum, each relative to the median. SHAC: raw exact gradients, 3 seeds, all recorded updates. Same policy: a SuperMimic reproduction's policy, the same objective and 32-step windows, exact (NablaEngine) and world-model gradients, no clipping.

backpropagates the window return (Eq. (1)) through the analytic adjoint, and updates the actor from the resulting action gradients. Algorithm 1 summarizes one update. Each component has its own subsection and ablation (Fig. 4, Table 2).

Inherited components. We keep the data path of SHAC and SuperMimic [40, 82] unchanged, so that our comparison isolates the source and handling of the gradient: window start states come from a replay of states visited by a separate data collector that runs the current policy, the collector's resets are sampled by return-based bins after BeyondMimic [41], and a critic V_ϕ fit with TD(λ) bootstraps the window return. Exact gradients make the replay unnecessary: simply continuing each window from where the previous one ended raises the return by about 0.02 in both recipes (Table 2). We keep the replay in this paper so that the comparison with SHAC and SuperMimic isolates the gradient; the simpler data path is also the better one.

Policy averaging. The policy we evaluate and deploy is an exponential moving average of the actor's weights; training uses the live actor. Its decay grows with the number of averaging steps n,

$$\beta_n = \min\left(0.999,\ \frac{1+n}{10+n}\right), \tag{3}$$

so early in training the average follows the fast-moving actor closely, and later it averages over about a thousand updates.

Fall handling. A fall ends the episode and bootstraps nothing, so the window return loses the rest of the episode; the environment then resets and the rest of the window, a new episode, still counts toward J, with no gradient through the reset. SuperMimic's survival weight instead drops everything after the first episode end in a window [40]. The critic is fit on the state each transition starts from, and a smooth termination penalty starts before the termination thresholds (h on the height error, $\theta_{\max}$ on the tilt), so the gradient sees an approaching fall:

$$r_{\text{pen}} = -W\,\Delta t\left[\left(\frac{\text{relu}(d_z - d_{z,0})}{h - d_{z,0}}\right)^2 + \left(\frac{\text{relu}(d_\theta - d_{\theta,0})}{\theta_{\max} - d_{\theta,0}}\right)^2\right] \tag{4}$$

with $W = 2$, $d_{z,0} = 0.08$, $d_{\theta,0} = 0.10$. Fall handling is not what separates ∇Mimic from the baselines: in the earlier recipe with SuperMimic's masking, removing all of it, the penalty included, changed the return by less than 0.01 at matched wall clock (3 seeds, both action parameterizations).

Effect of averaging and unmasked windows. Together, policy averaging and continuing the window after a reset halve the time to SuperMimic's score of 0.82, from 0.34 h to 0.15 h, and raise the return at 3 h from 0.875 to 0.913 (3 seeds each, both with a 500-update learning-rate warmup). Without averaging, removing the mask alone gives +0.007 at 1.5 h and at 3 h. The component ablations of this section use the earlier recipe (masked, no averaging).

Algorithm 1: One ∇Mimic update.

Input: actor π_θ, critic V_ϕ, FIFO replay $\mathcal{D}$, collector environments, window length H, clip factor κ, step size δ, regression steps M (values in Table 10)

`// Data collection (forward only)`

Step the collector with π_θ and action noise; sample collector resets by return-based bins; push visited states to $\mathcal{D}$

`// BPTT training windows on NablaEngine`

Sample a batch of window start states s_0 from $\mathcal{D}$

for $t = 0, \dots, H-1$ **do**

  $a_t \leftarrow \pi_\theta(o_t)$ with action noise

  $s_{t+1} \leftarrow f(s_t, a_t)$ (forward pass with tape)

  $r_t \leftarrow r(s_t, a_t) + r_{\text{pen}}$ (Eq. (4)); after an episode end, reset the environment and continue the window from the reset state

$J \leftarrow \sum_{t<H} \gamma^t r_t + \gamma^H V_\phi(s_H)$ (Eq. (1))

`// Backward pass with in-backward clipping`

for $t = H-1, \dots, 0$ **do**

  Propagate the cotangent through step t with the analytic adjoint

  Clip each sample's action cotangent to κ times the batch median norm (Eq. (5)) before it propagates further

Clip the window's action gradients $g_t = \partial J / \partial a_t$ again with Eq. (5)

`// Gradient-derived action targets`

$y_t \leftarrow a_t + \delta\, g_t / \mathrm{RMS}(g)$ (Eq. (6))

for $m = 1, \dots, M$ **do**

  Sample a minibatch of pairs (o_t, y_t); take one step of the separate Adam on $\|\pi_\theta(o_t) - y_t\|^2$

Update the averaged policy $\bar{\theta} \leftarrow \beta_n \bar{\theta} + (1-\beta_n)\,\theta$ (Eq. (3))

`// Critic`

Fit V_ϕ with TD(λ) on pre-transition states

5.1 Per-sample median-relative clipping

The extreme exact gradients of Section 4 live in individual samples: a few environments near contact events carry action cotangents orders of magnitude above the rest and dominate a batch-mean gradient. A global norm clip, as in SHAC, rescales the whole batch and destroys the learning signal of every other sample. ∇Mimic instead clips the action cotangent of each sample, the gradient of the window return with respect to the action a_t of one environment at one step, to κ times the batch median norm. This happens at every step of the backward pass, before the cotangent flows into the policy and from there into earlier states, and again on the window's action gradients; the cotangent that flows directly through the dynamics from one state to the previous one is left to the engine's rules for unbounded derivatives (Section 6):

$$g_i \leftarrow g_i \cdot \min\left(1,\ \frac{\kappa \cdot \mathrm{median}_j \|g_j\|}{\|g_i\|}\right), \qquad \kappa = 3. \tag{5}$$

The median is the reference because it is the one statistic the explosion never moves: in Fig. 2b the median and the 99.9th percentile stay flat through the update that halts SHAC, and only the maximum leaves. A median-relative threshold therefore follows the gradient scale over training, needs no absolute value to tune, and never cuts the backward chain. With $\kappa = 3$, clipping rescales 9–21% of the (step, environment) pairs per update; it preserves their direction and changes only their norm. The samples that matter are far fewer: those at 10^4 to 10^6 times the median, the tail of Fig. 2a, one of which is enough to take over a batch-mean update. The tail does not go away as training proceeds: in an instrumented ∇Mimic run, the largest per-sample action gradient is typically 10^4 to 10^5 times the batch median at every stage, and up to 1.9×10^6 times; over about 92,000 updates (3 seeds) no gradient was non-finite. Clipping is not free: the gradients of the clipped samples are biased. It trades this bias for stability.

Without clipping, one of three seeds diverges after 0.54 h and the other two reach 0.562, against 0.855 for the control (Fig. 4a). Clipping is what makes exact gradients usable at all. Its threshold matters much less: with $\kappa = 1$, 10 or 30 every seed trains and ends 0.015 to 0.02 below $\kappa = 3$ after 1.5 h; without clipping all three seeds diverge (residual recipe, 3 seeds each; Table 11). Where it acts matters for speed: clipping only the window's action gradients, not the cotangents inside the backward pass, gives the same return after 1.5 h (0.868 against 0.869) but reaches 0.82 about 11 minutes later

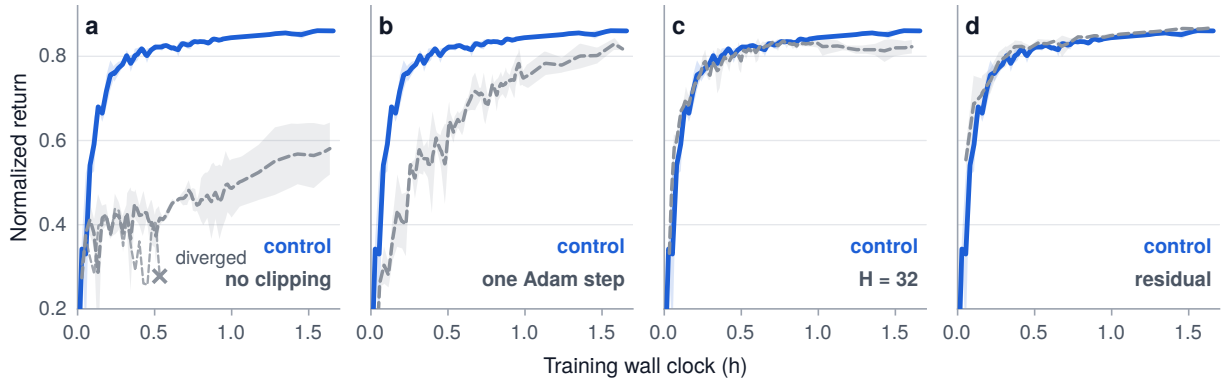


Figure 4. One panel per component of ∇Mimic: the control (blue) against the recipe with that component changed (gray), LAFAN1, mean ± sd over 3 seeds. (a) Per-sample clipping: without it one seed diverges and the others stay far below. (b) Multistep regression: one Adam step per window learns more slowly. (c) Window length: $H = 32$ plateaus lower. (d) Residual actions learn faster and plateau higher than absolute actions.

Table 2. Component ablations on LAFAN1: one change to the recipe at a time, 3 seeds each, against a 3-seed control at matched wall clock (about 1.6 h on one RTX PRO 6000, where the control is still rising and differences are largest; Appendix I gives the full curves). Change in normalized return for the absolute-action recipe (control 0.855), on which the ablations were first run (Fig. 4), and for the residual-action recipe of ∇Mimic (control 0.867). Full tables in Appendix I.

Change	Absolute actions	Residual actions (∇Mimic)
No per-sample clipping	−0.293 (1 of 3 seeds diverged)	all 3 seeds diverged
No multistep regression (one Adam step per window)	−0.049	−0.044
$H = 32$ instead of 64	−0.036	−0.042[a]
Residual instead of absolute actions (a parameterization)	+0.009	—
No replay starts (continued windows)	+0.019	+0.018

[a] Mean of 3 seeds; a fourth seed collapsed and is listed separately in Table 16.

in every seed.

5.2 Multistep regression with gradient-derived action targets

Every exact gradient is expensive: it costs a simulated window and its adjoint. A world-model method can afford to use each imagined window for one gradient step and discard it; we want to extract as much as possible from each simulated window. Taking several gradient steps on the same window, however, would move the policy beyond the region in which that gradient describes the return, and the gradient's scale, which varies by orders of magnitude over training and across the batch, would set the step size. We therefore adopt policy optimization by differentiable simulation (PODS) [51]: the gradient defines targets in action space, and the policy is fit to them by several steps of regression. ∇Mimic turns the clipped action gradient g_t of each window step into a target action

$$y_t = a_t + \delta \cdot \frac{g_t}{\mathrm{RMS}(g)}, \tag{6}$$

so that the RMS setpoint change over the batch is δ, in radians and decaying with the learning-rate schedule, and regresses the policy onto the targets with M minibatch steps under a separate Adam optimizer [33] (settings in Table 10). The multistep regression reuses each simulated window several times, while the regression stops pulling once the policy reaches y: the bounded targets regularize each update in physical units, without line search or KL term (Section 5.3). With one regression step and a matched learning rate the update reduces to a direct gradient step [51, Eq. 10; 85, Thm. 1]; the difference lies in the multistep regression, the physical step size and the separate optimizer state.

Replacing the multistep regression by one Adam step per window on the raw gradient lowers the return to 0.806 (−0.049) and slows learning throughout (Fig. 4b).

5.3 Clipping and multistep regression together: bounded targets

The two components bound the update in complementary ways. Let g_i be the clipped action gradient of sample i (one step of one window, $d = 29$ joints), m the batch median of $\|g_i\|$ and N the number of samples. The target of sample i moves its setpoints by $\delta\, g_i/\mathrm{RMS}(g)$ (Eq. (6)), where the RMS runs over all samples and joints, so the per-joint RMS change of sample i is

$$\frac{\delta\, \|g_i\|}{\sqrt{d}\ \mathrm{RMS}(g)}. \tag{7}$$

With clipping, $\|g_i\| \le \kappa m$, and at least half of the samples have $\|g_i\| \ge m$, so $\mathrm{RMS}(g) \ge m/\sqrt{2d}$. Every sample's per-joint RMS change is then at most $\sqrt{2}\,\kappa\,\delta$ (0.13 rad with our settings), whatever the scale of the gradients: each sample's target stays within a bounded distance of its current action. The bound holds for the targets the policy is regressed onto, and the regression inherits it as a regularizer. In practice the policy moves far less: after the regression steps of an update, the typical environment's setpoints change by 0.002 to 0.01 rad (per-joint RMS), and the largest change in any environment over a whole training run is 0.11 rad. Without clipping, a single sample with norm G far above the others gives $\mathrm{RMS}(g) \approx G/\sqrt{Nd}$; that sample's target moves by about $\delta\sqrt{N}$ (about 15 rad for our batch size), and the targets of all other samples barely move. The step-size budget of the whole batch goes to one sample, and the learning signal of the rest is lost. The multistep regression fixes the size of the step; clipping keeps any single sample from taking it over.

5.4 Window length

The window length sets the horizon trade-off of Section 4.2: longer windows rely less on the critic, shorter ones accumulate fewer extreme gradients (Fig. 3) and cost less per update. Halving the window to $H = 32$ lowers the return to 0.818 (−0.036) at matched wall clock and by 0.043 at matched updates (3 seeds; Fig. 4c). The sweep on the residual recipe shows both ends of the trade-off (return after 1.5 h, 3 seeds each; Table 11). Short windows fail through the critic: at $H = 16$ every seed diverges within minutes, the critic's value running away, and at $H = 32$ one of four seeds collapses the same way while the others reach 0.815–0.841. Long windows are stable but slow: $H = 128$ doubles the backward pass of each update, completes half as many updates and reaches 0.839. $H = 64$ (0.869, no collapse in any run) lies between the two.

5.5 Residual actions

The policy outputs an offset to the reference joint pose; the PD setpoint is reference plus offset. An untrained policy then already follows the reference pose, so learning refines a correction instead of reproducing the motion from scratch. The action-rate penalty is applied to the commanded setpoints, so the reward, and hence the task, is identical to predicting absolute setpoints; only the parameterization changes. Residual actions learn faster (Fig. 4d) and plateau higher: in the earlier recipe (masked, no averaging), they reach 0.82 in 0.34 h instead of 0.50 h with absolute actions, and 0.874 in 1.8 h instead of 0.864 at 3 h.

6 NablaEngine

NablaEngine is a differentiable simulation engine for the Unitree G1 that makes exact gradients cheap. It computes the dynamics of MuJoCo [75] for this robot, including the full contact model with all 502 contact pairs and the Newton constraint solver, and their analytic derivatives. It is specialized to one robot by design: fixing the kinematic tree, the contact pairs and the actuator layout at compile time is what lets one warp hold a world's step in registers and shared memory, and what makes the adjoint cheaper than another forward step. Its design rests on three ideas.

One world per warp, a megakernel for the adjoint. NablaEngine specializes the step to one robot and maps each world to one warp. Within a kernel, the world's state stays in shared memory and registers. The forward step runs as two fused kernels per substep, 8 physics kernels per 20 ms control step. The adjoint of a whole control step is a single megakernel: one warp per world reads the tape and runs the derivatives of all four substeps in reverse without leaving the kernel. (The forward step is split into two kernels per substep for occupancy; Appendix B.)

A hand-derived adjoint. The backward pass is derived by hand for every stage: kinematics, collision, the constraint solve and integration. The constraint solve is differentiated through its optimality conditions, not through the solver's iterations, with the discrete decisions (contact set, active constraints, actuator clamps) held fixed. The forward pass writes a compact tape, and the adjoint replays it in reverse. The result is the derivative of the discrete step: it matches

Table 3. Time per 20 ms control step for 4096 G1 worlds, ms (median over three processes).

	RTX PRO 6000	H100	RTX 3090	V100
Forward step				
NablaEngine	**1.21**	**1.70**	**3.28**	**5.79**
MuJoCo Warp	5.84	8.65	12.5	18.4
Step and its gradient				
NablaEngine	**1.78**	**2.73**	**5.26**	**10.6**
MuJoCo Warp with adjoint pull request	20.5	25.0	43.0	63.2

float64 finite differences of MuJoCo to 10^{-5} in median relative error, and the forward step matches float64 MuJoCo to 8.5×10^{-7} (Appendix C); the forward pass is bitwise deterministic.

Bounded where the derivative is not. Near two non-differentiable events of the G1's geometry the exact derivative is unbounded (Section 4): self-contact at vanishing distance and nearly parallel capsules. For training, NablaEngine freezes the corresponding geometric derivatives below a small threshold (1 mm, about 2°), and on force-saturated actuators it keeps the actuator damping term, which the exact derivative drops; making the latter exact does not improve learning (Section 4). Away from these events the training gradient equals the exact gradient bit for bit. These rules are part of the method, not an approximation of it: they define the gradient where the exact derivative is undefined or infinite and leave it untouched elsewhere. Appendix B gives the details.

Speed. Table 3 compares NablaEngine with MuJoCo Warp [22], the GPU implementation of the same dynamics. A forward step is 4.8× faster on an RTX PRO 6000; a step with its gradient is 11.5× faster than the open pull request that adds an adjoint to MuJoCo Warp [14], the only adjoint of MuJoCo's full contact dynamics on a GPU; MJX differentiates this model in forward mode only, and the other engines of Table 5 provide no gradient of it. On NablaEngine the gradient costs less than another forward step: a step with its gradient takes 1.78 ms against 1.21 ms for the step alone, so the entire adjoint, contact solve included, adds 0.57 ms. One RTX PRO 6000 simulates 3.39 M control steps per second, about 18.8 h of robot motion per second of wall clock. Other engines and GPUs are in Appendix B.

7 Experiments

Component ablations sit next to each component in Section 5; this section reports the main comparison against the other gradient sources and separates the engine's contribution from the algorithm's.

7.1 Setup

We train one policy on all 40 LAFAN1 clips [24] retargeted to the G1 [44] (264,705 frames, 2.45 h of motion). Each clip is evaluated once from its first frame on MuJoCo Warp with a deterministic actor and no randomization or noise. The score is the full-clip normalized return as defined by Liao et al. [40, §4]: the tracking reward minus the action-rate and joint-limit penalties, summed over the whole clip and divided by its maximum, so that a fall forfeits the rest of the clip; it is averaged over the 40 clips (Appendix G). Evaluation runs on MuJoCo Warp rather than on the training engine, so every score is also a transfer of the policy to the reference implementation of the same dynamics. Wall clock counts training only, without start-up compilation or evaluation. Each run uses one RTX PRO 6000 unless stated, and each configuration three seeds (mean ± sd). Seeds that diverge or collapse are reported separately and not averaged. A seed sets the network initialization and all random draws (action and observation noise, resets and randomization); the engine itself is deterministic.

The baselines cover the alternatives to our gradient source. SuperMimic [40], the surrogate-gradient state of the art, enters with the numbers its paper states (one H100), since its implementation is not public; the reproduction we use for the gradient analysis of Section 4 is described in Appendix E and is not a baseline. PPO [64] on NablaEngine is the zeroth-order reference, once in BeyondMimic's configuration [41] and once with SuperMimic's actor and reset sampling. SHAC [82] is the first-order reference (Section 4.1). InstantTrack [11], an open-source implementation of SuperMimic's method (a learned world model and backpropagation through it) with policy averaging, residual actions and an optimized GPU training loop, is a much stronger reproduction of SuperMimic than our own (Appendix E), and the strongest public one we know of. It is therefore the surrogate-gradient baseline we run: we train it with its own code and evaluate it with its own evaluator, whose reward, termination and normalization match ours, in our physics (self-collision on, 10 solver iterations) on the same GPU, 3 seeds.

Table 4. LAFAN1, all 40 clips: normalized return at equal training wall clock, time to SuperMimic's score of 0.82 (first crossing of the mean curve), and clips with a fall at the end of training. Ours: one RTX PRO 6000 per run, mean ± sd over 3 seeds unless noted. All runs except SuperMimic and InstantTrack (MuJoCo Warp) train on NablaEngine.

Method	0.5 h	1 h	2 h	3 h	To 0.82	Falls (/40)
∇Mimic	**0.866** ± 0.006	**0.885** ± 0.011	**0.899** ± 0.016	**0.910** ± 0.004	**0.13 h**	0.0
InstantTrack, our physics[d]	0.848 ± 0.014	0.869 ± 0.006	0.877 ± 0.001	0.882 ± 0.007	0.30 h	0.0
∇Mimic, earlier recipe[e]	0.824 ± 0.008	0.854 ± 0.005	0.872 ± 0.001	0.875 ± 0.001	0.34 h	0.0
∇Mimic, absolute actions[e]	0.821 ± 0.007	0.844 ± 0.004	0.861 ± 0.001	0.864 ± 0.001	0.50 h	0.0
PPO, BeyondMimic configuration[a]	0.465	0.529	0.641	0.638	not reached	8
PPO, SuperMimic actor and resets[a]	0.428	0.489	0.586	0.658	not reached	3
SHAC (changed guards)	0.724 ± 0.019	0.790 ± 0.019	0.807 ± 0.008	0.816 ± 0.005	2.6 h per seed[c]	2.0
SHAC (published guard)	stopped at 0.06 h by its gradient guard (return 0.373 ± 0.093)				not reached	23.0
SuperMimic (stated)[b]	n/s	n/s	n/s	n/s	within 1.2 h	n/s

[a] One seed, 6 h budget; falls at 6 h. [b] One H100, as stated in Liao et al. [40, §4.1]; n/s: not stated. [c] Per-seed crossings 2.4–3.1 h; the mean curve reaches 0.816 at 3 h. [d] Its own code and evaluator; trained in our physics on the same GPU. [e] Masked windows, no policy averaging (Section 5).

7.2 Main result: minutes instead of hours

∇Mimic reaches SuperMimic's LAFAN1 score of 0.82 in 8 min on one GPU (per seed 0.12 ± 0.01 h; Fig. 1a). SuperMimic's implementation is not public, so the comparison is with the number its paper states: 0.82 within 1.2 h on one H100 [40, §4.1]. On the same GPU type, the absolute-action version of ∇Mimic, in the earlier recipe, reaches 0.82 in 0.41 h (Appendix I). Our wall clock counts training only. InstantTrack, run in the same physics on the same GPU, reaches 0.82 in 0.30 h, well within SuperMimic's stated time. Its optimized training loop completes 20,700 iterations per hour, 2.2× as many as ∇Mimic's updates; per policy update the gap is larger. One ∇Mimic update builds the action targets of one 64-step exact-gradient window and regresses onto them in 8 minibatch steps; one InstantTrack iteration takes one Adam step on the gradient of a 32-step window through its world model, after 500 iterations that train only the world model. ∇Mimic reaches 0.82 after 1,250 updates and InstantTrack after 5,000 policy steps; 0.866 after 4,250 against 15,500.

Training does not stop there. The return levels off at about 0.91: 0.910 at 3 h, against InstantTrack's 0.882 and SuperMimic's best reported 0.866. No seed falls on any clip from 31 min on, and the three seeds agree to a standard deviation of 0.004 at 3 h. Runs of 6 h (with a 500-update warmup, Table 11) confirm the plateau: the 3-seed mean stays between 0.905 and 0.907 in every hour from 2 to 6 h. No run diverged; the collapses in Table 2 occur only in variants that remove clipping or change the window. The policies of the earlier recipe hold up beyond the nominal protocol: under the training-time randomization and observation noise, external pushes and random start frames, ∇Mimic keeps the highest return in every condition and the highest success rate in all but whole-clip pushes of 1 m/s (Appendix H), and its seed mean is above that of SHAC with changed guards on every one of the 40 clips.

The other gradient sources on the same engine are far behind. PPO with BeyondMimic's configuration reaches 0.64 after 3 h and 0.70 after 6 h, even though NablaEngine makes each PPO iteration 2.9× faster than stock MuJoCo Warp (0.123 vs 0.355 s): cheaper samples do not close the gap, the gradient does. In transitions, ∇Mimic reaches 0.82 after 3.1×10^8; PPO consumes 1.7×10^{10} in 6 h without reaching it. This matches the independent PPO result in SuperMimic's paper: PPO-Sim reaches 0.60, 0.70 and 0.72 after 18 h on 1, 4 and 8 H100s. SHAC stops on its gradient guard at 0.06 h; with the guard relaxed so that exploding updates are clipped instead of halting training, it keeps learning and crosses 0.82 after 2.4 to 3.1 h per seed (mean curve 0.816 at 3 h), 18 to 24× later than ∇Mimic on the same engine. Table 4 gives the numbers; all curves and the cost to reach 0.82 are in Appendix I.

Is it the gradient or the recipe? The recipe does not explain the gain by itself. Applied to the world-model gradients of the SuperMimic reproduction, per-sample clipping leaves the return after 2.7 h at 0.80 against 0.79 without it, and clipping with multistep regression together collapses after 1.5 h (Section 4). Applied to exact gradients, the same components turn a method that halts within minutes into one that reaches 0.82 in 8 minutes. The components are what make exact gradients usable; the gradient is what they make usable.

7.3 Same algorithm, two simulators

Is the speed-up the engine's or the algorithm's? Training the absolute-action version of ∇Mimic on the MuJoCo Warp adjoint, with NablaEngine's rules for unbounded derivatives ported to it, answers this: per update the two curves overlap

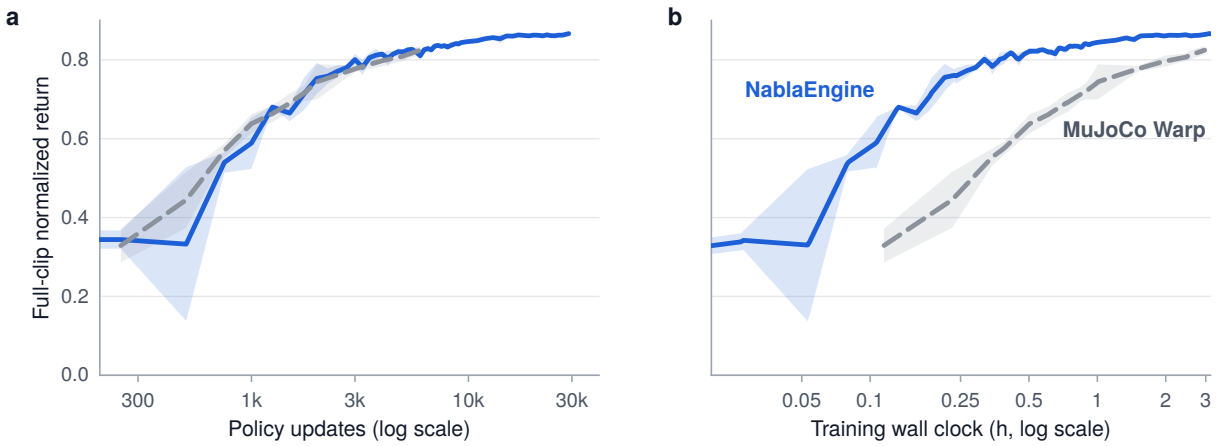


Figure 5. The engine changes speed, not learning. ∇Mimic (absolute actions) on NablaEngine and on the MuJoCo Warp adjoint with the engine's gradient semantics, 3 seeds each (mean ± sd) on the same GPU, LAFAN1. (a) Per policy update the curves overlap. (b) Per hour of training, NablaEngine is faster.

(Fig. 5a; at matched updates the 3-seed means differ by at most about 0.02, in either direction); per hour NablaEngine is 4.7× ahead (2.02 h vs 0.43 h to 4k updates; 4.4× per update, 1.63 vs 0.37 s). The engine changes speed, not learning.

8 Limitations and future work

One robot, the Unitree G1, on flat ground is the scope of this version; the kernel design, one world per warp with a hand-derived adjoint of the whole control step, carries to other robots with their own compile-time specialization, and uneven terrain needs height fields and further collision types. Our results are on LAFAN1 in simulation: we have not yet trained on larger motion libraries or deployed the policies on hardware. The gradient holds the discrete decisions of the contact solve fixed, so it is the exact derivative of the step almost everywhere rather than a smoothed gradient across contact changes. Per-sample clipping rescales the norms of 9–21% of the samples per update, preserving their direction. SuperMimic enters with its stated numbers, and the H100 run of Appendix I puts the two on the same GPU type; speed comparisons between simulators are made within one GPU type. Extending NablaEngine to more robots and terrains, scaling ∇Mimic to large motion libraries and transferring the policies to real robots are the natural next steps.

9 Conclusion

First-order policy learning has long promised far more informative updates than zeroth-order methods, but on contact-rich whole-body control two things held it back: rare unbounded samples in the gradient, and the cost of computing it. NablaEngine removes the second for the Unitree G1: it computes the full MuJoCo dynamics of the robot, all 502 contact pairs included, with analytic gradients that match float64 finite differences, and its adjoint costs less than another forward step. ∇Mimic removes the first: per-sample median-relative clipping bounds the rare extreme samples, and multistep regression onto gradient-derived action targets reuses each costly sample with action targets whose size does not depend on their scale. Together they train a single policy that tracks all 40 LAFAN1 clips and reaches the score of the world-model state of the art in 8 minutes on one GPU.

Cutting the cost of a tracking policy from hours to minutes makes training on far larger and more diverse motion corpora practical, and can thereby accelerate the scaling of humanoid foundation models.

References

[1] Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 308–318, 2016. doi: 10.1145/2976749.2978318. URL https://arxiv.org/abs/1607.00133.

[2] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller.

Maximum a Posteriori Policy Optimisation. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://arxiv.org/abs/1806.06920.

[3] Dyuman Aditya, Jin Cheng, Clemens Schwarke, Quan Nguyen, Gaurav Sukhatme, Stelian Coros, and Gabriele Fadini. Bundled contact gradients: Stabilizing differentiable simulation for deployable dynamic tasks, 2026.

[4] Joseph Amigo, Rooholla Khorrambakht, Elliot Chane-Sane, Nicolas Mansard, and Ludovic Righetti. First order model-based RL through decoupled backpropagation. In *Proceedings of the 9th Conference on Robot Learning (CoRL)*, volume 305 of *PMLR*, pages 5234–5255, 2025.

[5] Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. Differentially Private Learning with Adaptive Clipping. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://arxiv.org/abs/1905.03871.

[6] Joshua Bagajo, Clemens Schwarke, Victor Klemm, Ignat Georgiev, Jean-Pierre Sleiman, Jesus Tordesillas, Animesh Garg, and Marco Hutter. DiffSim2Real: Deploying quadrupedal locomotion policies purely trained in differentiable simulation, 2024. CoRL 2024 Workshop on Differentiable Optimization Everywhere.

[7] Bones Studio. BONES-SEED: Skeletal Everyday Embodiment Dataset. https://huggingface.co/datasets/bones-studio/seed, 2026. 142,220 annotated motion sequences (288 h) from 522 actors, SOMA and Unitree G1 formats.

[8] Justin Carpentier and Nicolas Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and Systems (RSS)*, 2018. doi: 10.15607/RSS.2018.XIV.038.

[9] Zixuan Chen, Mazeyu Ji, Xuxin Cheng, Xuanbin Peng, Xue Bin Peng, and Xiaolong Wang. GMT: General Motion Tracking for Humanoid Whole-Body Control. *arXiv preprint arXiv:2506.14770*, 2025.

[10] Ignasi Clavera, Violet Fu, and Pieter Abbeel. Model-Augmented Actor-Critic: Backpropagating through Paths. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/2005.08068.

[11] Huy-Quang Dao. InstantTrack: Fast humanoid motion tracking with a learned world model. https://github.com/Huy-Quang-Dao/instantTrack, 2026. Version 1.1.0, code release.

[12] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13:6, 2019. doi: 10.3389/fnbot.2019.00006. arXiv:1611.01652.

[13] etaoxing. Fix pyramidal-cone contact skip and joint actuatorfrcrange clamp in the adjoint. GitHub pull request #1 to etaoxing/mujoco_warp, opened 2026-10-23, open as of 2026-10-29, 2026. URL https://github.com/etaoxing/mujoco_warp/pull/1.

[14] etaoxing. hybrid analytic differentiability. GitHub pull request #1535, google-deepmind/mujoco_warp, opened 2026-07-19, open as of 2026-10-29, 2026. URL https://github.com/google-deepmind/mujoco_warp/pull/1535.

[15] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation. In *NeurIPS Datasets and Benchmarks Track*, 2021.

[16] Levi Fussell, Kevin Bergamin, and Daniel Holden. SuperTrack: Motion Tracking for Physically Simulated Characters using Supervised Learning. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 40(6):197:1–197:13, 2021. doi: 10.1145/3478513.3480527.

[17] Genesis Authors. Genesis: A Generative and Universal Physics Engine for Robotics and Beyond. https://github.com/Genesis-Embodied-AI/Genesis, December 2024.

[18] Ignat Georgiev, Krishnan Srinivasan, Jie Xu, Eric Heiden, and Animesh Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pages 15418–15437, 2024.

[19] Ignat Georgiev, Varun Giridhar, Nicklas Hansen, and Animesh Garg. PWM: Policy Learning with Multi-Task World Models. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2407.02466.

[20] James Gleeson, Daniel Snider, Yvonne Yang, Moshe Gabel, Eyal de Lara, and Gennady Pekhimenko. Optimizing Data Collection in Deep Reinforcement Learning. In *MLBench Workshop*, 2022. URL https://arxiv.org/abs/2207.07736.

[21] Google DeepMind. MuJoCo XLA (MJX). https://mujoco.readthedocs.io/en/stable/mjx.html, 2024. Software, part of the MuJoCo repository; no standalone paper.

[22] Google DeepMind and NVIDIA. MuJoCo Warp (MJWarp). https://github.com/google-deepmind/mujoco_warp, 2025. Software; no standalone paper found.

[23] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1912.01603.

[24] Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust Motion In-betweening. *ACM Transactions on Graphics*, 39(4):60:1–60:12, 2020. doi: 10.1145/3386569.3392480. Proc. SIGGRAPH 2020; introduces the LAFAN1 dataset.

[25] Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. OmniH2O: Universal and Dexterous Human-to-Humanoid Whole-Body Teleoperation and Learning. In *Conference on Robot Learning (CoRL)*, 2024.

[26] Tairan He, Zhengyi Luo, Wenli Xiao, Chong Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Learning Human-to-Humanoid Real-Time Whole-Body Teleoperation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[27] 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 "Jim" Fan, Yuke Zhu, Changliu Liu, and Guanya Shi. ASAP: Aligning Simulation and Real-World Physics for Learning Agile Humanoid Whole-Body Skills. In *Robotics: Science and Systems (RSS)*, 2025.

[28] Nicolas Heess, Greg Wayne, David Silver, Timothy Lillicrap, Yuval Tassa, and Tom Erez. Learning Continuous Control Policies by Stochastic Value Gradients. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. URL https://arxiv.org/abs/1510.09142.

[29] Taylor A. Howell, Simon Le Cleac'h, Jan Brüdigam, Qianzhong Chen, Jiankai Sun, J. Zico Kolter, Mac Schwager, and Zachary Manchester. Dojo: A differentiable physics engine for robotics, 2022.

[30] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations (ICLR)*, 2020.

[31] Mazeyu Ji, Xuanbin Peng, Fangchen Liu, Jialong Li, Ge Yang, Xuxin Cheng, and Xiaolong Wang. ExBody2: Advanced Expressive Humanoid Whole-Body Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

[32] Gijeong Kim, Seungbin Oh, Dongyun Kang, and Hae-Won Park. Differentiable contact dynamics for legged locomotion: Policy learning and contact-implicit trajectory optimization, 2026.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

[34] Anastasia Koloskova, Hadrien Hendrikx, and Sebastian U. Stich. Revisiting Gradient Clipping: Stochastic bias and tight convergence guarantees. In *International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2305.01588.

[35] Aleš Kučera and Karel Zimmermann. Ostrich: Taking large strides through stiff contact in differentiable dynamics, 2026.

[36] Jonathan Wilder Lavington, Sharan Vaswani, Reza Babanezhad, Mark Schmidt, and Nicolas Le Roux. Target-based Surrogates for Stochastic Optimization. In *International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2302.02607.

[37] Quentin Le Lidec, Louis Montaut, Yann de Mont-Marin, Fabian Schramm, Emilien Biré, and Justin Carpentier. Highly-efficient differentiable simulation for robotics, 2024. v1 title: End-to-End and Highly-Efficient Differentiable Simulation for Robotics.

[38] Yu Lei, Zhengyi Luo, Tairan He, Jinkun Cao, Guanya Shi, and Kris Kitani. Scalable Humanoid Whole-Body Control via Differentiable Neural Network Dynamics. In *ICLR 2025 Workshop on World Models: Understanding, Modelling and Scaling*, 2025. URL https://openreview.net/forum?id=FPgKt7bA8w. Referred to as SuperDyno in SuperMimic (arXiv:2610.28480).

[39] Sergey Levine and Vladlen Koltun. Guided Policy Search. In *International Conference on Machine Learning (ICML)*, volume 28 of *PMLR*, 2013. URL https://proceedings.mlr.press/v28/levine13.html.

[40] Qiayuan Liao, Yiyang Shao, Junfeng Long, Kevin Zakka, Roberto Horowitz, Pieter Abbeel, and Koushil Sreenath. SuperMimic: Scaling Humanoid Whole-Body Control with One Differentiable World Model. In *Conference on Robot Learning (CoRL)*, 2026.

[41] 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. *Science Robotics*, 11 (117):eadx8924, 2026. doi: 10.1126/scirobotics.adx8924.

[42] Zhengyi Luo, Jinkun Cao, Alexander Winkler, Kris Kitani, and Weipeng Xu. Perpetual Humanoid Control for Real-time Simulated Avatars. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[43] Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Fernando Castañeda, Sirui Chen, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, Jinhyung Park, David Sami, Zi Wang, Xingye Da, Runyu Ding, Cyrus Hogg, Lina Song, Edy Lim, Eugene Jeong, Tairan He, Haoru Xue, Wenli Xiao, Simon Yuen, Jan Kautz, Yan Chang, Umar Iqbal, Linxi "Jim" Fan, and Yuke Zhu. SONIC: Supersizing Motion Tracking for Natural Humanoid Whole-Body Control. *Science Robotics*, 11(117):eaed4592, 2026. doi: 10.1126/scirobotics.aed4592.

[44] lvhaidong. LAFAN1 Retargeting Dataset. https://huggingface.co/datasets/lvhaidong/LAFAN1_Retargeting_Dataset, 2025. LAFAN1 retargeted to Unitree H1, H1-2 and G1; referred to as Unitree-retargeted LAFAN1 in BeyondMimic.

[45] Michel Ma, Tianwei Ni, Clement Gehring, Pierluca D'Oro, and Pierre-Luc Bacon. Do Transformer World Models Give Better Policy Gradients? In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.05290.

[46] Miles Macklin. Warp: A high-performance python framework for GPU simulation and graphics. NVIDIA GPU Technology Conference (GTC), March 2022. URL https://github.com/nvidia/warp.

[47] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of Motion Capture as Surface Shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. doi: 10.1109/ICCV.2019.00554.

[48] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[49] Ibrahim Merad and Stéphane Gaïffas. Robust Stochastic Optimization via Gradient Quantile Clipping. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2309.17316.

[50] Luke Metz, C. Daniel Freeman, Samuel S. Schoenholz, and Tal Kachman. Gradients are not all you need, 2021.

[51] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. PODS: Policy optimization via differentiable simulation. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *PMLR*, pages 7805–7817, 2021.

[52] Igor Mordatch and Emanuel Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2014. doi: 10.15607/RSS.2014.X.052. URL https://www.roboticsproceedings.org/rss10/p52.html.

[53] MuJoCo. [MJX] jax.lax.while_loop in solver.py prevents computation of backward gradients. GitHub issue #2259, google-deepmind/mujoco, opened 2024-11-29, 2024. URL https://github.com/google-deepmind/mujoco/issues/2259.

[54] Newton. Improve the contact differentiability. GitHub pull request #2164, newton-physics/newton, merged 2026-03-30, 2026. URL https://github.com/newton-physics/newton/pull/2164.

[55] NVIDIA, Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, et al. Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning, 2025. URL https://arxiv.org/abs/2511.04831. arXiv preprint.

[56] Ku Onoda, Paavo Parmas, Manato Yaguchi, and Yutaka Matsuo. Does "do differentiable simulators give better policy gradients?" give better policy gradients? In *International Conference on Learning Representations (ICLR)*, 2026.

[57] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. In *International Conference on Machine Learning (ICML)*, 2013. URL https://arxiv.org/abs/1211.5063.

[58] Anselm Paulus, A. René Geist, Pierre Schumacher, Vít Musil, Simon Rappenecker, and Georg Martius. Differentiable simulation of hard contacts with soft gradients for learning and control. In *International Conference on Learning Representations (ICLR)*, 2026.

[59] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Transactions on Graphics*, 37(4), 2018. doi: 10.1145/3197517.3201311. Proc. SIGGRAPH 2018.

[60] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning, 2019. URL https://arxiv.org/abs/1910.00177. arXiv preprint.

[61] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[62] Jiawei Ren, Cunjun Yu, Siwei Chen, Xiao Ma, Liang Pan, and Ziwei Liu. DiffMimic: Efficient motion mimicking with differentiable physics. In *International Conference on Learning Representations (ICLR)*, 2023.

[63] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, 2021. URL https://arxiv.org/abs/2109.11978.

[64] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347. arXiv preprint.

[65] Clemens Schwarke, Victor Klemm, Joshua Bagajo, Jean-Pierre Sleiman, Ignat Georgiev, Jesus Tordesillas, Animesh Garg, and Marco Hutter. Learning deployable locomotion control via differentiable simulation. In *Conference on Robot Learning (CoRL)*, volume 305 of *PMLR*, 2025.

[66] Brennan Shacklett, Luc Guy Rosenzweig, Zhiqiang Xie, Bidipta Sarkar, Andrew Szot, Erik Wijmans, Vladlen Koltun, Dhruv Batra, and Kayvon Fatahalian. An Extensible, Data-Oriented Architecture for High-Performance, Many-World Simulation. *ACM Transactions on Graphics (SIGGRAPH)*, 42(4), 2023. doi: 10.1145/3592427.

[67] Umut Simsekli, Levent Sagun, and Mert Gurbuzbalaban. A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks. In *International Conference on Machine Learning (ICML)*, 2019. URL https://arxiv.org/abs/1901.06053.

[68] Sanghyun Son, Laura Yu Zheng, Ryan Sullivan, Yi-Ling Qiao, and Ming C. Lin. Gradient informed proximal policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[69] Yunlong Song, Sangbae Kim, and Davide Scaramuzza. Learning quadruped locomotion using differentiable simulation. In *Conference on Robot Learning (CoRL)*, 2024.

[70] H. J. Terry Suh, Tao Pang, and Russ Tedrake. Bundled gradients through contact via randomized smoothing. *IEEE Robotics and Automation Letters*, 2022.

[71] H. J. Terry Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

[72] Chen Tessler, Yunrong Guo, Ofir Nabati, Gal Chechik, and Xue Bin Peng. MaskedMimic: Unified Physics-Based Character Control Through Masked Motion Inpainting. *ACM Transactions on Graphics*, 43(6), 2024. doi: 10.1145/3687951. Proc. SIGGRAPH Asia 2024.

[73] The Newton Contributors. Newton: GPU-accelerated physics simulation for robotics and simulation research. https://github.com/newton-physics/newton, 2025. Software (Linux Foundation project; Disney Research, Google DeepMind, NVIDIA).

[74] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061, 2014.

[75] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[76] Sharan Vaswani, Olivier Bachem, Simone Totaro, Robert Mueller, Shivam Garg, Matthieu Geist, Marlos C. Machado, Pablo Samuel Castro, and Nicolas Le Roux. A general class of surrogate functions for stable and efficient reinforcement learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022. URL https://arxiv.org/abs/2108.05828.

[77] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C. Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. In *Robotics: Science and Systems (RSS)*, 2021. doi: 10.15607/RSS.2021.XVII.034.

[78] Ronald J. Williams and Jing Peng. An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories. *Neural Computation*, 2(4):490–501, 1990. doi: 10.1162/neco.1990.2.4.490.

[79] Weiji Xie, Jinrui Han, Jiakun Zheng, Huanyu Li, Xinzhe Liu, Jiyuan Shi, Weinan Zhang, Chenjia Bai, and Xuelong Li. KungfuBot: Physics-Based Humanoid Whole-Body Control for Learning Highly-Dynamic Skills. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[80] Eliot Xing, Vernon Luk, and Jean Oh. Stabilizing reinforcement learning in differentiable multiphysics simulation. In *International Conference on Learning Representations (ICLR)*, 2025.

[81] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. In *Robotics: Science and Systems (RSS)*, 2021.

[82] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations (ICLR)*, 2022.

[83] Heyuan Yao, Zhenhua Song, Baoquan Chen, and Libin Liu. ControlVAE: Model-Based Learning of Generative Controllers for Physics-Based Characters. *ACM Transactions on Graphics*, 41(6), 2022. doi: 10.1145/3550454.3555434. Proc. SIGGRAPH Asia 2022.

[84] Kangning Yin, Weishuai Zeng, Ke Fan, Minyue Dai, Zirui Wang, Qiang Zhang, Zheng Tian, Jingbo Wang, Jiangmiao Pang, and Weinan Zhang. UniTracker: Learning Universal Whole-Body Motion Tracker for Humanoid Robots. *arXiv preprint arXiv:2507.07356*, 2025.

[85] Haoxiang You, Yilang Liu, and Ian Abraham. Accelerating visual-policy learning through parallel differentiable simulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[86] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. MuJoCo Playground. In *Robotics: Science and Systems (RSS)*, 2025. URL https://arxiv.org/abs/2502.08844.

[87] Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. mjlab: A lightweight framework for GPU-accelerated robot learning, 2026.

[88] Yanjie Ze, Zixuan Chen, João Pedro Araújo, Zi-ang Cao, Xue Bin Peng, Jiajun Wu, and C. Karen Liu. TWIST: Teleoperated Whole-Body Imitation System. *arXiv preprint arXiv:2505.02833*, 2025.

[89] Huzhenyu Zhang, Shenghai Yuan, Wenrui Yan, Li Ma, Hengjie Li, Jingcheng Pang, and Dmitry Yudin. GPUSimBench: Towards Scalable and Reliable GPU-Accelerated Simulators in Embodied AI. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. URL https://arxiv.org/abs/2607.13059. Accepted; arXiv preprint.

[90] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1905.11881.

[91] Ziyu Zhang, Sergey Bashkirov, Dun Yang, Yi Shi, Michael Taylor, and Xue Bin Peng. Physics-Based Motion Imitation with Adversarial Differential Discriminators. In *SIGGRAPH Asia 2025 Conference Papers*, 2025. doi: 10.1145/3757377.3763819.

[92] Hai Zhong, Xun Wang, Zhuoran Li, and Longbo Huang. Reparameterization proximal policy optimization. In *International Conference on Machine Learning (ICML)*, 2026. Venue from ICML 2026 virtual poster page.

A Task details

Data. The 40 LAFAN1 clips [24] are the Unitree G1 retarget of the LAFAN1 Retargeting Dataset [44] (its `g1/` folder), used without modification. At load time the 30 FPS clips are resampled to the 50 Hz control rate (linear interpolation for root position and joint angles, spherical linear interpolation for the root orientation), and body poses and velocities are computed by forward kinematics on the robot model.

Reward. The reward follows BeyondMimic [41] with the weights and widths of SuperMimic [40, Table A.2]. Tracking terms have the form $w \exp(-e/\sigma^2)$, with e the squared error (orientation: squared rotation angle), averaged over the bodies for the body terms. The body terms are evaluated over 14 bodies, with the reference re-anchored to the robot's torso in horizontal position and yaw; the anchor terms use the torso itself.

Term	σ	w
Body position	0.3	1.0
Body orientation	0.4	1.0
Body linear velocity	1.0	1.0
Body angular velocity	3.14	1.0
Anchor position	0.3	0.5
Anchor orientation	0.4	0.5
Action rate (penalty)	–	−0.1
Joint limits (penalty)	–	−2.0

Every term is multiplied by the control period $\Delta t = 0.02$ s, so the per-step tracking reward is at most 0.1. During training only, ∇Mimic adds the termination penalty of Table 10; evaluation uses the reward above.

Observations. The actor sees a history of the last 10 proprioception vectors (93 dimensions each): 29 joint positions relative to the default pose, 29 joint velocities, base angular velocity, gravity direction in the base frame and the previous action. Uniform noise of SuperMimic's Table A.1 is added during training. The goal consists of the next 10 reference frames, each with 68 features expressed in the reference's own frame (joint positions and velocities, gravity direction, root angular velocity, root linear velocity in the heading frame, root height), plus the reference anchor's orientation relative to the robot's anchor (6 dimensions, two rotation-matrix columns), as in BeyondMimic's variant without state estimation. The critic is privileged: it sees noise-free proprioception, the goal, root errors and the randomization parameters.

Domain randomization. Per episode, following SuperMimic's Table A.3: foot friction $\mathcal{U}[0.3, 1.2]$, joint encoder bias $\mathcal{U}[-0.01, 0.01]$ rad and a torso center-of-mass offset. None is applied at evaluation.

B Engine details

Model. NablaEngine simulates the mjlab Unitree G1 model [87]: a free base and 29 actuated hinge joints, 31 bodies, and 502 contact pairs among capsules, spheres and the floor, self-collision included. The dynamics follow MuJoCo [75]:

- recursive Newton–Euler and composite rigid-body algorithms;
- MuJoCo's soft-constraint contact model (solref/solimp) with pyramidal friction cones;
- joint limits, dof friction loss and actuator force limits;
- Newton's method with exact line search, 10 iterations, tolerance 10^{-8};
- the implicitfast integrator, 4 substeps of 5 ms per 20 ms control step, with the control held;
- PD position actuators.

Per-world foot friction and torso center-of-mass offsets support domain randomization.

Kernel structure. One world maps to one thread block of 32 threads (one warp). Within each kernel the world's state stays in shared memory and registers (6.9 kB and 8.2 kB per world for the two forward kernels), and the model's constants sit in GPU constant memory. Each substep runs two forward kernels:

- the first computes kinematics, the bias and mass terms, actuator forces, collision and the constraint rows;
- the second runs the Newton solve and the integration.

A control step is therefore 8 physics kernels for all worlds; the handoff between the two kernels (mass matrix, forces, constraint rows and contact Jacobians, about 33 kB per world) and the state between substeps pass through global memory. Up to 11–12 worlds are resident per SM on the RTX PRO 6000 and the H100. The whole control step can also run as one kernel, but it needs the full per-world state (31.9 kB) in shared memory, fits fewer worlds per SM, and is slower (forward with tape, 4096 worlds, kernel time: 2.76 vs 1.23 ms on the RTX PRO 6000, 4.37 vs 1.76 ms on the H100, 6.79 vs 3.13 ms on the RTX 3090, 10.88 vs 5.98 ms on the V100). The adjoint of a control step is one kernel: one warp per world reads the tape and runs the derivatives of all four substeps in reverse, with the per-world adjoint state in shared memory.

Tape and adjoint. For the gradient, the forward pass also writes a tape of 1,472 float32 values per world and substep: 23.6 kB per world and control step, or 96.5 MB at 4096 worlds. The adjoint replays the contact set, constraint rows and solver state from it and runs every stage in reverse:

- integration;
- the constraint solve, differentiated through its optimality conditions with the discrete decisions (contact set, active constraints, actuator clamps) held fixed;
- collision;
- kinematics and dynamics.

Constraint-solve adjoint. Per substep, with state (q, v) and control u, let $M(q)$ be the mass matrix, $c(q, v)$ the bias force, $\tau(q, v, u)$ the actuator force and $a_0 = M^{-1}(\tau - c)$ the unconstrained acceleration. The constraint solve returns

$$a^\star = \arg\min_a \tfrac{1}{2}(a - a_0)^\top M(a - a_0) + \sum_e s_e(J_e a - r_e), \tag{8}$$

over constraint rows e (joint limits and the edges of the pyramidal contact cones), with Jacobians $J_e(q)$, reference accelerations $r_e(q, v) = -b_e J_e v - k_e \iota_e d_e$, and the penalty $s_e(x) = \frac{1}{2} D_e x^2$ on the active side of row e and 0 otherwise (D_e from the impedance $\iota_e(d_e)$). With the active set $\mathcal{A}$ held fixed, the optimality condition

$$F(a, \theta) = M(a - a_0) + \sum_{e \in \mathcal{A}} D_e J_e^\top (J_e a - r_e) = 0, \qquad \theta = (q, v, u), \tag{9}$$

is smooth in both arguments, and $\partial F/\partial a = H = M + \sum_{e \in \mathcal{A}} D_e J_e^\top J_e$ is symmetric positive definite. By the implicit function theorem, $\mathrm{d}a^\star/\mathrm{d}\theta = -H^{-1}\, \partial F/\partial \theta$, so for an output cotangent $\bar{a}$

$$\lambda = H^{-1}\bar{a}, \qquad \bar{\theta} = -\nabla_\theta \left[\lambda^\top F(a^\star, \theta)\right] \quad (a^\star, \lambda \text{ held constant}). \tag{10}$$

Since $Ma_0 = \tau - c$,

$$\lambda^\top F = \lambda^\top \mathrm{ID}(q, v, a^\star) - \lambda^\top \tau(q, v, u) + \sum_{e \in \mathcal{A}} D_e (J_e \lambda)(J_e a^\star - r_e), \tag{11}$$

where $\mathrm{ID}(q, v, a) = M(q)a + c(q, v)$ is inverse dynamics. $\bar{\theta}$ is therefore the sum of three gradients of scalars:

- inverse dynamics contracted with λ, which is one reverse pass of recursive Newton–Euler;
- the actuator law contracted with λ, which is zero for force-clamped actuators in exact mode;
- the constraint term, which depends on q through J_e, d_e and the contact frame and is differentiated through the collision geometry (contact point, normal, distance).

The implicitfast integrator ($Qa_{\mathrm{adv}} = Ma^\star$ with $Q = M + hK_v$, $v' = v + ha_{\mathrm{adv}}$, $q' = q \oplus hv'$) is differentiated directly: $y = Q^{-1}\bar{a}_{\mathrm{adv}}$ and $\bar{a}^\star = My$, plus the q-dependence of M through $\nabla_q[y^\top M(q)(a^\star - a_{\mathrm{adv}})]$. A control step chains the 4 substeps in reverse with u held, summing $\bar{u}$. Held fixed are:

- the contact set and the collision branch;
- the active set $\mathcal{A}$, with each pyramid edge separately;
- the actuator clamps.

The derivative is that of the optimality conditions at the returned iterate $a^\star$, not of the solver iterations; the warm start receives no gradient.

Gradient modes. Exact mode (clamp_kv = 1, normal = 0) is the derivative of the discrete step. The training default (clamp_kv = 0, normal = 3) keeps the actuator damping term on force-clamped actuators, freezes the derivative of a self-contact normal when the two geoms' core distance (between sphere centers or capsule axes) is below 1 mm, and holds the segment parameters of near-parallel capsule pairs (below about 2°) fixed. On held-out states the two modes have the same median gradient errors on groups G0, G5 and G6; on G4, the states with clamped actuators, the truth has no clamp_kv = 0 counterpart.

Unbounded derivatives. Two geometric events of the G1 make the exact derivative of the step unbounded.

- *Self-contact at vanishing distance.* The derivative of the contact normal scales as the inverse of the distance between the two geoms' cores. One state with nearly touching feet had a step Jacobian norm of about 3×10^8. On 24 training windows of $H = 64$ (2048 environments each), the gradient grew by more than 10^6 through 5 windows, by up to 4.9×10^{21}, and 32 rows were not finite. With the normal's derivative frozen below 1 mm, 1 of 24 windows grew by more than 10^6 (at most 2.2×10^6) and no row was non-finite.
- *Nearly parallel capsules.* The closest points of two capsule segments are ill-conditioned, roughly as $1/\sin^2\theta$. On 172 constructed foot-crossing states, the exact rule amplified a vector–Jacobian product by up to 3.6×10^8 per control step (median 29); repeated over 64 steps, 40 of the 172 states overflowed to non-finite values. With the segment parameters held fixed below about 2°, the amplification was at most 109 and no state overflowed; ordinary held-out states reach at most 273.

Other events (contact on or off, a constraint row at the edge of its zone, actuator saturation, the solver at its iteration limit) make the derivative jump rather than diverge. The training default removes the two unbounded terms and, on force-saturated actuators, keeps the actuator damping term (Appendix B, gradient modes); on held-out states without these events it equals the exact gradient bit for bit.

Determinism. Each world's whole step runs in one warp with a fixed assignment of work to lanes, and worlds never interact. Reductions are warp shuffles in a fixed order, with no floating-point atomics. The operations the adjoint replays from the tape use explicitly rounded arithmetic, so the replay equals the forward pass bit for bit. Outputs and gradients are bitwise identical from run to run: across three processes on 8192 held-out states (V100), and across repeated calls within a process in the verification suite (V100, H100).

Timing protocol. Times are per 20 ms control step for 4096 worlds, on held-out states. Each value is the median over three processes. Every simulator runs in its own standard configuration, and speed-ups are always within one GPU. Isaac Lab rows use Isaac Lab's own G1 asset and task defaults; Isaac Sim requires an RT-core GPU, so they are not available on the H100 and V100. The MuJoCo C reference runs on one AMD EPYC 9555 (16 threads) with MuJoCo's own thread pool. The MuJoCo Warp gradient times are those of the open adjoint pull request [14] as published; its public follow-up fixes [13], which the accuracy tables also report, change the cost by less than 2%. The MuJoCo Warp training run of Section 7.3 uses the pull request with these fixes and with NablaEngine's training gradient rules ported to it.

All engines. Table 5 lists forward times of other GPU engines in their standard configurations, Table 6 the step-and-gradient times and Fig. 6 their split.

Table 5. Forward time per control step, 4096 worlds, ms (process medians, held-out states). n/s: not supported, because Isaac Sim requires an RT-core GPU. Speed-ups hold within one GPU.

Engine	RTX PRO 6000	H100	RTX 3090	V100
NablaEngine	**1.21**	**1.70**	**3.28**	**5.79**
MuJoCo Warp 3.17.0	5.84	8.65	12.5	18.4
Genesis 1.4.5	13.45	16.36	36.73	139.11
MJX 3.17 (same model)	255	216	510	710
MJX 3.17 (Menagerie G1, fewer contacts)	50.7	58.4	129.2	150.7
Isaac Lab 3.0, Newton (3 box colliders)	12.9	n/s	35.15	n/s
Isaac Lab 3.0, PhysX (3 box colliders)	27.5	n/s	45.16	n/s
Isaac Lab 3.0, PhysX (38 shapes, self-collision)	54.1	n/s	87.45	n/s
MuJoCo C 3.17 (CPU, float64; AMD EPYC 9555, 16 threads)	34.7			

Table 6. One control step and its gradient from the input state, 4096 worlds, ms.

	RTX PRO 6000	H100	RTX 3090	V100
NablaEngine (forward with tape + adjoint)	**1.78**	**2.73**	**5.26**	**10.6**
MuJoCo Warp + PR #1535 (adjoint graph)	20.5	25.0	43.0	63.2
Speed-up	11.5×	9.2×	8.2×	6.0×

C Full accuracy tables

Held-out benchmark: four corpora of 4096 states each (16,384 in total). The truth is MuJoCo 3.17 C in float64, with the same model and per-world randomization, the Newton solver at tolerance 10^{-13} and up to 2000 iterations, and one control step from each state. Gradient errors are relative errors of vector–Jacobian products with three stored random output directions per state (i.i.d. standard-normal entries, fixed seed, unnormalized), against float64 central finite differences at step 10^{-6}, floored per state at the truth's own precision (the difference between the finite differences at steps 10^{-7} and 10^{-6}, typically about 10^{-7}).

Groups. Group labels come from MuJoCo C alone, per control step (any substep). A state is *smooth* if none of the following occurs: a self-contact core distance below 5 mm; a contact flip (the contact set changes, or a distance below 10^{-6} m); a kink (a constraint row at the edge of its zone, $|x_e|/(|J_e a| + |r_e|) < 10^{-4}$); a contact frame near its basis switch; the solver at its iteration limit.

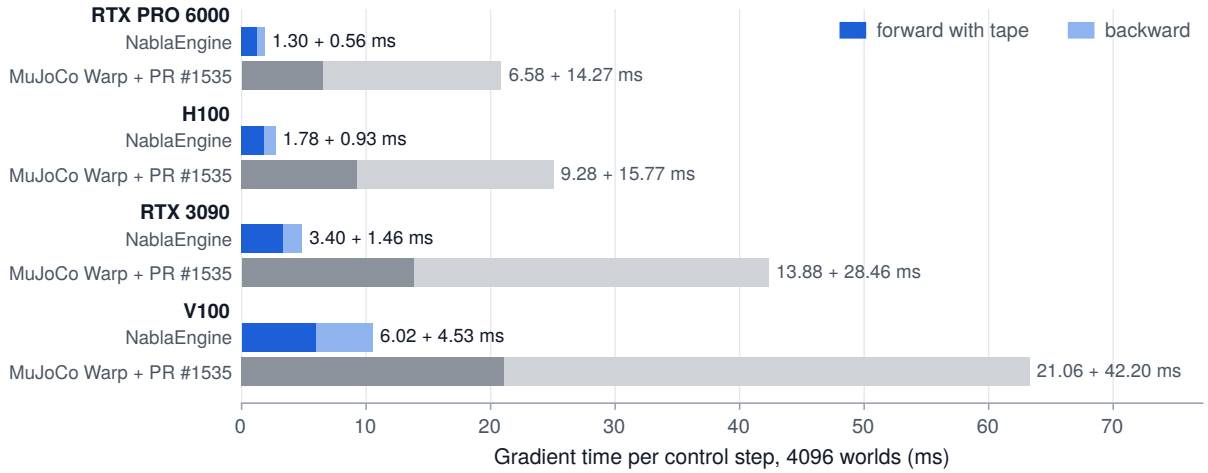


Figure 6. Gradient time per control step for 4096 worlds, split into the forward pass with tape (dark) and the backward pass (light), for NablaEngine and MuJoCo Warp with the open pull request [14]. The two parts are timed separately, so their sum can differ from the single-call times in Table 6.

- G0: smooth, no clamped actuator.
- G4: smooth, a clamped actuator (PD force at or beyond its range).
- G5: smooth, an active joint-limit row, no clamped actuator.
- G6: smooth, a contact between two robot geoms (self contact), no clamped actuator.

Accuracy is verified on every state at which a derivative exists: smooth states with and without self-contact, active joint limits and clamped actuators. At non-smooth states the derivative of the step jumps or is undefined and finite differences measure nothing there; what matters at those states is that no output is non-finite, which holds on all 16,384 states. G5 and G6 are subsets of G0. The forward errors use every state of a group on which the model is valid. Gradient errors use up to 64 states per group and corpus (seed 0), 256 per group pooled. Section 6 reports the forward error over all states and the gradient error on G0. With either gradient mode, NablaEngine produced no non-finite output or gradient on any of the 16,384 states. Both modes share one forward pass, which is bitwise identical.

MuJoCo C finite differences are forward differences with step 10^{-6}, chained over the four substeps. Versions: MuJoCo Warp [22] and MJX [21] 3.17.0, Genesis [17] 1.4.5. The open pull request is etaoxing [14], the follow-up fix etaoxing [13].

D Hyperparameters and baselines

Sensitivity. Table 11 varies one setting of the residual recipe at a time. In the final recipe, a 500-update learning-rate warmup instead of 100 trains more slowly (0.82 after 9.3 instead of 7.9 min) but converges slightly better (0.913 against 0.910 at 2.99 h, 3 seeds each).

Network architecture. The actor follows SuperMimic's encoder–decoder design [40]. A motion encoder maps the next 10 reference frames (68 features each) through an MLP with hidden widths 2048–1024–512–256 to a 32-dimensional latent code with a tanh output. A decoder with the same hidden widths maps the latent code, the 10-frame proprioception history (93 features per frame) and the 6-dimensional anchor orientation to the 29 joint offsets. The critic is an MLP with hidden widths 256–128 on privileged inputs (Appendix A).

The design encodes the reference motion into a compact latent space and lets the decoder act as a general motor policy conditioned on that code. This is the structure along which tracking policies are scaled toward foundation models for humanoid control: the latent code is the interface through which motion generators, teleoperation or downstream policies command the tracker, and the same decoder serves all motions [9, 41, 43]. We keep it, rather than a monolithic MLP, so that a policy trained with ∇Mimic can be used the same way.

Baselines.

Table 7. Accuracy on held-out states against MuJoCo C 3.17 in float64. Forward: median relative qvel error. Gradient: median relative VJP error against float64 central finite differences, for position / velocity / action. The truth itself is known to about 10^{-7}.

	Gradient (median relative VJP error)		
Engine	Position	Velocity	Action
NablaEngine	3.6×10^{-5}	1.0×10^{-5}	1.9×10^{-6}
MJX, forward-mode Jacobian	9.6×10^{-5}	2.1×10^{-5}	2.9×10^{-6}
MuJoCo C finite differences (step 10^{-6})	6.7×10^{-4}	1.5×10^{-4}	7.8×10^{-6}
Genesis (velocity only)	—	2.3×10^{-1}	—
MuJoCo Warp, open pull request	0.29	0.34	0.015
MuJoCo Warp, with the follow-up fix (etaoxing/mujoco_warp#1)[a]	5.3×10^{-5}	1.0×10^{-5}	2.1×10^{-6}
	Forward (median relative qvel error)		
NablaEngine	8.5×10^{-7}		
MuJoCo Warp	1.3×10^{-6}		
MJX	2.9×10^{-6}		
Genesis	5.5×10^{-2}		

[a] etaoxing [13].

Table 8. Forward error per group, 10 solver iterations: median / 99th percentile of the relative qvel error, and in parentheses the 99th percentile of the qpos error (rad). States per group: G0 13,480, G4 476, G5 5,645, G6 5,668. MuJoCo C in float64 with 10 iterations has median 0 and 99th percentile at most 1.9×10^{-15} in every group.

	G0	G4	G5	G6
NablaEngine	8.6e-7 / 4.7e-6 (5.4e-7)	4.6e-7 / 6.6e-6 (1.8e-6)	8.4e-7 / 5.5e-6 (6.5e-7)	9.4e-7 / 6.9e-6 (1.0e-6)
MuJoCo Warp	1.3e-6 / 8.8e-6 (1.1e-6)	6.3e-7 / 7.2e-6 (1.5e-6)	1.2e-6 / 6.8e-6 (9.2e-7)	1.3e-6 / 8.8e-6 (9.9e-7)
MuJoCo Warp, open PR	1.3e-6 / 9.3e-6 (1.2e-6)	6.4e-7 / 1.0e-5 (3.4e-6)	1.2e-6 / 8.6e-6 (1.2e-6)	1.4e-6 / 1.1e-5 (1.4e-6)
MJX (float32)	2.7e-6 / 6.3e-4 (7.2e-5)	1.3e-6 / 4.2e-4 (1.1e-4)	2.6e-6 / 5.1e-4 (6.1e-5)	5.5e-6 / 6.6e-4 (7.3e-5)
Genesis (float32)	4.9e-2 / 4.8e-1 (6.0e-2)	5.7e-2 / 5.4e-1 (1.4e-1)	4.9e-2 / 4.8e-1 (6.1e-2)	1.0e-1 / 5.1e-1 (7.1e-2)
Reference: MuJoCo Warp against itself, run to run				
	3.1e-7 / 3.2e-6 (4.3e-7)	3.0e-7 / 2.2e-6 (8.0e-7)	3.4e-7 / 3.3e-6 (3.8e-7)	3.4e-7 / 3.5e-6 (3.9e-7)

- **PPO, BeyondMimic configuration** [41, 64]: on NablaEngine (forward only), 4096 environments × 24 steps per iteration, 5 epochs × 4 minibatches, lr 10^{-3} (adaptive, KL 0.01), clip 0.2, entropy 0.005, $\gamma = 0.99$, $\lambda = 0.95$, MLP 512–256–128, BeyondMimic's adaptive sampling; 6 h budget.
- **PPO, SuperMimic actor and reset sampling**: as above with 8192 environments, SuperMimic's encoder–decoder actor (2048–1024–512–256) and critic (256–128), and return-based reset sampling; 6 h budget.
- **SHAC** [82]: see Appendix F.

All baselines use the same environment, observation and evaluation reward as ∇Mimic. The smooth termination penalty (Eq. (4)) is part of our recipe and does not account for the gaps: removing all of fall handling, the penalty included, changes the return of ∇Mimic by less than 0.01 at matched wall clock (3 seeds, both recipes).

E SuperMimic reproduction and cost estimate

Reproduction. The gradient analysis of Section 4 uses a public reproduction of SuperMimic (https://github.com/CuboiLeo/SuperMimic-Reproduced) with the hyperparameters stated in its paper: windows of 32 imagined steps for 4096 rollouts per update, an update-to-data ratio of 0.4, and a world model that is an MLP with three hidden layers of 2048 units and SiLU activations, predicting the change of the canonicalized state from the state, the action and the simulation context. Its policy observes the anchor orientation of the reference as ours does. We trained one run on an RTX PRO 6000 and evaluated its policy with our protocol (Appendix G); it reaches a full-clip return of 0.789 after 2.88 h. It follows the hyperparameters SuperMimic states and serves one purpose: to provide world models and policies

Table 9. Gradient error per group and input block, 10 solver iterations, 256 states per group: median floored relative VJP error for position / velocity / action. Exact and default denote NablaEngine's gradient modes. "n/c": not comparable (the truth is the exact derivative and has no counterpart of the default mode on clamped actuators); "—": not supported.

	G0	G4	G5	G6
NablaEngine, exact	3.6e-5 / 1.0e-5 / 1.9e-6	4.3e-5 / 1.3e-5 / 2.0e-6	3.8e-5 / 1.1e-5 / 1.8e-6	3.1e-5 / 1.0e-5 / 2.3e-6
NablaEngine, default	same as exact	n/c	same as exact	same as exact
MJX, forward-mode Jacobian	9.6e-5 / 2.1e-5 / 2.9e-6	1.2e-4 / 2.3e-5 / 3.0e-6	9.4e-5 / 2.0e-5 / 3.1e-6	1.0e-4 / 2.5e-5 / 4.8e-6
MuJoCo C, finite differences	6.7e-4 / 1.5e-4 / 7.8e-6	7.1e-4 / 1.8e-4 / 7.2e-6	6.6e-4 / 1.5e-4 / 7.9e-6	5.5e-4 / 1.3e-4 / 7.8e-6
Genesis (velocity only)	— / 2.3e-1 / —	— / 3.4e-1 / —	— / 2.7e-1 / —	— / 3.4e-1 / —
MuJoCo Warp, open PR	2.9e-1 / 3.4e-1 / 1.5e-2	5.0e-1 / 4.8e-1 / 8.9e-2	3.1e-1 / 3.5e-1 / 1.8e-2	3.5e-1 / 4.0e-1 / 1.9e-2
MuJoCo Warp, follow-up fix	5.3e-5 / 1.0e-5 / 2.1e-6	n/c	5.1e-5 / 1.1e-5 / 1.9e-6	4.5e-5 / 1.0e-5 / 2.3e-6

Table 10. ∇Mimic hyperparameters.

Item	Setting
Environments	4096 BPTT training windows per update
Window length	$H = 64$ simulated steps (1.28 s)
Discount, TD	$\gamma = 0.99$, $\lambda = 0.95$
Actor	encoder–decoder MLP 2048–1024–512–256; anchor orientation into the decoder
Critic	MLP 256–128, fit on the pre-transition state
Actions	residual: offset to the reference joint pose
Learning rate	2×10^{-4}, 100-update warmup, cosine decay to 0.1× over 16,000 updates
Action targets	RMS setpoint change 0.03 rad (decays to 0.3× with the LR); 8 regression steps of 16,384 samples; separate Adam, lr 6×10^{-4}
Per-sample clipping	$\kappa = 3\times$ batch median of the action-cotangent norm
Termination penalty	$W = 2$, $dz_0 = 0.08$, $d\theta_0 = 0.10$
Episode ends	reset and continue the window; a fall bootstraps nothing (no survival mask)
Policy averaging	EMA of the actor, decay $\min(0.999, (1+n)/(10+n))$; the averaged policy is evaluated
Window action noise	0.05
Start states	replay of 1000×4096 states from a data collector (action noise 0.1, update-to-data 0.4)
Reset sampling	return-based bins
Physics	NablaEngine, 10 solver iterations, FP32; network matmuls TF32
Time per update	0.37 s on one RTX PRO 6000

at several stages of training on which exact and surrogate gradients can be compared along the same trajectories. It is not a baseline, and its checkpoints at 0.44, 1.14, 1.83 and 2.88 h provide the policies and world models of Section 4.

Cost estimate. SuperMimic states 320,000 training iterations and that the world model and the policy are updated in every iteration. Its BONES-SEED training curve, with a checkpoint every 4,000 iterations, ends at 9.37 h on one H100, which gives about 34,000 iterations per hour, or about 0.1 s per iteration; we assume that LAFAN1 iterations cost the same. Reaching 0.82 within 1.2 h then corresponds to about 41,000 iterations. If one instead counts policy updates at the stated update-to-data ratio of 0.4, as the public reproduction does, these become about 16,000 updates of about 0.26 s each, and ∇Mimic's 3,235 updates to 0.82 are about one fifth of them rather than one thirteenth.

F SHAC details

Both SHAC variants use the same environment, actor, critic, 4096 environments, learning-rate schedule and window noise as ∇Mimic. SHAC's algorithm:

- $H = 32$ windows continued across updates, with no replay starts and no collector;
- the actor's own window gradient, one Adam step per window;
- no per-sample clipping and no termination penalty;
- a TD($\lambda = 0.95$) critic, empirical observation normalization and uniform reset sampling.

SHAC (published guard) clips the actor gradient to global norm 1.0 and stops training at the first update whose norm

Table 11. Sensitivity of the earlier residual recipe (masked, no policy averaging): full-clip normalized return after 1.5 h of training on one RTX PRO 6000, mean ± sd over 3 seeds (4 for $H = 32$). Recipe values in bold. Diverged: training stopped by the divergence guard.

Setting	Value	Return after 1.5 h	Note
Clipping threshold κ	1	0.853 ± 0.008	
	3	**0.869 ± 0.003**	
	10	0.849 ± 0.011	
	30	0.849 ± 0.005	
	none	—	3 of 3 diverged
Clipping location	**backward pass and targets**	**0.869 ± 0.003**	0.82 reached at 0.34–0.43 h
	targets only	0.868 ± 0.001	0.82 reached at 0.52–0.62 h
Gradient rules	**NablaEngine rules**	**0.869 ± 0.003**	
	exact everywhere	0.867 ± 0.002	
Window length H	16	—	3 of 3 diverged (critic value runaway)
	32	0.815–0.841	1 of 4 collapsed (critic value runaway)
	64	**0.869 ± 0.003**	about 14.3k updates
	128	0.839 ± 0.004	about 7.9k updates
Windows per update	1024	0.865 ± 0.002	about 27.0k updates
	2048	0.867 ± 0.004	about 20.3k updates
	4096	**0.869 ± 0.003**	about 14.3k updates
	8192	0.852 ± 0.002	about 8.6k updates

before clipping is not finite or exceeds 10^6, as in the published code. **SHAC (changed guards)** instead clips to global norm 1.0 and then element-wise to 0.1, as SuperMimic does, and skips non-finite updates instead of stopping. Both run on NablaEngine; the published-guard variant also runs on MuJoCo Warp with the engine's gradient semantics. They differ from ∇Mimic in several components at once, so they are baselines, not single-component ablations.

G Evaluation protocol

Each of the 40 LAFAN1 clips is played once from its first reference frame. Playback uses MuJoCo Warp with 10 solver iterations, a deterministic actor (the mean action), no domain randomization and no observation or action noise. The clip's return is the sum of the per-step reward over its steps until the clip ends or the first termination (height error above 0.25 m or tilt above 0.8). The per-step reward is the six tracking terms minus the action-rate and joint-limit penalties. The sum is divided by the clip's maximum attainable tracking reward, and the score is the unweighted mean over the 40 clips. A clip with a fall therefore keeps the reward earned before the fall and scores zero afterwards. This matches the normalized return defined by Liao et al. [40]. Failed clips are clips with a fall.

Every checkpoint of the first training hour is evaluated (every 250 updates), then every 1000 updates, plus the last. Training wall clock excludes start-up compilation and all evaluation. Times to a score are first crossings of the 3-seed mean curve, linearly interpolated between evaluations.

H Robustness

Table 12 evaluates the final policies under the training-time randomization and observation noise, under external pushes, and from random start frames, on MuJoCo Warp with 10 solver iterations; its nominal condition reproduces the scores of Table 4 (earlier-recipe row for ∇Mimic).

I Additional LAFAN1 results

Figure 7 shows every LAFAN1 run against wall clock, including the absolute-action variant of ∇Mimic, SHAC and both PPO baselines, with the numbers stated by Liao et al. [40]. Table 13 lists all runs, including the H100 and MuJoCo Warp runs and the 6 h PPO budgets. Table 14 gives the cost to reach 0.82, Tables 15 and 16 the full ablation results on the absolute- and residual-action recipes, and Table 17 the training-loop cost per policy update.

Table 12. Robustness on LAFAN1 (final checkpoints; ∇Mimic in the earlier recipe, masked and without policy averaging, and SHAC with changed guards: mean of 3 seeds; PPO, BeyondMimic configuration: one seed). Return: normalized return; success: share of clip runs without a fall; MPJPE: mean position error of the 14 tracked bodies with the reference re-anchored to the robot in horizontal position and yaw, in cm. Randomization and noise as in training; pushes change the base's horizontal velocity by the stated amount every 1 to 3 s in a random direction; 10 s windows start at random frames. Four repeats per clip, eight windows per clip. Best return and success per row in bold.

	∇Mimic			SHAC (changed guards)			PPO		
Condition	Return	Success	MPJPE	Return	Success	MPJPE	Return	Success	MPJPE
Nominal	**0.872**	**0.992**	2.47	0.820	0.975	3.41	0.695	0.775	3.95
Randomization	**0.864**	**0.990**	2.53	0.805	0.956	3.45	0.688	0.775	3.98
Observation noise	**0.868**	**1.000**	2.54	0.803	0.965	3.50	0.711	0.825	4.04
Randomization + noise	**0.857**	**0.992**	2.61	0.793	0.946	3.56	0.677	0.762	4.03
Pushes 0.5 m/s	**0.749**	**0.840**	3.37	0.693	0.779	3.96	0.637	0.719	4.33
Pushes 1 m/s	**0.478**	0.377	4.36	0.407	0.237	4.56	0.467	**0.444**	4.82
Rand. + noise, 10 s windows	**0.899**	**0.999**	2.79	0.836	0.989	3.80	0.793	0.991	4.32
Pushes 1 m/s, 10 s windows	**0.750**	**0.969**	4.28	0.707	0.924	4.67	0.678	0.922	5.07

Table 13. LAFAN1: full-clip normalized return at equal training wall clock

Method	Operator	Training hardware	Seeds n	Budget (h)	Return @0.5 h	Return @1 h	Return @2 h	Return @3 h	Return @6 h	Time to 0.82 (h)	Per-seed time to 0.82 (h)	Final return (h)	Failed clips at end (of 40)
∇Mimic (NablaEngine)	NablaEngine	RTX PRO 6000 (Blackwell)	3	3	0.866 ± 0.006	0.885 ± 0.011	0.899 ± 0.016	0.910 ± 0.004	—	0.13	0.12 ± 0.01	0.911 ± 0.002 (3.07 h)	0.0 ± 0.0
∇Mimic, absolute actions (NablaEngine)	NablaEngine	RTX PRO 6000 (Blackwell)	3	3	0.821 ± 0.007	0.844 ± 0.004	0.861 ± 0.001	0.864 ± 0.001	—	0.50	0.46 ± 0.09	0.865 ± 0.003 (3.14 h)	0.0 ± 0.0
∇Mimic, absolute actions (NablaEngine, H100)‡	NablaEngine	H100	1	3	0.816	0.848	0.861	0.862	—	0.41	0.41	0.858 (3.10 h)	0
∇Mimic, absolute actions (MuJoCo Warp)	MuJoCo Warp	RTX PRO 6000 (Blackwell)	3	3	0.636 ± 0.024	0.742 ± 0.043	0.797 ± 0.014	0.826 ± 0.009	—	2.82	2.60 ± 0.16 (2 of 3 seeds reach it)	0.825 ± 0.011 (3.00 h)	0.7 ± 0.6
SHAC (Xu et al., 2022; published guard), our scale (NablaEngine)	NablaEngine	RTX PRO 6000 (Blackwell)	3	3	—	—	—	—	—	not reached (best 0.373)	—	0.373 ± 0.093 (0.06 h)	23.0 ± 3.6
SHAC (published guard), our scale (MuJoCo Warp)‡	MuJoCo Warp	RTX PRO 6000 (Blackwell)	1	3	—	—	—	—	—	not reached (best 0.484)	—	0.484 (0.36 h)	17
SHAC (changed guards: non-finite updates skipped, element clip 0.1; NablaEngine)	NablaEngine	RTX PRO 6000 (Blackwell)	3	3	0.724 ± 0.019	0.790 ± 0.019	0.807 ± 0.008	0.816 ± 0.005	—	mean curve does not reach it	2.64 ± 0.34	0.809 ± 0.018 (3.13 h)	2.0 ± 2.0
SHAC (changed guards; MuJoCo Warp)‡	MuJoCo Warp	RTX PRO 6000 (Blackwell)	1	3	0.478	0.572	0.680	0.756	—	not reached (best 0.760)	—	0.760 (3.13 h)	4
PPO, BeyondMimic (NablaEngine)‡	NablaEngine (forward only)	RTX PRO 6000 (Blackwell)	1	6	0.465	0.529	0.641	0.638	0.695	not reached (best 0.751)	—	0.695 (6.00 h)	8
PPO, SuperMimic actor and reset sampling (NablaEngine)‡	NablaEngine (forward only)	RTX PRO 6000 (Blackwell)	1	6	0.428	0.489	0.586	0.658	0.692	not reached (best 0.692)	—	0.692 (6.00 h)	3
SuperMimic (paper, 1x H100)[a]	learned world model	H100 (1 GPU)	—	—	—	—	—	—	—	within 1.2 (stated)	—	—	not reported
PPO-Sim (SuperMimic paper)[b]	simulator (paper setting)	H100 (1 / 4 / 8 GPUs)	—	18	—	—	—	—	—	not reached in 18 h (stated)	—	0.60 / 0.70 / 0.72 (18 h; 1 / 4 / 8 GPUs)	not reported
Appendix rows (earlier recipe)													
∇Mimic, absolute actions, earlier recipe (no anchor-to-decoder)‡	NablaEngine	RTX PRO 6000 (Blackwell)	1	3	0.797	0.842	0.859	0.863	—	0.47	0.47	0.865 (3.13 h)	0

‡ single seed (no seed variance; a seed-noise reference is given in the notes)
[a] stated: reaches 0.82 within 1.2 h on one H100
[b] stated: does not reach 0.82 within 18 h; final 0.60 / 0.70 / 0.72 on 1 / 4 / 8 GPUs
Failed clips = clips with a fall (training termination test) under the nominal evaluation.
Falls, ∇Mimic (NablaEngine) (3 seeds): no seed fails a clip at any evaluated checkpoint from update 4,750 (0.51 h) on (the last fall is at update 4,500: 1 failed clip on 1 seed); update 1,250 (0.14 h) is the first checkpoint with 0 of 40 failed clips on all seeds.
Falls, ∇Mimic, absolute actions (NablaEngine) (3 seeds): no seed fails a clip at any evaluated checkpoint from update 27,000 (2.88 h) on (the last fall is at update 26,000: 1 failed clip on 1 seed); update 7,000 (0.74 h) is the first checkpoint with 0 of 40 failed clips on all seeds.
'(n=k of N)' in a cell: only k of the row's N seeds have a value there (a seed cut at a divergence or not evaluated up to that hour yet); final and crossing columns use only seeds evaluated up to the budget.
Training wall clock: training time only for our runs, SHAC and PPO (start-up compilation and all evaluation excluded); SuperMimic and PPO-Sim hours as stated in the SuperMimic paper.
SuperMimic's stated numbers are on H100 GPUs; every run of ours and of the baselines we trained is on one RTX PRO 6000 (Blackwell) unless the hardware column says otherwise.
Evaluation GPU is A40 or RTX PRO 6000; the evaluation is deterministic, small cross-GPU floating-point differences are possible.
Seed-noise references (different protocols; neither is a test of a single-seed row): 1-h screen (5 solver iterations), MuJoCo Warp, RTX PRO 6000 (Blackwell): sd 0.0138 of the full return over n=3 seeds; nominal evaluation on NablaEngine runs at matched updates: sd 0.0075 of the full return over n=3 seeds.

Table 14. Cost to reach SuperMimic's LAFAN1 score of 0.82

Method	Hardware	n	Time to 0.82 (h)	Policy updates there	Differentiable window steps	Forward-only simulator steps	Total simulator transitions	Per-seed time to 0.82 (h)
∇Mimic (NablaEngine)	RTX PRO 6000 (Blackwell)	3	0.13	1,151 updates	3.02e8	1.18e7	3.13e8	0.12 ± 0.01
∇Mimic, absolute actions (NablaEngine)	RTX PRO 6000 (Blackwell)	3	0.50	4,650 updates	1.22e9	4.76e7	1.27e9	0.46 ± 0.09
∇Mimic, absolute actions (NablaEngine, H100)‡	H100	1	0.41	3,719 updates	9.75e8	3.81e7	1.01e9	0.41
∇Mimic, absolute actions (MuJoCo Warp)	RTX PRO 6000 (Blackwell)	3	2.82	5,651 updates	1.48e9	5.79e7	1.54e9	2.60 ± 0.16 (2 of 3 seeds reach it)
SHAC (Xu et al., 2022; published guard), our scale (NablaEngine)	RTX PRO 6000 (Blackwell)	3	not reached (best 0.373 at 0.06 h)	—	—	—	—	—
SHAC (published guard), our scale (MuJoCo Warp)‡	RTX PRO 6000 (Blackwell)	1	not reached (best 0.484 at 0.36 h)	—	—	—	—	—
PPO, BeyondMimic (NablaEngine)‡	RTX PRO 6000 (Blackwell)	1	not reached (best 0.751 at 5.75 h)	—	—	—	—	—
PPO, SuperMimic actor and reset sampling (NablaEngine)‡	RTX PRO 6000 (Blackwell)	1	not reached (best 0.692 at 6.00 h)	—	—	—	—	—
SuperMimic (paper, 1x H100)[a]	H100 (1 GPU)	—	within 1.2 (stated)	not stated	—	not stated	not stated	—

‡ single seed (no seed variance; a seed-noise reference is given in the notes)
[a] stated: reaches 0.82 within 1.2 h on one H100
Per-update step counts: ∇Mimic = 4096 x 64 = 262,144 differentiable window steps + 4096 / 0.4 = 10,240 forward-only collector steps; SHAC = 4096 x 32, no collector; PPO = 4096 x 24 (BeyondMimic) or 8192 x 24 (SuperMimic actor and reset sampling) forward steps per iteration.
Hours are training time only for our runs (start-up compilation and evaluation excluded); SuperMimic's hours are as stated in its paper.

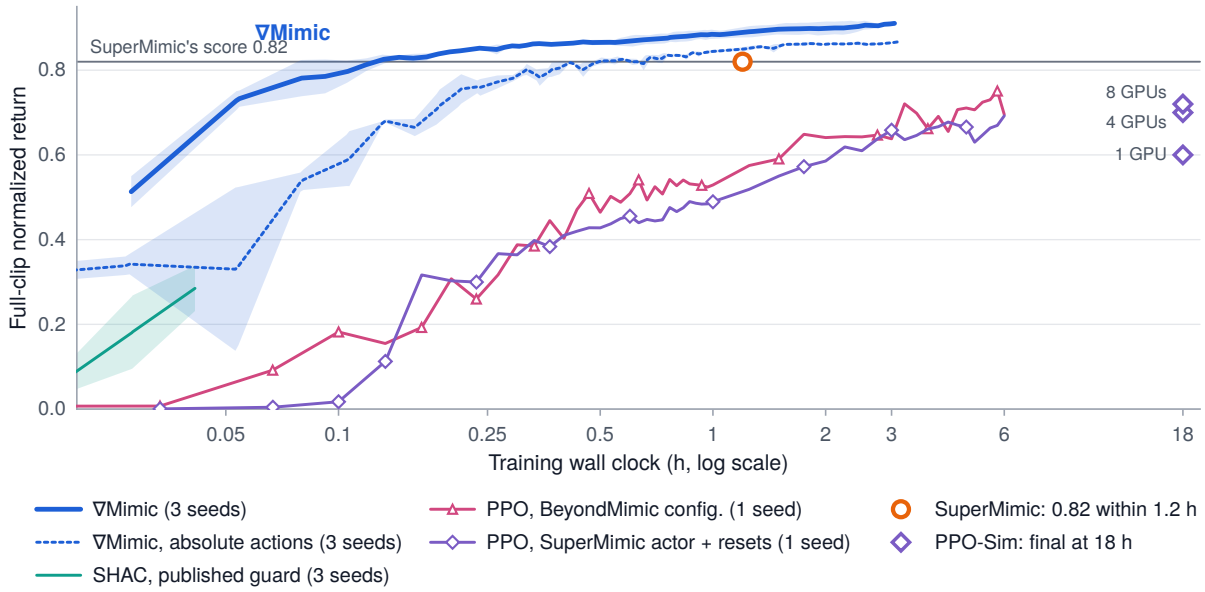


Figure 7. All LAFAN1 runs: full-clip normalized return against training wall clock. Lines are seed means with a ±1 sd band when at least two seeds are evaluated. Every run trains on NablaEngine on one RTX PRO 6000. SuperMimic and PPO-Sim are the numbers stated in Liao et al. [40, §4.1].

Table 15. Component ablations of ∇Mimic on the absolute-action recipe (LAFAN1, 3 seeds, ~1.5 h)

Change to the final recipe	Component origin	Seeds	Matched at	Return: control	Return: variant	Δ return	Failed clips (of 40): control	Failed clips (of 40): variant
No PODS action targets (one Adam step per window)	ours	3	1.64 h (control checkpoints 1.14-1.63 h)	0.855 ± 0.001	0.806 ± 0.011	-0.049	0.1 ± 0.1	1.9 ± 0.8
No per-sample gradient clipping[a]	ours	2 of 3 (s0 diverged at 0.54 h, update 5,060)	1.65 h (control checkpoints 1.18-1.66 h)	0.855 ± 0.001	0.562 ± 0.079 (n=2 of 3)	-0.293	0.1 ± 0.1	10.9 ± 4.7 (n=2 of 3)
Residual actions instead of absolute setpoints[b]	action parameterization	3	1.64 h (control checkpoints 1.14-1.63 h)	0.855 ± 0.001	0.864 ± 0.001	+0.009	0.1 ± 0.1	0.0 ± 0.0
Uniform resets instead of return-based reset sampling	inherited: SuperMimic (reset sampling follows BeyondMimic)	3	1.65 h (control checkpoints 1.18-1.66 h)	0.855 ± 0.001	0.861 ± 0.003	+0.005	0.1 ± 0.1	0.1 ± 0.1
No replay starts (windows continue from the previous window's end state)	inherited: SuperMimic (replay-buffer starts)	3	1.62 h (control checkpoints 1.14-1.63 h)	0.855 ± 0.001	0.874 ± 0.002	+0.019	0.1 ± 0.1	0.0 ± 0.0
H=32 windows instead of H=64	ours	3	1.64 h (control checkpoints 1.14-1.63 h)	0.855 ± 0.001	0.818 ± 0.019	-0.036	0.1 ± 0.1	0.2 ± 0.2
H=32 windows instead of H=64, at matched updates	ours	3	update 26,000 (control updates 22,000-26,000, 2.28-2.80 h)	0.862 ± 0.002	0.818 ± 0.019	-0.043	0.1 ± 0.1	0.2 ± 0.2

[a] additional seeds outside the 3-seed arm (not averaged): s3 diverged at 0.40 h, s4 completed
[b] additional seeds outside the 3-seed arm (not averaged): s3 completed, s4 completed
Component origin: 'ours' = introduced by this work; 'inherited' = taken from SuperMimic (whose return-based reset sampling follows BeyondMimic); residual actions change the action parameterization rather than remove a component.
Seeds that diverged (non-finite loss) or collapsed (critic value runaway with finite numbers: critic loss above 1e3, return falling to near zero) are listed in the Seeds column and left out of the mean; the variant cell then says '(n=k of 3)'.
Seed-noise references (different protocols): 1-h screen (5 solver iterations), MuJoCo Warp, RTX PRO 6000 (Blackwell): sd 0.0138 of the full return over n=3 seeds; nominal evaluation on NablaEngine runs at matched updates: sd 0.0075 of the full return over n=3 seeds.

Table 16. Component ablations of ∇Mimic on the residual-action recipe (LAFAN1, 3 seeds, ~1.5 h)

Change to the final recipe	Component origin	Seeds	Matched at	Return: control	Return: variant	Δ return	Failed clips (of 40): control	Failed clips (of 40): variant
No PODS action targets (one Adam step per window)	ours	3	1.63 h (control checkpoints 1.14-1.58 h)	0.866 ± 0.002	0.822 ± 0.012	-0.044	0.1 ± 0.2	1.3 ± 0.8
No per-sample gradient clipping	ours	0 of 3 (s0 diverged at 0.59 h, update 5,510; s1 diverged at 0.31 h, update 2,930; s2 diverged at 0.37 h, update 3,500)						
Uniform resets instead of return-based reset sampling[a]	inherited: SuperMimic (reset sampling follows BeyondMimic)	3	1.64 h (control checkpoints 1.14-1.58 h)	0.866 ± 0.002	0.811 ± 0.065	-0.055	0.1 ± 0.2	2.8 ± 3.0
No replay starts (windows continue from the previous window's end state)	inherited: SuperMimic (replay-buffer starts)	3	1.64 h (control checkpoints 1.14-1.58 h)	0.866 ± 0.002	0.884 ± 0.006	+0.018	0.1 ± 0.2	0.0 ± 0.0
H=32 windows instead of H=64[b]	ours	3 of 4 (s2 collapsed (critic value runaway) at 1.10 h, update 17,080)	1.65 h (control checkpoints 1.15-1.66 h)	0.867 ± 0.002	0.824 ± 0.010 (n=3 of 4)	-0.042	0.1 ± 0.2	0.4 ± 0.0 (n=3 of 4)
H=32 windows instead of H=64, at matched updates[c]	ours	3 of 4 (s2 collapsed (critic value runaway) at 1.10 h, update 17,080)	update 25,583 (control updates 21,000-25,000, 2.18-2.63 h)	0.873 ± 0.002	0.824 ± 0.010 (n=3 of 4)	-0.048	0.0 ± 0.0	0.4 ± 0.0 (n=3 of 4)

[a] a transient collapse falls inside the averaging window; the last checkpoints of all three seeds are 0.86-0.87
[b] additional seeds outside the 3-seed arm (not averaged): s3 completed
[c] additional seeds outside the 3-seed arm (not averaged): s3 completed
Component origin: 'ours' = introduced by this work; 'inherited' = taken from SuperMimic (whose return-based reset sampling follows BeyondMimic); residual actions change the action parameterization rather than remove a component.
Seeds that diverged (non-finite loss) or collapsed (critic value runaway with finite numbers: critic loss above 1e3, return falling to near zero) are listed in the Seeds column and left out of the mean; the variant cell then says '(n=k of 3)'.
Seed-noise references (different protocols): 1-h screen (5 solver iterations), MuJoCo Warp, RTX PRO 6000 (Blackwell): sd 0.0138 of the full return over n=3 seeds; nominal evaluation on NablaEngine runs at matched updates: sd 0.0075 of the full return over n=3 seeds.

Table 17. Training-loop cost per policy update

Method	Operator	Hardware	Seconds per policy update (median)	Policy updates per hour (wall-clock slope)	Differentiable steps per update	Forward-only steps per update	Simulated steps per second	Engine speed-up vs MuJoCo Warp
∇Mimic[a]	NablaEngine	RTX PRO 6000 (Blackwell)	0.374	9,491	262,144	10,240	729k	4.4x
∇Mimic, absolute actions[b]	NablaEngine	RTX PRO 6000 (Blackwell)	0.373	9,383	262,144	10,240	729k	4.4x
∇Mimic, absolute actions, hoisted encoder (timing run)[c]	NablaEngine	RTX PRO 6000 (Blackwell)	0.369	—	262,144	10,240	738k	4.4x
∇Mimic, absolute actions	MuJoCo Warp	RTX PRO 6000 (Blackwell)	1.633	1,966	262,144	10,240	167k	—
∇Mimic, absolute actions (H100)	NablaEngine	H100	0.382	9,275	262,144	10,240	712k	5.9x[§]
∇Mimic, absolute actions (H100)[d]	MuJoCo Warp	H100	2.258	—	262,144	10,240	121k	—
SHAC, our scale	NablaEngine	RTX PRO 6000 (Blackwell)	0.196	18,365	131,072	0	669k	4.3x
SHAC, our scale	MuJoCo Warp	RTX PRO 6000 (Blackwell)	0.839	3,455	131,072	0	156k	—
PPO, BeyondMimic[e]	NablaEngine (forward only)	RTX PRO 6000 (Blackwell)	0.126	28,240	0	98,304	781k	—
PPO, SuperMimic actor and reset sampling[f]	NablaEngine (forward only)	RTX PRO 6000 (Blackwell)	0.549	6,542	0	196,608	358k	—

[a] residual actions
[b] final recipe
[c] speed reference for the final recipe's encoder hoisting (bitwise-equivalent up to matmul rounding); 70-update timing run
[d] O64 recipe without anchor-to-decoder: a different recipe from the final engine runs; 30-update timing run
[e] per PPO iteration
[f] per PPO iteration
Simulated steps per second count differentiable window steps (forward + adjoint) and forward-only steps together.
The engine runs of the paper pin one engine release; earlier timing runs used the engine build of their day, so per-update times across rows mix engine versions with recipe changes.
[§] on a speed-up: the MuJoCo Warp timing comes from a run of a different recipe (named in its notes), so the speed-up mixes the operator with recipe changes; the matched-recipe figure needs the final MuJoCo Warp run.