Swapping the Physics Engine Under a Legged-Robot RL Task: A Comparative Analysis of PhysX and Newton in Isaac Lab 3.0

Yusuf Guenena

***Abstract*: Isaac Lab 3.0 lets the same reinforcement-learning task run on either NVIDIA PhysX or the Newton engine with its MuJoCo-Warp solver, selected by a single configuration preset. That makes a backend comparison look trivial: flip the preset, rerun, compare. We carry out that comparison on the stock Unitree Go2 flat-terrain velocity-tracking task and find that the flip changes more than the engine. We report four measurements. (i) Compute: on one consumer Blackwell-architecture GPU, Newton steps the task $1.93\times$ to $2.23\times$ faster than PhysX across 256 to 4096 parallel environments (10 timed repeats per point), trains $1.64\times$ faster end to end (IQM over 10 seeds), and uses $5.3\times$ less per-process GPU memory during training, a ratio we attribute to each backend's stock buffer sizing rather than to the engines themselves. (ii) Dynamics: under an identical open-loop action tape, each backend replays bit-identically, while the two backends diverge by 0.41 m in base position and 25° in orientation within 6 s, and already disagree by 0.118 rad in joint position while the robot stands still. (iii) Learning: at 300 PPO iterations PhysX reaches an IQM training reward of 30.36 [28.71, 31.09] against 19.48 [16.77, 22.36] for Newton. We show that this gap is not a backend-only result: a recursive audit of the resolved configurations finds that the stock PhysX preset also enables startup base-mass randomization and a fixed robot friction override that the Newton preset omits. (iv) Transfer: without remapping, every policy evaluated on the other backend fails (success 0.00 and 0.02 from 0.94 and 0.97). That collapse is an artifact. The two backends enumerate the robot's joints in different orders, so the same observation and action vectors address different joints. With joint semantics enforced, a 3-policy pilot per direction retains most of its performance across backends (PhysX to Newton: mean return 28.3 to 27.1; Newton to PhysX: 29.9 to 25.1). We release the configuration audit and the runtime joint-contract check as preflight gates, and state which of our numbers survive them.**

***Index Terms*: Physics simulation, reinforcement learning, legged robots, benchmarking, reproducibility, Isaac Lab, MuJoCo, GPU simulation.**

I. Introduction

GPU-parallel simulation made it routine to train a quadruped locomotion policy in minutes [1], [2]. Frameworks built on that capability, Orbit [3] and its successor Isaac Lab [4], have until recently tied the learning stack to a single physics engine, PhysX. Isaac Lab 3.0 changes that. It ships an experimental second backend, Newton [5], whose default solver is MuJoCo-Warp [6], a GPU reimplementation of MuJoCo [7]. A task can declare both backends side by side, and the user picks one with a Hydra override: `presets=physx` or `presets=newton`.

For a practitioner, the questions are concrete. Is Newton faster on my task? Does a policy trained on one backend still work on the other? Can I switch a validated pipeline without re-validating it? The mechanism invites a fast answer: run both presets and compare. This report is a careful version of that comparison on one stock task, and most of what we learned concerns what the fast answer gets wrong.

Contributions.

1) A compute characterization of both backends on the stock Go2 flat-terrain task: pure-step throughput and per-process memory across four environment counts with 10 timed repeats each, plus end-to-end training throughput over 10 seeds per backend (Section V-A).
2) An open-loop dynamics-equivalence probe with a same-backend determinism control, showing the backends are individually deterministic and mutually non-equivalent on this asset, including a systematic disagreement during a static hold (Section V-B).
3) Two validity traps that the preset mechanism hides, each with a released check. The stock presets differ in non-physics randomization events (Section V-C), and the backends enumerate joints in different orders, which turns a working policy into a failing one when it moves across backends without remapping (Section V-D).
4) An explicit accounting of which results survive those checks (Table VI). The throughput, memory, and dynamics results do. The learning-curve comparison does not and is reported as a confounded artifact. The remapped transfer result survives but is a 3-policy pilot.

II. Background and Related Work

GPU-parallel robot learning. Isaac Gym [2] moved both physics and policy data onto the GPU and removed the host round trip that limited earlier simulators. Rudin *et al.* [1] showed that thousands of parallel quadrupeds trained with PPO [8] can learn to walk in minutes, and the Go2 velocity task we use descends from that recipe. Brax [9] took a similar route in JAX. Orbit [3] and Isaac Lab [4] wrapped PhysX-based simulation in a modular task and learning framework.

Multiple engines behind one interface. MuJoCo [7] is the reference engine for much of continuous-control RL. MuJoCo-Warp [6] ports it to NVIDIA Warp, and Newton [5] is a

Y. Guenena is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, USA (e-mail: yusuf.a.guenena@gmail.com). Code, raw artifacts, and figure scripts: https://github.com/yusufdxb/physx-newton-bench

Technical report, September 2026. Not peer reviewed and not submitted to any venue.

TABLE I
SOFTWARE ENVIRONMENT

Component	Version
Isaac Lab	3.0.0 (commit `a4a7602f`)
Newton / Warp / mujoco-warp	1.0.0 / 1.12.0 / 3.5.0.2
rsl-rl-lib / PyTorch	5.0.1 / 2.10.0+cu128
NVIDIA driver / CUDA	570.211.01 / 12.8
GPU	one consumer Blackwell-class GPU

Warp-based physics framework that exposes MuJoCo-Warp as a solver. Isaac Lab 3.0 integrates Newton as an alternative to PhysX behind the same task interface.

Comparing simulators. Erez *et al.* [10] compared Bullet, Havok, MuJoCo, ODE and PhysX on speed-accuracy trade-offs for model-based robotics, and Collins *et al.* [11] surveyed physics simulators for robotic applications. Both predate GPU-batched RL, where the quantities of interest are throughput at thousands of environments, memory headroom, and whether a policy learned in one engine behaves the same in another. Engine-to-engine disagreement is a sim-to-sim version of the sim-to-real gap [12], which domain randomization [13] tries to cover. That is why a mismatch in randomization events between two arms of a comparison matters: it changes the training distribution, not only the engine.

Statistics for RL comparisons. Henderson *et al.* [14] showed how seed count and reporting choices can reverse RL conclusions. Agarwal *et al.* [15] recommend interquartile mean (IQM) aggregates with stratified bootstrap confidence intervals in the few-run regime. We follow that recommendation and also report plain means and standard deviations.

III. EXPERIMENTAL SETUP

Task. `Isaac-Velocity-Flat-Unitree-Go2-v0`: a 12-joint quadruped tracks commanded planar velocities on flat ground. The simulation step is $\Delta t = 5\,\text{ms}$ with an action decimation of 4 (50 Hz policy). Actions are joint-position offsets around the default stance, converted to torques by an explicit DC-motor PD model (stiffness 25, damping 0.5). Episodes last 1000 policy steps.

Backends. The task declares a `PhysicsCfg` preset with two arms. The PhysX arm uses the stock `PhysxCfg` with a large preallocated rigid-patch pool (`gpu_max_rigid_patch_count` $= 10 \cdot 2^{15}$). The Newton arm uses `MJWarpSolverCfg` with `njmax`$=65$, `nconmax`$=35$, a pyramidal friction cone, the `implicitfast` integrator, and one substep. We deliberately did not retune either arm. This is a stock-versus-stock comparison, which is what a user flipping the preset gets.

Software and hardware. Table I. All runs used one workstation with a single consumer Blackwell-architecture NVIDIA GPU. The full package freeze, Isaac Lab commit, and driver capture are in the repository.

Measurement discipline. Isaac Sim's Kit runtime captures the process standard output, so anything printed after initialization is silently lost. Every number in this report is therefore read from an on-disk artifact (TensorBoard event extracts, JSON, CSV, NumPy archives) and not from console output. GPU memory is measured per process through `nvidia-smi --query-compute-apps`, not for the whole device.

IV. METHODS

A. Pure-step throughput probe

For each backend and each $N \in \{256, 1024, 2048, 4096\}$ environments we step the task with zero actions. Because actions are offsets from the default stance, zero action holds the robot standing under PD control, with about four foot contacts per environment, well below the Newton contact budget. After 10 warm-up steps we time 10 independent blocks of 100 steps and report mean $\pm$ standard deviation of environment-steps per second.

B. Training

We train PPO with the stock rsl_rl configuration for 300 iterations at 2048 environments, 10 seeds per backend (seeds 0 to 9), 20 runs in total. The resolved `agent.yaml` files are identical apart from the run name. We record the training reward, throughput, wall-clock, and per-process memory peak for every iteration. Aggregates are IQM with 95% stratified bootstrap intervals [15].

C. Open-loop dynamics probe

With one environment, all randomization disabled, and base-contact termination disabled, both backends start from the same initial state and replay the same action tape: a 1 s zero-action hold followed by 5 s of per-joint sinusoids (300 policy steps). Actions are mapped to a canonical joint order for each backend. We record each backend twice to measure same-backend repeatability, then compare base position, base orientation, joint positions, and foot contact forces across backends. Link masses agree to $4.8 \times 10^{-7}\,\text{kg}$, with a total of 15.019 kg in both.

D. Semantic configuration audit

Both arms dump their fully resolved environment and agent configurations. A recursive diff classifies every differing field as *harmless representation* (for example log paths, or a disabled event printed as `null` in one dump and omitted in the other), *required backend-specific* (the physics manager, solver, contact buffers, and contact-sensor class), or *uncontrolled confound* (anything else that changes the learning problem). A comparison can support a backend-only claim only if the confound set is empty.

E. Cross-backend fixed-checkpoint evaluation

We take the final checkpoint of each trained policy and evaluate it, with no further training, on both backends: 64 environments, 64 episodes, fixed seed. Observation noise and every non-shared startup, push, and external-force event are disabled in both evaluations, so the two evaluation environments differ only in backend. Success means the episode reaches the time limit without a base-contact termination. A zero-action controller serves as a stability baseline: under this

TABLE II
PURE-STEP THROUGHPUT (ENV-STEPS/S, MEAN ± S.D., 10 REPEATS) AND PER-PROCESS MEMORY

N	PhysX	Newton	ratio	MiB (PhysX / Newton)	
256	36,877 ± 2,829	82,281 ± 10,609	2.23×	2451	304
1024	136,708 ± 9,811	282,463 ± 33,819	2.07×	2723	368
2048	247,140 ± 14,538	477,471 ± 49,831	1.93×	3019	434
4096	371,229 ± 18,567	732,796 ± 62,073	1.97×	3553	566

TABLE III
TRAINING, 2048 ENVIRONMENTS, 300 ITERATIONS, 10 SEEDS PER BACKEND. REWARD ROWS ARE CONFOUNDED (SECTION V-C).

Metric	PhysX	Newton
Train frames/s, IQM	147.2k	241.1k
95% CI	[145.9k, 148.5k]	[240.5k, 242.4k]
Train frames/s, mean ± s.d.	147.2k ± 2.2k	241.6k ± 2.2k
Wall-clock per run (s)	115.7 ± 1.8	79.3 ± 0.4
Peak process memory (MiB)	3247	617
Final reward†, IQM [95% CI]	30.36 [28.71, 31.09]	19.48 [16.77, 22.36]
Final reward†, mean ± s.d.	29.78 ± 2.65	19.63 ± 3.45

†Not a backend-only comparison; see Section V-C and Table IV.

protocol it completes all 64 episodes on both backends (mean return 9.85 on PhysX, 9.27 on Newton).

A runtime contract check reads, from the live environment, the robot's joint and body order and the joint names behind each joint-indexed observation and action term, and compares them across backends. The *remapped* evaluations fix the policy-facing joint order to the order of the checkpoint's training backend.

V. RESULTS

A. Compute

Newton steps the task roughly twice as fast at every environment count (Fig. 1a, Table II). Two details of the probe matter for anyone reusing these numbers. First, all environments start together, so the synchronized 1000-step episode reset falls inside the last timed block for both backends and lowers that block by about 15 to 25%. We keep it in the statistics because it affects both arms equally. Second, an earlier single-shot probe at 2048 environments read $2.23\times$. With 10 repeats the mean is $1.93\times$, so single probes of this kind are noisy at roughly the $\pm 10\%$ level.

End to end, the advantage shrinks (Table III). Newton trains at 241,070 frames/s against 147,155 (IQM, $1.64\times$), and a 300-iteration run takes 79.3 s against 115.7 s ($1.46\times$). The PPO update, logging, and rollout bookkeeping cost the same on both backends, so they dilute the stepping advantage. A user whose workload is dominated by simulation (data generation, evaluation sweeps) should expect a gain near $2\times$, and a user training small policies should expect about $1.6\times$.

Memory differs most (Fig. 1b). Newton's per-process footprint grows from 304 to 566 MiB while the environment count rises sixteenfold. PhysX's grows from 2451 to 3553 MiB. At the training peak the ratio is $5.3\times$ (3247 against 617 MiB). This is a stock-configuration fact, not an engine property. PhysX preallocates a large GPU patch pool by default, while the Newton arm sizes its contact and constraint buffers tightly per environment. A comparison with both arms tuned to the task would narrow the gap. The stock ratio is still what a user switching presets will see, and on memory-limited GPUs it translates directly into environment-count headroom.

B. Dynamics equivalence

Each backend replayed the tape bit-identically: across two recordings per backend, same-backend divergence was exactly 0.0 m and 0.0 rad in every channel. The cross-backend gap is therefore not noise from random seeds or from the harness. It is large (Fig. 2). Base position separates by 0.41 m and orientation by 25° over 6 s, crossing 5 cm at 1.72 s and 5° at 1.86 s. Mean joint-position RMSE is 0.176 rad, with a maximum of 0.269 rad at the rear-left calf. Total foot-contact force correlates only weakly across backends ($r = 0.39$).

The late-time numbers mix a systematic difference with the sensitivity to initial conditions that any contact-rich system shows, so they overstate the systematic part. The cleaner signal is the zero-action hold. Before any sinusoid, while both robots stand under the same PD targets, joint positions already differ by up to 0.118 rad. The two engines settle the same standing robot into different equilibria.

Gain representation. The backends also report different joint-drive gains for the same asset: PhysX exposes physics-level stiffness and damping of 0/0, Newton exposes 25/0.5. The Go2 actuator is an explicit DC-motor model, which computes the PD torque in Python and applies it as an effort, so a nonzero physics-level gain on top of it would amount to double PD control. We traced the Newton import path and confirmed on the live model that MuJoCo-Warp builds no actuators (`mjw_model.nu` $= 0$) and that all 12 joints are in effort mode. The reported gains are never used. Our comparison is single PD against single PD. The write path that produces the echoed gains becomes a genuine double-PD defect for assets whose USD files author nonzero drive gains, and it was reported upstream as Isaac Lab issue #5806 [16], where we posted this trace. The issue was closed in August 2026, after our runs; we have not re-measured on a later Isaac Lab commit.

C. Learning: a confounded artifact

Fig. 3 shows the reward curves from the 20 training runs. Measured per iteration, the arms track each other for the first hundred or so iterations and then separate. The IQM bootstrap intervals are disjoint from iteration 144 onward, and PhysX ends at 30.36 against 19.48. Measured per second, Newton's IQM is higher at 87% of the evaluated wall-clock points, because its iterations are cheaper. At the end of Newton's 60.6 s budget it leads 19.2 [16.9, 21.8] to 16.8 [15.4, 18.3], with overlapping intervals.

A reader could take this as "PhysX learns a better policy per sample, and Newton gets there faster per second." The configuration audit (Table IV) shows that the data do not support either half of that reading. In the stock task, the PhysX events preset is defined as the Newton events preset plus a

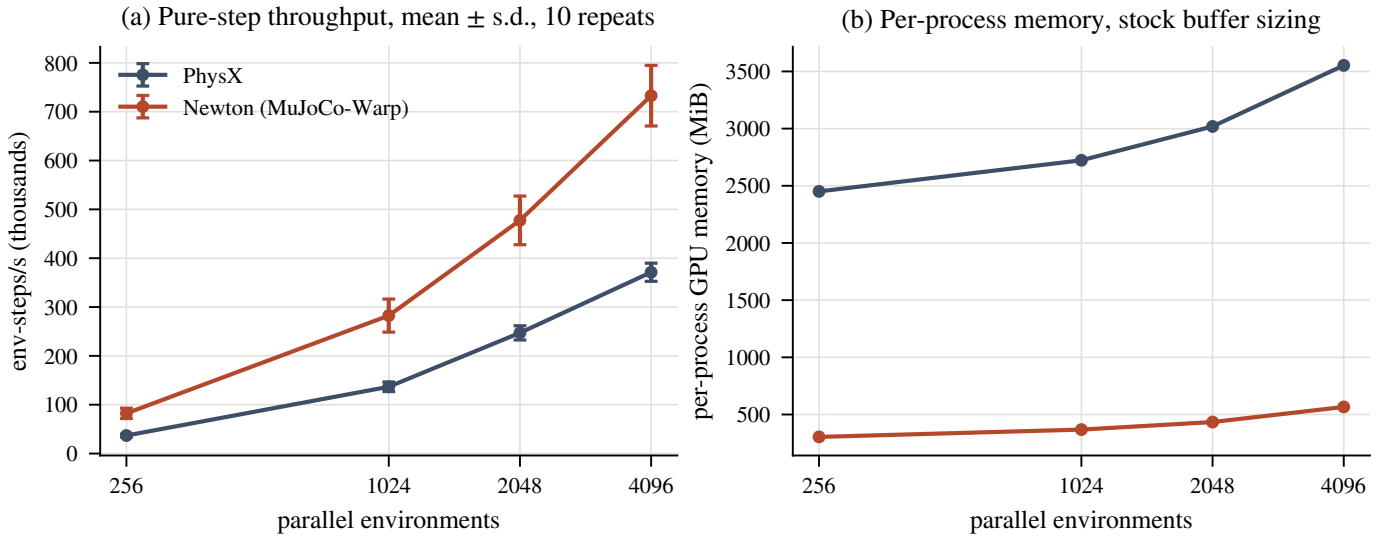


Fig. 1. Compute scaling on the stock Go2 flat task. (a) Zero-action stepping throughput, mean ± s.d. over 10 timed blocks of 100 steps. Newton is 1.93× to 2.23× faster at every size. (b) Per-process GPU memory. The gap reflects stock buffer sizing (a large preallocated PhysX patch pool against tight MuJoCo-Warp contact and constraint budgets), not a tuned-for-tuned engine comparison.

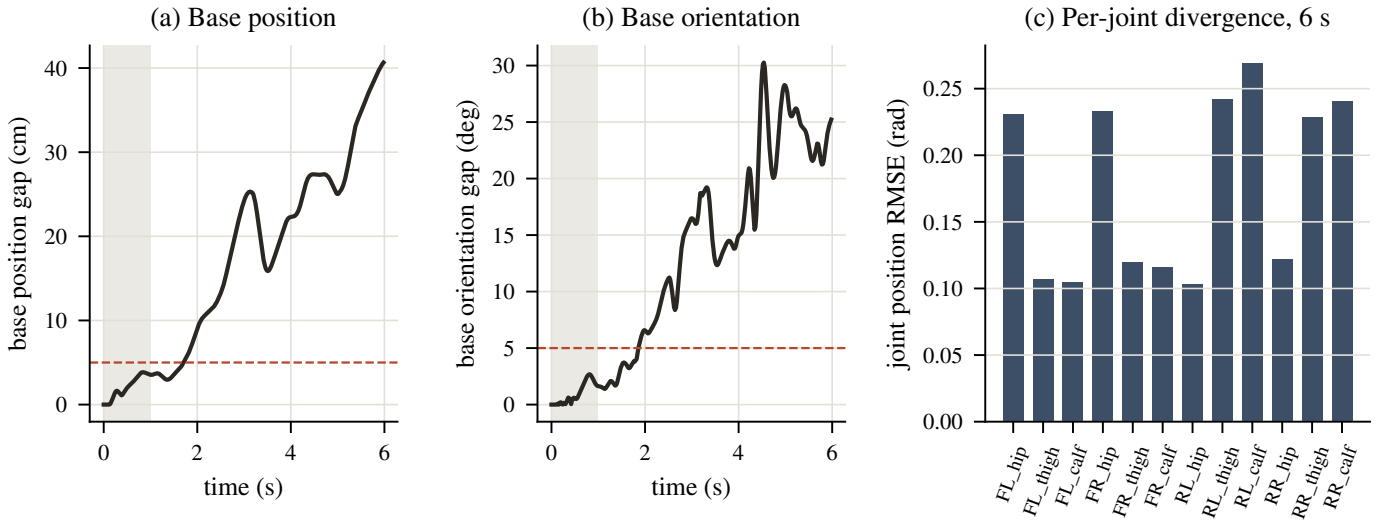


Fig. 2. Open-loop dynamics probe: identical initial state and action tape on both backends, one environment, no randomization. Shaded region: 1 s zero-action hold. Dashed lines: 5 cm and 5° divergence thresholds, crossed at 1.72 s and 1.86 s. Same-backend repeats diverge by exactly 0 in every channel, so all of the gap shown is attributable to the backend.

set of startup events. Two of them are live in the resolved configuration. `add_base_mass` adds a uniform draw from $[-1, 3]$ kg to the Go2 trunk, and `physics_material` sets the robot's friction to fixed static and dynamic values of 0.8 and 0.6, while the Newton arm keeps the spawned material at 1.0 and 1.0. The PhysX policies were thus trained on a randomized-mass, lower-friction robot, and the Newton policies on a nominal one. Either difference alone could plausibly move the reward curve, and nothing in these runs separates them from the engine.

We therefore report Table III's reward rows and Fig. 3 as an auditable record of what the stock presets produce, not as evidence about the engines. The throughput and memory rows of Table III are unaffected: startup events run once and do not change per-step cost. A backend-only learning comparison requires the preflight to pass (identical non-backend events in both resolved configurations) and all 20 runs repeated. We have not yet done that.

D. Cross-backend transfer, and an artifact that looks like a result

Our first cross-backend evaluation produced a striking result (Fig. 4a, b). Every one of 20 policies, 10 per training backend, falls over when moved to the other engine. Returns drop from about 26 to 29 to about -4 to -5, success falls to zero, and the collapse repeats with a second evaluation seed. Taken with the dynamics probe, a "policies do not transfer between engines" conclusion would have been easy to write.

It would have been wrong. The runtime contract check shows why. PhysX enumerates the Go2's joints type-major (all

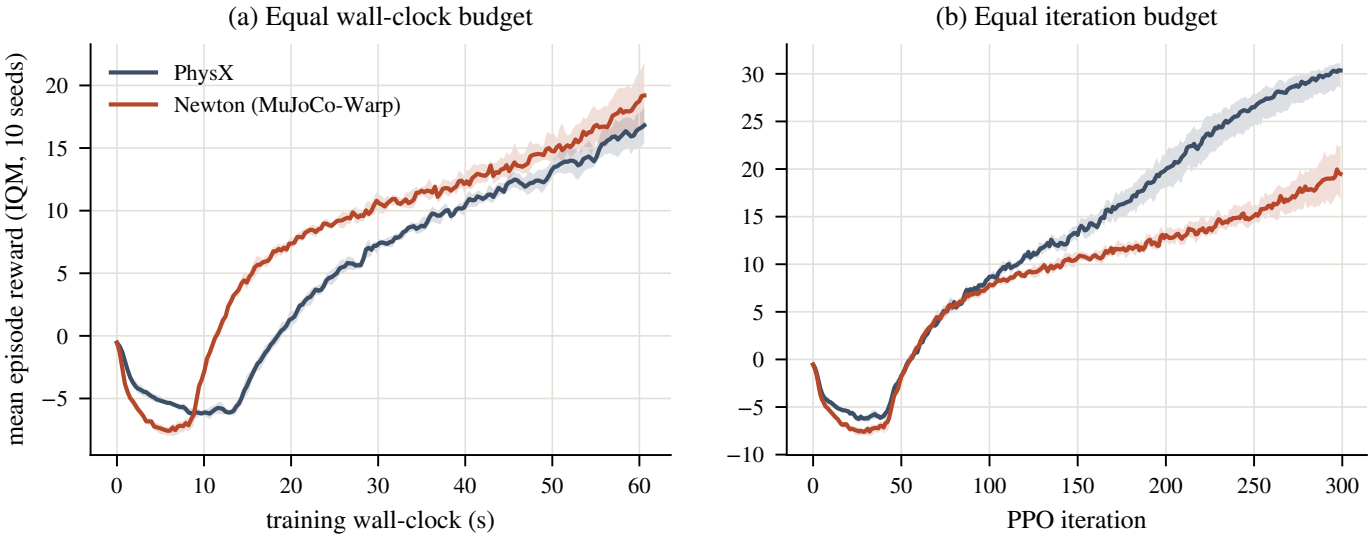


Fig. 3. Training reward, IQM over 10 seeds with 95% bootstrap bands. (a) Against wall-clock, truncated at the shortest Newton run (60.6 s). (b) Against PPO iteration. The two arms differ in more than the backend (Table IV), so the gap in (b) must not be read as a backend effect.

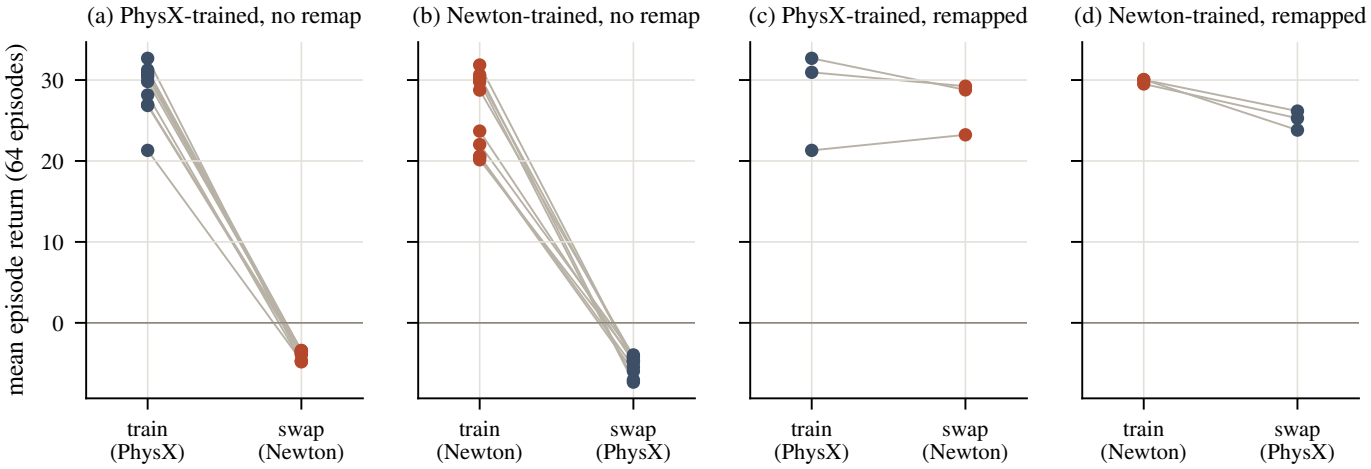


Fig. 4. Fixed-checkpoint returns on the training backend and after swapping backends, 64 episodes each, non-shared events disabled in both evaluations. (a, b) Without joint remapping, all 20 policies fail on the other backend. The runtime contract check shows that the policy's joint-indexed observations and actions address different joints after the swap. (c, d) With the checkpoint's joint order enforced, most performance survives the swap (3 policies per direction).

hips, then all thighs, then all calves), while Newton enumerates them leg-major (hip, thigh, calf for each leg in turn). The observation and action terms index joints by position in that list. A policy trained on one backend and loaded on the other reads its own joint positions and velocities in the wrong slots and commands the wrong motors. The failure comes from the interface, not the physics. Neither the task, the loader, nor the training logs report it.

With the checkpoint's joint order enforced at evaluation time, the contract check passes for the policy-facing joint semantics, and the picture changes (Fig. 4c, d, Table V). For three PhysX-trained policies evaluated on Newton, mean return moves from 28.3 to 27.1 and success rises from 0.90 to 0.97. The per-policy return differences are -1.7 $[-4.0, 0.7]$, $+1.9$ $[-1.3, 5.1]$, and -3.9 $[-6.1, -1.6]$. For three Newton-trained policies evaluated on PhysX, return drops from 29.9 to 25.1, all three differences are negative with intervals excluding zero (-6.2, -4.2, -3.9), and success stays at 0.92 or above. Transfer is imperfect, and it appears asymmetric, but it is nowhere near the collapse the unmapped runs showed.

Two cautions bound this result. First, it is a pilot: three policies per direction, with confidence intervals that bootstrap episodes within one simulator process and so do not capture process-to-process variation. Second, the source policies come from the confounded training runs above. The PhysX-trained policies saw mass and friction variation that the Newton-trained ones did not, which may explain part of the asymmetry. Rank statistics are not informative at $n = 3$. In the unmapped 10-policy runs, repeating the evaluation with a new seed on the same backend already produced 2 to 11 pairwise preference reversals out of 45, so return rankings at this episode count are themselves seed-sensitive. We do not interpret the unmapped cross-backend reversals as evidence about the engines.

TABLE IV
RESOLVED-CONFIGURATION AUDIT OF THE TWO STOCK PRESETS

Class	Fields
Harmless representation	Observations, actions, rewards, terminations, actuator model, Δt, decimation, policy network, PPO hyperparameters, seeds, environment count, horizon; log paths and run names; disabled events printed differently.
Required backend-specific	Physics manager and solver settings (PhysX iteration counts and GPU buffer capacities; MuJoCo-Warp `njmax`, `nconmax`, cone, integrator, substeps); contact-sensor implementation.
Uncontrolled confound	`events.add_base_mass` (startup, $+\mathcal{U}[-1,3]$ kg on the trunk): PhysX only. `events.physics_material` (startup, robot friction 0.8/0.6): PhysX only.

TABLE V
CROSS-BACKEND FIXED-CHECKPOINT EVALUATION (MEAN OVER POLICIES)

Direction	Joints	n	Return (own → swap)	Success (own → swap)	Valid
PhysX → Newton	raw	10	28.9 → −4.0	0.94 → 0.00	no
Newton → PhysX	raw	10	25.8 → −5.2	0.97 → 0.02	no
PhysX → Newton	mapped	3	28.3 → 27.1	0.90 → 0.97	pilot
Newton → PhysX	mapped	3	29.9 → 25.1	0.98 → 0.95	pilot

TABLE VI
STATUS OF EACH CLAIM AFTER THE PREFLIGHT CHECKS

Claim	Status
Newton steps $\approx 2\times$ faster (stock configs)	Holds; 10 repeats × 4 sizes
Newton trains $1.64\times$ faster end to end	Holds; 10 seeds per arm
$5.3\times$ lower per-process memory	Holds as a stock-config result only
Backends are individually deterministic and mutually non-equivalent	Holds; one tape, one asset
PhysX learns a higher-reward policy per iteration	**Not supported**: config confound
Newton is ahead at equal wall-clock	**Not supported**: same confound; intervals overlap at the end
Policies collapse across backends	**Invalid**: joint-order artifact
Remapped policies retain most performance	Pilot; $n = 3$ per direction

VI. DISCUSSION

A preset is not an intervention. The preset mechanism makes a backend swap look like a single-variable change, and on this task it is not. The stock PhysX events preset inherits from the Newton one and adds startup randomization. That is a reasonable engineering choice (some randomization terms may not yet be supported on the experimental backend), but it means that "same task, different preset" is not "same experiment, different engine." Diffing only the user-facing configuration would not have caught it. We found it only by recursively diffing the fully resolved configurations. We recommend that any published backend comparison include that diff.

Interfaces fail silently, and the failure looks like physics. The joint-ordering mismatch produced a clean, repeatable, seed-robust collapse that fit a plausible physical story, because the dynamics probe had already shown the engines disagree. Fixed-checkpoint evaluation is exactly where this trap springs: training on either backend alone is self-consistent, and only moving a policy exposes it. The same mismatch would affect any tooling that exchanges joint-indexed vectors across backends, such as action tapes, recorded demonstrations, or observation normalizers. A runtime check that compares the live joint names behind every joint-indexed term costs seconds and turns a silent failure into a loud one.

What a practitioner can take from this today. On this task and hardware, Newton delivers roughly $2\times$ simulation throughput and a much smaller memory footprint with stock settings, which is useful for memory-limited GPUs and simulation-dominated workloads. The two engines do not produce the same dynamics for the same asset, so a policy validated on one should be re-validated on the other before any claim about it moves across. Whether either engine yields better policies, or policies that transfer better to hardware, is not answered here.

VII. LIMITATIONS

This study covers one task, one robot, and one GPU. Both arms use stock solver settings, and per-backend tuning could change every ratio reported. The throughput probe steps zero actions. Policy-driven stepping with more contact events could behave differently, particularly near Newton's contact budget. The dynamics probe uses a single action tape at a single Δt, and its late-time divergence mixes systematic and chaotic components. The learning comparison is confounded and is reported only as an artifact. The transfer result is a 3-policy pilot built on confounded source policies. Newton support in Isaac Lab 3.0 is experimental, and these numbers will move as it matures. We make no sim-to-real claims.

VIII. CONCLUSION

Flipping the physics preset on a stock Isaac Lab task measured a real and repeatable compute advantage for Newton: about $2\times$ stepping, $1.64\times$ end-to-end training, and a several-fold smaller stock memory footprint. It also showed that the two engines produce different dynamics for the same robot. The same flip produced two results that did not survive inspection: a learning-quality gap that coincides with a hidden change in randomization events, and a total transfer failure caused by a joint-ordering mismatch. Both were invisible at the level of the preset. The next step is a rerun with the configuration preflight passing and the joint contract enforced from the start, followed by the question that decides adoption, which is transfer to the physical robot.

ARTIFACT AVAILABILITY

All raw artifacts (probe JSONs, per-iteration training extracts, resolved configurations, dynamics trajectories, per-episode evaluation records), the semantic-configuration and runtime-contract checks, and the script that regenerates every figure and derived statistic in this report are available at https://github.com/yusufdxb/physx-newton-bench. The preflights are the scripts `compare_semantic_configs.`

`py` (run with `--fail-on-unsafe`) and `compare_runtime_contracts.py`.

REFERENCES

[1] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Proc. Conf. Robot Learning (CoRL)*, ser. PMLR, vol. 164, 2022, pp. 91–100.

[2] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High performance GPU-based physics simulation for robot learning," arXiv:2108.10470, 2021.

[3] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.

[4] NVIDIA, "Isaac Lab," https://github.com/isaac-sim/IsaacLab, 2026, version 3.0.0, commit a4a7602f.

[5] Newton contributors, "Newton: An open-source, GPU-accelerated physics simulation engine built upon NVIDIA Warp," https://github.com/newton-physics/newton, 2026, version 1.0.0.

[6] Google DeepMind and NVIDIA, "MuJoCo Warp," https://github.com/google-deepmind/mujoco_warp, 2026, version 3.5.0.2.

[7] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.

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

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

[10] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2015, pp. 4397–4404.

[11] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021.

[12] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *Proc. IEEE Symp. Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[13] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artificial Intelligence*, 2018.

[15] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[16] Isaac Lab contributors, "[Bug Report] Newton backend re-enables joint target stiffness/damping for explicit actuators, causing different rest pose from PhysX," https://github.com/isaac-sim/IsaacLab/issues/5806, 2026, issue #5806, closed August 2026.