EDEN: ONE TASK DEFINITION FROM SIMULATION TO REAL ROBOTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Simulation has become central to robot learning, yet a task is often reimplemented as it moves from training to data collection, evaluation, and real-world deployment. In particular, deploying a policy on a physical robot typically requires reconstructing its observation and action pipelines against a new hardware interface, creating duplicated code and opportunities for subtle inconsistencies. We present Eden, a robot-learning framework in which a task is defined once and executed across simulation and hardware. Eden represents each task as a typed, serializable configuration of reusable terms operating over name-resolved entities. This separation of task semantics from the underlying execution substrate allows a simulated robot to be replaced by a hardware-backed entity while preserving the observation, action, and command code used during training. The same task definition also supports data collection and evaluation under alternative physics without reimplementation. We test whether this abstraction yields measurable benefits. In a controlled coding-agent benchmark across Eden, IsaacLab, and mjlab, agents reach a working policy faster in Eden on new tasks that require building a scene and composing components, despite less prior exposure to Eden code, but not on tasks adapted from existing code, and Eden does not reduce token use in general. We show that policies trained in simulation transfer to hardware such as the Unitree G1 and LimX Oli without rewriting their observation or action pipelines. Separately, a vision-language-action policy fine-tuned on 100 simulated demonstrations collected in Eden succeeds in 27/30 real-world pick-and-place trials. Together, these results show that training, data collection, evaluation, and deployment can be different executions of a single robot-learning task definition rather than separate implementations.

1 INTRODUCTION

Robot policies are increasingly trained in simulation. A typical workflow has four stages: train a policy in parallel simulated environments, collect demonstrations, evaluate the policy, and deploy it on a physical robot. However, while the policy itself may transfer to the physical world, the software defining its task rarely does, leaving each stage to re-implement the same environment interfaces. Each stage can introduce its own implementation of the same task. Deployment is where this costs most: running a policy on a physical robot usually means rewriting its observation and action processing against the robot's SDK (coordinate transforms, history stacking, normalization, action scaling, and PD-target computation). Any mismatch between this second implementation and the training code changes the policy's inputs or outputs, and every later change to the task must be made in both places.

Recent infrastructure has made simulation fast and environments modular. Isaac Gym (Makoviychuk et al., 2021) introduced massively parallel GPU simulation, followed by GPU-accelerated MuJoCo (Zakka et al., 2025; 2026), SAPIEN (Xiang et al., 2020; Tao et al., 2024), Newton (Newton Physics Contributors, 2025), and Genesis (Authors, 2024; Genesis AI Team, 2026). Manager-based frameworks such as IsaacLab (Mittal et al., 2023) and mjlab (Zakka et al., 2026) compose an environment from reusable observation, action, reward, termination, command, and event terms. These frameworks still keep a term's configuration and its implementation in separate classes, and they deploy policies through robot-specific scripts that reimplement the observation and action processing.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

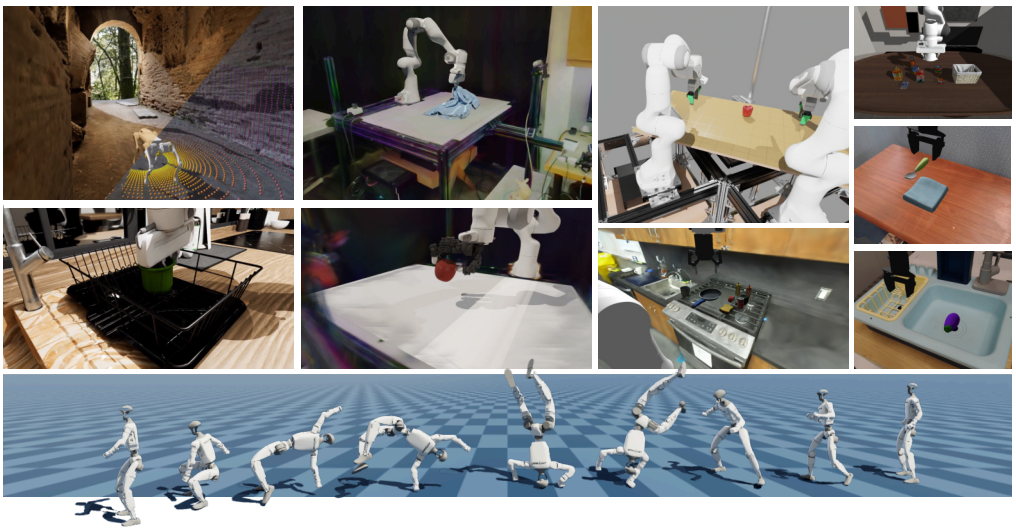


Figure 1: **One task definition for training, data collection, evaluation, and deployment.** A single typed Eden configuration is used for training, demonstration collection, evaluation under changed physics settings, and deployment on hardware.

This raises the question we address: *can one task definition be used unchanged for training, data collection, evaluation, and deployment?*

We present Eden, a robot-learning framework built on Genesis that answers this question by construction. In Eden, a task is a typed, serializable configuration object, the `EdenConfig`, composed of reusable terms. Each term declares its own parameters, so the configuration is derived from the implementation rather than maintained as a separate class. Configurations are validated when they are created, resolve robot links and joints by name when the environment is built, and are saved together with the code versions that produced them.

Terms read state and send commands only through *entities*, which they reference by name. In simulation, the robot entity is backed by Genesis. At deployment, it is replaced by an entity with the same interface that reads the physical robot, so the observation, action, and command code from training runs unchanged; the robot-specific code only reads state, initializes the robot, and sends commands. Solver settings, the contact model (Genesis' rigid solver, IPC, or our batched Mochi add-on), and a MuJoCo backend are likewise selected in the configuration without changing the task, and teleoperation and recording use the environment that was used for training.

We evaluate whether this design makes tasks easier to implement and whether a saved task can be run on hardware. First, we give coding agents the same task description, matched documentation, and a fixed time budget in Eden, IsaacLab, and mjlab, and score the resulting policies with success checkers that are independent of the training reward. Despite less prior exposure to Eden code, agents implement new tasks that require building a scene and combining components faster in Eden. Second, we run saved tasks on hardware: policies trained in Eden run on a Unitree G1 and a LimX Oli without rewriting their observation or action code, and a VLA fine-tuned on 100 demonstrations collected in Eden succeeds in 27/30 real pick-and-place trials.

Contributions.

- **A single-definition task configuration.** Terms declare their own parameters and access the world through entities referenced by name, so an `EdenConfig` is a complete, serializable task definition with no separate configuration classes (§3.1–3.2).
- **Deployment by entity replacement.** Replacing the simulated robot entity lets a saved task run on a physical Unitree G1 without rewriting its observation, action, or command code; five whole-body policies are deployed this way. Solver settings, three contact models, and a MuJoCo backend are configuration fields, and the contact models are compared at engine level (Appendix G; §3.3, §4.2).
- **A coding-agent benchmark for framework comparison.** We hold the task description, documentation budget, agent, and success checker fixed and vary only the framework, and report the effort

Framework	Engine	Task structure	Term definition	Deployment	Changed physics
Isaac Gym	PhysX	monolithic	—	—	✗
Playground	MJX	monolithic	—	robot-specific script	✗
ManiSkill3	SAPIEN/PhysX	monolithic	—	—	✗
IsaacLab	PhysX/Newton*	manager	separate class	robot-specific script	○*
mjlab	MuJoCo Warp	manager	separate class	—	✗
RoboVerse	multiple	wrapper	—	—	○
Eden	Genesis	**manager**	**derived from term**	**✓ training task, unchanged**	✓

Table 1: **Task definitions in robot-learning frameworks.** *Term definition*: whether a term's configuration is a separate class that points to its implementation, or is derived from the term itself. *Deployment*: whether hardware runs a robot-specific script, or the training task with its observation, action, and command managers. *Changed physics*: whether a trained policy can be evaluated under different physics without reimplementing the task. *Experimental at the time of writing.

needed to reach a working policy in Eden, IsaacLab, and mjlab, separately for tasks that can be adapted from existing code and tasks that must be built from components (§4.1).

2 Related Work

Physics engines and robot-learning frameworks. Physics engines provide dynamics, contact, and constraint solving; platforms add rendering, assets, sensors, and devices; robot-learning frameworks define the environments in which policies are trained, evaluated, and deployed. GPU-parallel engines and platforms include PhysX/Isaac Gym (Makoviychuk et al., 2021), MuJoCo and its accelerated backends (Todorov et al., 2012; Zakka et al., 2025; 2026), SAPIEN (Xiang et al., 2020; Tao et al., 2024), Newton (Newton Physics Contributors, 2025), and Genesis (Authors, 2024; Genesis AI Team, 2026). Eden is a framework built on Genesis. Physics throughput, contact models, solver breadth, and hardware acceleration come from Genesis; our evaluation measures only what the framework adds.

Task definitions. IsaacGymEnvs (Makoviychuk et al., 2021), the ManiSkill series (Mu et al., 2021; Gu et al., 2023; Tao et al., 2024; Shukla et al., 2024), and MuJoCo Playground (Zakka et al., 2025) implement each task as a self-contained environment class. IsaacLab (Mittal et al., 2023) introduced manager-based environments, in which observations, actions, rewards, terminations, commands, and events are composed from reusable terms; mjlab (Zakka et al., 2026) brings this design to MuJoCo Warp. Eden follows the manager-based design and differs in two ways (Table 1). First, a term's configuration is derived from its implementation rather than written as a separate class (§3.1). Second, the task is used unchanged on hardware. IsaacLab and MuJoCo Playground deploy policies through robot-specific scripts that reimplement the observations and actions; in Eden, the training configuration and its observation, action, and command managers run against an entity that reads the physical robot (§3.3). Cross-simulator frameworks such as RoboVerse (Geng et al., 2025) provide common task and benchmark interfaces across engines; they do not keep a task's implementation fixed from training to deployment.

Evaluating robot-learning frameworks. Infrastructure such as Gymnasium (Towers et al., 2024), Stable-Baselines3 (Raffin et al., 2021), CleanRL (Huang et al., 2022), and TorchRL (Bou et al., 2024) is evaluated by the abstractions it provides, the workloads it supports, and the overhead it adds. We also measure a property that is usually assessed qualitatively: the effort to implement a new task. Coding-agent benchmarks such as SWE-bench (Jimenez et al., 2024) fix the software and vary the agent. We do the reverse: we fix the agent, task description, documentation budget, and evaluation, and vary the framework (§4.1). To our knowledge, coding agents have not previously been used to compare robot-learning frameworks in this way.

3 Eden Design

3.1 Task Configuration

In Eden, a single configuration object fully encapsulates a task setup. An `EdenConfig` holds the scene, the environment settings, and typed configurations for observations, actions, rewards,

terminations, commands, events, curricula, metrics, recording, and the learning algorithm. The environment is built from this object alone, and the object can be saved, inspected, modified, and loaded again.

Terms declare their own configuration. A *term* is a reusable computation such as an observation, reward, action transform, termination condition, or command generator. In IsaacLab and mjlab, a term is typically written twice: once as an implementation and once as a configuration class that points to the implementation and repeats its parameters and defaults. The two can diverge as either changes, and a user must keep both in mind. In Eden, a term declares its typed parameters and defaults itself, and `configure(...)` returns a typed options object derived from that declaration. The returned object is data, so it can be composed into a task, validated, saved, and used later to rebuild the same term (Figure 9, Appendix B).

Errors are reported before the first step. Unknown parameters are rejected when a term is configured. Names that depend on the scene (entities, links, and joints) are resolved when the environment is built. A misspelled parameter or joint name therefore fails before the first environment step rather than during training. In the agent suite of §4.1, Eden rejects two misspelled-field faults at import that IsaacLab and mjlab accept.

A saved configuration identifies the run. A configuration is saved with its concrete type and the Eden and Genesis versions that produced it; loading it rebuilds the same components and reports version mismatches. Training wrappers also record the learning-library version and random seeds with the checkpoint. A saved configuration and checkpoint therefore identify both the weights and the task they were trained on.

3.2 Terms and Entities

A task can run on a different simulator or on hardware only if its terms do not depend on simulator objects. Terms never access Genesis objects directly. They refer to entities, links, and joints by name, for example `entity_name="robot"` or `dofs_name="joint.*"`, and these names are resolved when the environment is built.

Entities provide state and actuation. A term reads the world only through the entity interface, which provides joint positions and velocities, link poses, Jacobians, and contact state. An observation term that needs joint positions calls the same entity method whether the positions come from Genesis, from MuJoCo, or from a physical robot. An action term converts a policy output into joint targets without depending on what applies them.

Referencing robots by name also separates a task from a particular robot where the task allows it. A reaching reward that refers to `entity_name="robot"` and an end-effector link by name is unchanged when the robot and link name are replaced. A task that refers to bodies specific to one morphology stays tied to that morphology; referencing by name makes this dependency explicit rather than removing it.

A common term interface. Stateless function terms and stateful class terms are registered and configured the same way, and user-defined terms share the registries of library terms. A term may cache derived values such as joint indices, but never a reference to an entity object; it looks the entity up by name. This rule lets the entity behind a name change without changing the term. Figure 2 summarizes the design. An `EdenConfig` defines managers composed of terms and a scene composed of named entities. Changing the implementation behind an entity changes where the task runs, not what the task computes.

3.3 Deployment and Changed Physics

The same `EdenConfig` and managers run with a Genesis entity, a MuJoCo model, or an entity that reads a physical robot. In each case the policy receives observations from the same terms and its outputs pass through the same action processing; only the source of state and the destination of commands change.

Deployment by entity replacement. During training, the robot entity is backed by Genesis. At deployment, Eden loads the training configuration and replaces the simulated robot entity with a `RobotStateEntity` that has the same interface. Each control step writes the robot's measured

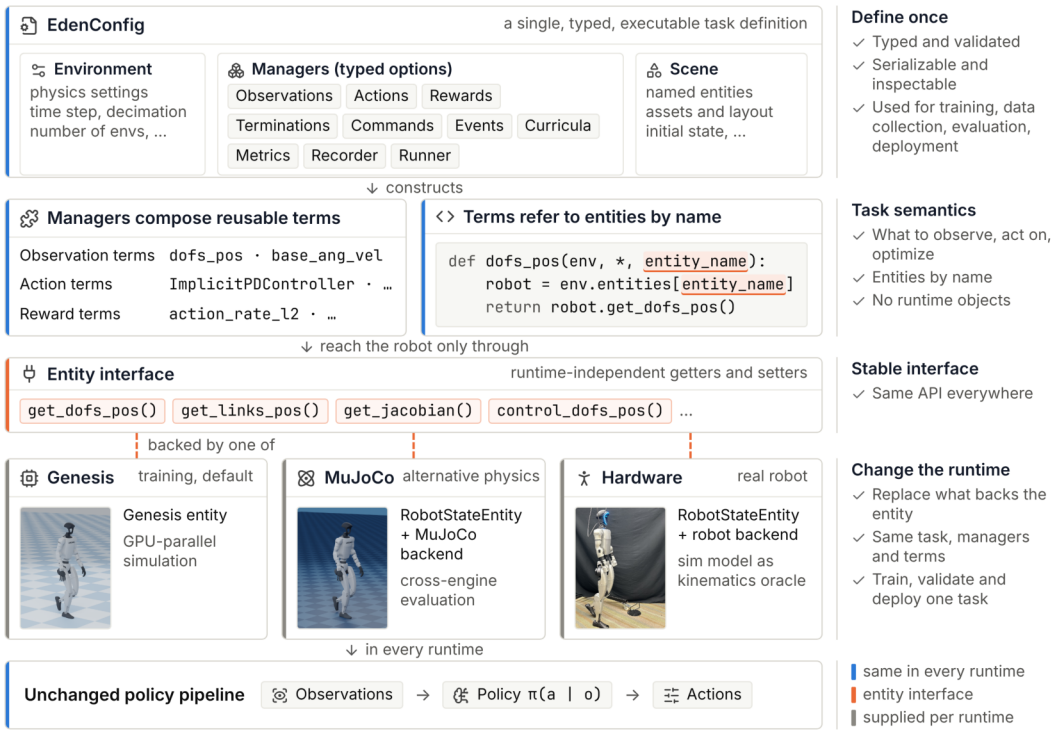


Figure 2: **Eden architecture.** One `EdenConfig`, optionally modified by `TaskMod`s, holds one typed options object per manager and per scene entity. Vectorized environments are built from it for RL (rsl_rl, rl_games, etc.) and for imitation-learning data collection (LeRobot export). At deployment, the simulated robot entity is replaced by one that reads the physical robot, and the observation and action managers run unchanged.

state into this entity, so observation terms read the same quantities by the same names as in training. The simulated entity is kept as a kinematic model: after it is set to the measured joint positions, link poses and Jacobians are computed with the same model and conventions used in training.

The environment's action manager processes the policy output before it reaches the robot, so scaling, clipping, joint selection, and PD-target computation use the training code rather than a deployment script. The robot-specific backend implements five operations: `connect`, `close`, `init_sequence`, `read_state`, and `send_payload`. It exchanges data with the task through a common `RobotState`/`RobotCommand` interface.

Entity replacement does not close the sim-to-real gap. An observation that the robot cannot measure must be removed from the task or estimated, and actuator mappings and gains must match the physical robot. What entity replacement removes is a second implementation of the policy's observation and action processing.

Changed physics. Physics settings are fields of the task configuration. Within Genesis, the solver, contact, iteration, and substep settings are fields of `EnvOptions`, and changing them changes no observation, action, reward, termination, or reset term. The same holds for the contact model. An entity is simulated by Genesis' constraint-based rigid solver, by IPC (Li et al., 2020) through Genesis' IPC coupler, or by Mochi; the choice is set by the entity's material and, for IPC and Mochi, one matching option on `EnvOptions`, and the combination is validated when the environment is constructed. Mochi is our batched reimplementation, in Genesis' kernel language, of the physics engine of Project SuperDex (Meta Platforms, Inc., 2026), added to Genesis as an add-on solver. Appendix G compares the three at engine level and finds that none is best on every test: IPC never interpenetrates but leaves resting contacts up to a millimeter apart and misestimates sliding friction, while the rigid solver is the most accurate on friction and fails on deformables. The choice therefore depends on the task, and Eden makes it a configuration field rather than a port to another framework. For sim-to-sim transfer, the MuJoCo backend runs the same managers against a MuJoCo model. In

each case the dynamics change and the task code does not. Section 4.2 tests this on the largest change, from simulation to a physical robot.

3.4 Task Variants, Training, and Data Collection

Task variants. A behavior is often needed for several robots, controllers, observation sets, or learning algorithms. Eden expresses each variation as a typed `TaskMod` that modifies a configuration before the environment is built, rather than as a separate task implementation. Mods along independent axes compose, and the saved result is a complete configuration (Appendix E).

Training, evaluation, and data collection. RL adapters connect the environment to different learning libraries. Evaluation loads the saved configuration and checkpoint rather than a separate evaluation environment. For data collection, a teleoperation device replaces the policy as the source of actions, and the recorder manager saves states, observations, actions, and sensor outputs at fixed points in the same step and exports trajectories for imitation or offline learning.

4 Evaluation

We measure two things: the effort a coding agent needs to implement a new task, with and without similar existing task code (§4.1); and whether a saved task runs on a physical robot without rewriting its observation and action code (§4.2). Engine-level throughput and physical fidelity are properties of Genesis and are reported in the appendix, including a comparison of the three contact models a task can select (Appendix G).

4.1 Effort to Implement New Tasks

We measure how long a coding agent needs to implement a new task in each framework, giving the same agent the same task and information and changing only the framework.

Setup. We compare Eden, IsaacLab, and mjlab on five tasks. Two, Go1 spin and Go1 backward crawl, can be adapted from the Go1 velocity-tracking task that all three frameworks ship. Three, door opening, bin picking, and deformable-object cutting, have no counterpart in any of them: the agent must build the scene, add the robot and objects, and compose the observations, actions, rewards, and terminations. A sixth task, the Go1 backflip, is repeated with four coding models, and Claude Opus 5.5 also runs cube pushing and a G1 handstand. In every framework the agent receives the same task description, a documentation package matched in structure and size, and the installed framework source, and has no web access. Each rollout is a fresh session with a 3 h budget and must leave code that trains a policy and exports a rollout. We run that code and score the exported behavior with a task checker written before the experiment. The checker does not use the training reward, starts from perturbed initial states where an open-loop replay could pass, and rejects fabricated or replayed trajectories (Appendix J).

Results. All three frameworks reach a working policy. Eden and IsaacLab pass 5/5 rollouts on every task, and mjlab passes 5/5 on every task it can represent except bin picking (4/5; Table 2). Deformable cutting requires a separable continuum material, which mjlab does not provide, so it is not entered rather than scored as a failure. The frameworks differ in the time to a working policy.

Eden is fastest when a task is built from components. On door opening, bin picking, and deformable cutting, agents reach a working policy in 47.6, 29.4, and 64.0 min with Eden. The other frameworks take 53.7–98.4 min on door opening and 60.2–117.6 min on bin picking, a factor of 2.0 over mjlab and 4.0 over IsaacLab on the latter; IsaacLab's reference cell on deformable cutting, run on its experimental Newton MPM backend, takes 102.0 min. When the task can be adapted from shipped code, Eden gives no advantage: IsaacLab is fastest on spin (48.5 against 146.5 min for Eden), and crawl takes 69.1–77.2 min in all three. Building a task from components exercises the term and entity interfaces of §3, whereas adapting one mostly copies existing code. Because Eden runs on the same engine for all five tasks and is slowest on spin, engine throughput alone does not explain this pattern.

	pass@5			Wall-clock (min)			Tokens (M)		
Task	IsaacLab	mjlab	Eden	IsaacLab	mjlab	Eden	IsaacLab	mjlab	Eden
Adapted from a shipped task									
Go1 spin	5/5	5/5	5/5	**48.5**	116.8	146.5	6.0	23.8	18.5
Go1 backward crawl	5/5	5/5	5/5	**69.1**	77.2	70.5	6.2	11.4	18.4
Built from components; the agent authors the scene									
Door opening	5/5	5/5	5/5	98.4	53.7	**47.6**	21.9	27.4	51.2
Bin picking	5/5	4/5	5/5	117.6	60.2	**29.4**	18.7	15.8	9.6
Deformable cutting	5/5*	–	5/5	102.0*	–	**64.0**	23.2*	–	14.5

Table 2: **Coding-agent benchmark**, Claude Opus 5, five independent rollouts per cell. pass@5 counts the rollouts whose exported policy passes the task's success checker within the 3 h budget. Wall-clock (time to a working policy) and tokens are means over rollouts. The shortest wall-clock per task is in bold. Deformable cutting requires a separable continuum material, which mjlab does not provide (–); *IsaacLab runs it on its experimental Newton MPM backend.

	pass@k				
Framework	Opus 4.7	Gemini 3.1 Pro	Opus 5	Haiku 4.5	**Total**
IsaacLab	3/5	4/5	3/5	0/3	10/18
mjlab	4/5	1/5	5/5	0/3	10/18
Eden	4/5	4/5	5/5	0/3	**13/18**

Table 3: **Go1 backflip under four coding models**, five independent rollouts per cell (three for Claude Haiku 4.5). A rollout passes if the checker passes its exported policy within the 3 h budget. The checker requires a full backward rotation, at least 30% of it accumulated in free flight, and an upright, settled landing; the Opus 5 and Haiku 4.5 rollouts were re-scored under it from an earlier checker version, with no outcome changed. One IsaacLab Opus 5 rollout passes the checker after 207 min and counts as a failure. Wall-clock and tokens on the release are reported for Claude Opus 5.5 only (Appendix J.2).

The backflip result holds across models. Under each of four coding models, Eden passes at least as many backflip rollouts as either other framework, 13 of 18 in total against 10 of 18 for IsaacLab and for mjlab (Table 3). No framework passes with Claude Haiku 4.5.

Median agent-side time is lowest with Eden. Wall-clock includes the training the agent launches, which depends on the schedule it chose and on engine throughput. With Claude Opus 5.5, one rollout per cell, the harness separates agent-side time, spent reading, writing, and running code, from compute-side training time (Table 33). Every entered cell passes. On the seven tasks that all three frameworks run, the median agent-side time is 9.4 min with Eden, 11.0 with mjlab, and 22.2 with IsaacLab (21.5 without two cells whose split the harness could not separate). Eden is lowest on four of the seven tasks and within 0.4 min of the lowest on the other three. With one rollout per cell this is descriptive, not a ranking.

Eden does not reduce token use in general. With Claude Opus 5, IsaacLab uses the fewest tokens on spin, crawl, and door opening, and Eden the fewest on bin picking and deformable cutting. With Claude Opus 5.5, Eden uses the most tokens on five of seven tasks. The differences in time therefore do not come from lower token use.

Small edits to one task. An exploratory 36-task suite, run before its protocol was frozen, isolates the configuration surface from policy training. Each task is a small edit to the flat-terrain G1 velocity-tracking task that all three frameworks ship, the tasks together cover every manager, and a framework-neutral probe checks each result by building the task and inspecting manager state (Appendix J.3). All three frameworks pass all 36 tasks with similar agent time. Eden's solutions are 31% shorter than IsaacLab's and 33% shorter than mjlab's (734 against 1066 and 1096 lines), mostly because a configuration becomes a named task through a one-line decorator rather than a registration call that also names the environment class and the learning configuration (42 lines against 279 and 217). Two debug tasks start from a misspelled parameter field. Eden rejects both at import with an

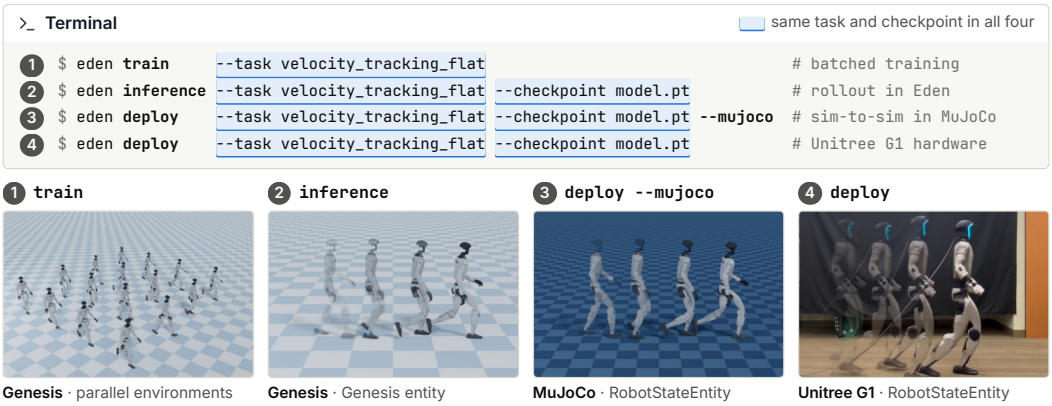


Figure 3: **One configuration and one checkpoint, four commands.** The task configuration of §3.1 is what each command loads; training, evaluation, sim-to-sim validation, and hardware differ in the command and, for the last two, in which entity backs the robot. Appendix H lists and specifies the tasks.

error that names the field and the accepted fields, whereas IsaacLab and mjlab accept both, and the fault shows only as an unchanged reward weight or resampling time in the built task (§3.1).

Coding models have likely seen IsaacLab and mjlab code, documentation, and examples during pretraining, whereas Eden is new. This asymmetry, which we cannot measure, favors IsaacLab and mjlab (§5).

4.2 One Task Definition in Simulation and on Hardware

Starting from a saved task and checkpoint, we change what runs the task. The robot's behavior may change; the test is whether the same observation, action, command, reward, termination, and reset code runs throughout (Figure 3).

Changed physics. Solver, contact, iteration, and substep settings, and the contact model itself (Genesis' rigid solver, IPC, or Mochi), are fields of the saved task (§3.3), and the MuJoCo backend runs the same managers against a MuJoCo model (Figure 3). Changing the physics therefore changes no term, by construction. We do not report how trained policies behave under each change. Appendix G measures the three contact models at engine level on eleven scenes against simulator-independent ground truth. Eden's role in that comparison is to select the contact model by configuration and to report the defaults each one ships with rather than hide them (for example, libuipc's default Newton tolerance makes a sliding cube stick).

Deployment on hardware. We load the training configuration of each policy and replace only the G1 entity with one that reads the physical G1. The observation, action, and command managers, including history stacking, scaling, clipping, joint selection, and PD-target computation, run the training code unchanged. We test five BeyondMimic policies with ten trials each. The deployed variant is trained without the privileged base linear velocity observation, which the robot does not measure, and all training-time randomization is disabled during evaluation. Basic walking, short emote, sidekick, and balancing succeed in 10/10 trials; dynamic dancing succeeds in 7/10. The three failures coincide with a disconnected wired link or excessive gantry-tether tension, not with a change to the policy interface. No observation, action, or command code is rewritten for any of the five policies (Table 4, Appendix I).

Figure 35 in Appendix I shows frames from two hardware deployments.

Collecting demonstrations for a real robot. The same environment is used for data collection. For a UR10e with a Robotiq 2F-85, the workcell is reconstructed as a 3D Gaussian splat, 100 demonstrations are teleoperated in Eden with a GELLO leader arm and saved by the recorder manager, and GR00T N1.7 is fine-tuned on their LeRobot export. On the real workcell, where the robot is driven by the VLA's own inference server rather than an Eden deployment backend, the fine-tuned policy succeeds in 27/30 trials. The same collection in IsaacLab on the same splat

Robot / task	Policy	Condition	Success (real)
Unitree G1, whole-body tracking	BeyondMimic, basic walking	Eden training config, entity replaced	10/10
	BeyondMimic, short emote	same	10/10
	BeyondMimic, sidekick	same	10/10
	BeyondMimic, balancing	same	10/10
	BeyondMimic, dynamic dancing	same	7/10‡

Table 4: **Unitree G1 hardware results**, ten trials per policy. Policies were trained in Eden and deployed by entity replacement with no observation, action, or command code rewritten, at a 50 Hz policy rate with all training-time randomization disabled. ‡All three failures were attributed to external causes (a disconnected wired link, excessive gantry-tether tension); Appendix I gives the setup.

scene yields 23/30, and demonstrations without the reconstructed background yield 0/30 (Table 30 in Appendix I).

5 Limitations

Evaluation scope. The benchmark measures coding agents, not human users. Coding models may have seen IsaacLab and mjlab code during pretraining, whereas Eden is new, and we cannot measure this exposure. Five rollouts per cell (one for Claude Opus 5.5) leave substantial variance, and the 36-task suite was run before its protocol was frozen.

Hardware and changed physics. Entity replacement is tested only on the Unitree G1 with whole-body tracking policies; we report no trials for the LimX Oli of Figure 35. The UR10e experiment tests data collection, not deployment: the VLA's own inference server drives the robot. We do not measure how trained policies behave when the solver, contact model, or simulator changes; the contact-model comparison in Appendix G is engine-level and involves no policy.

Backend requirements. A task runs unchanged only if the target backend provides the quantities and operations it uses. An observation the physical robot does not measure must be removed during training or estimated, and a simulator-specific operation needs an equivalent on the backend. The entity interface makes these dependencies explicit but does not remove them.

Hardware integration and control rate. A new robot still requires a backend for connection, initialization, state reading, actuator mapping, and command sending; Eden removes only the second implementation of the observation and action processing. The Python deployment loop is paced to the 50 Hz policy rate and reports every tick that overruns it. The G1 trials predate that reporting, so we do not report their achieved rate; controllers at much higher rates need a faster implementation.

6 Conclusion

We presented Eden, a robot-learning framework on Genesis in which a task is one typed, serializable configuration whose terms access robots through named entities, so the same task serves training, data collection, evaluation under changed physics, and deployment. Coding agents reach a working policy faster in Eden than in IsaacLab and mjlab when a task must be built from components, gain no clear advantage when a similar task can be adapted, and do not use fewer tokens in general. Five whole-body policies run on a Unitree G1 with unchanged observation and action code, and a VLA fine-tuned on 100 demonstrations collected in Eden succeeds in 27/30 real pick-and-place trials.

References

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

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. TorchRL: A data-driven decision-making library for PyTorch. In *International Conference on Learning Representations (ICLR)*, 2024.

Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. The YCB object and model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics (ICAR)*, pp. 510–517, 2015.

Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

Genesis AI Team. The role of simulation in scalable robotics, Genesis World 1.0, and the path forward. Genesis AI Blog, May 2026. URL `https://www.genesis.ai/blog/the-role-of-simulation-in-scalable-robotics-genesis-world-10-and-the-path-forward`.

Haoran Geng, Feishi Wang, Songlin Wei, et al. RoboVerse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. ManiSkill2: A unified benchmark for generalizable manipulation skills. *International Conference on Learning Representations (ICLR)*, 2023.

Jinrui Han, Dewei Wang, Chenyun Zhang, Xinzhe Liu, Ping Luo, Chenjia Bai, and Xuelong Li. HUSKY: Humanoid skateboarding system via physics-aware whole-body control. In *Robotics: Science and Systems (RSS)*, 2026.

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening (lafan1), 2020.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Arhan Jain, Mingtong Zhang, Kanav Arora, William Chen, Marcel Torne, Muhammad Zubair Irshad, Sergey Zakharov, Yue Wang, Sergey Levine, Chelsea Finn, Wei-Chiu Ma, Dhruv Shah, Abhishek Gupta, and Karl Pertsch. PolaRiS: Scalable real-to-sim evaluations for generalist robot policies. *arXiv:2512.16881*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2023.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: Rapid motor adaptation for legged robots. In *Robotics: Science and Systems (RSS)*, 2021.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Transactions on Graphics (SIGGRAPH)*, 39(4), 2020.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. SimplerEnv: Simulated manipulation policy evaluation environments for real robot setups. *arXiv preprint arXiv:2405.05941*, 2024.

Yutong Liang, Quanquan Peng, Ri-Zhao Qiu, and Xiaolong Wang. ConTrack: Constrained hand motion tracking with adaptive trade-off control. *arXiv preprint arXiv:2606.03177*, 2026.

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. *arXiv preprint arXiv:2508.08241*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer in lifelong robot learning. *Neural Information Processing Systems (NeurIPS)*, 2024.

Zhengyi Luo, Jinkun Cao, Kris Kitani, Weipeng Xu, et al. Perpetual humanoid control for real-time simulated avatars. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10895–10904, 2023.

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision (ICCV)*, 2019.

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 *Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021.

Meta Platforms, Inc. Project SuperDex: A unified platform for dexterous manipulation research. `https://projectsuperdex.com/`, 2026. Code: `https://github.com/facebookresearch/project_superdex`, initial release 2026-08-24.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters (RA-L)*, 8(6):3740–3747, 2023.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021.

Newton Physics Contributors. Newton: A GPU-accelerated physics simulation engine for robotics. `https://github.com/newton-physics/newton`, 2025. Linux Foundation project initiated by NVIDIA, Google DeepMind, and Disney Research.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. In *ArXiv Preprint*, March 2025.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Yuto Shibata, Kashu Yamazaki, Lalit Jayanti, Yoshimitsu Aoki, Mariko Isogawa, and Katerina Fragkiadaki. Learning to assist: Physics-grounded human-human control via multi-agent reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 38345–38354, June 2026.

Arth Shukla, Stone Tao, and Hao Su. ManiSkill-HAB: A benchmark for low-level manipulation in home rearrangement tasks. *arXiv preprint arXiv:2412.13211*, 2024.

Stone Tao, Fanbo Xiang, Arth Shukla, et al. ManiSkill3: GPU parallelized robotics simulation and rendering for generalizable embodied AI. *arXiv preprint arXiv:2410.00425*, 2024.

Chen Tessler, Yifeng Jiang, Xue Bin Peng, Erwin Coumans, Yi Shi, Haotian Zhang, Davis Rempe, Gal Chechik, and Sanja Fidler. ProtoMotions3: An open-source framework for humanoid simulation and control. `https://github.com/NVlabs/ProtoMotions/`, 2025.

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)*, pp. 5026–5033. IEEE, 2012.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators, 2023.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yixin Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. SAPIEN: A simulated part-based interactive environment. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11097–11107, 2020.

Kevin Zakka, Philipp Wu, Laura Smith, Nimrod Gileadi, Taylor Howell, Xue Bin Peng, Sumeet Singh, Yuval Tassa, Pete Florence, Andy Zeng, and Pieter Abbeel. RoboPianist: Dexterous piano playing with deep reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2023.

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. *arXiv preprint arXiv:2502.08844*, 2025.

Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. mjlab: A lightweight framework for GPU-accelerated robot learning. *arXiv preprint arXiv:2601.22074*, 2026.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, Yifeng Zhu, and Kevin Lin. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

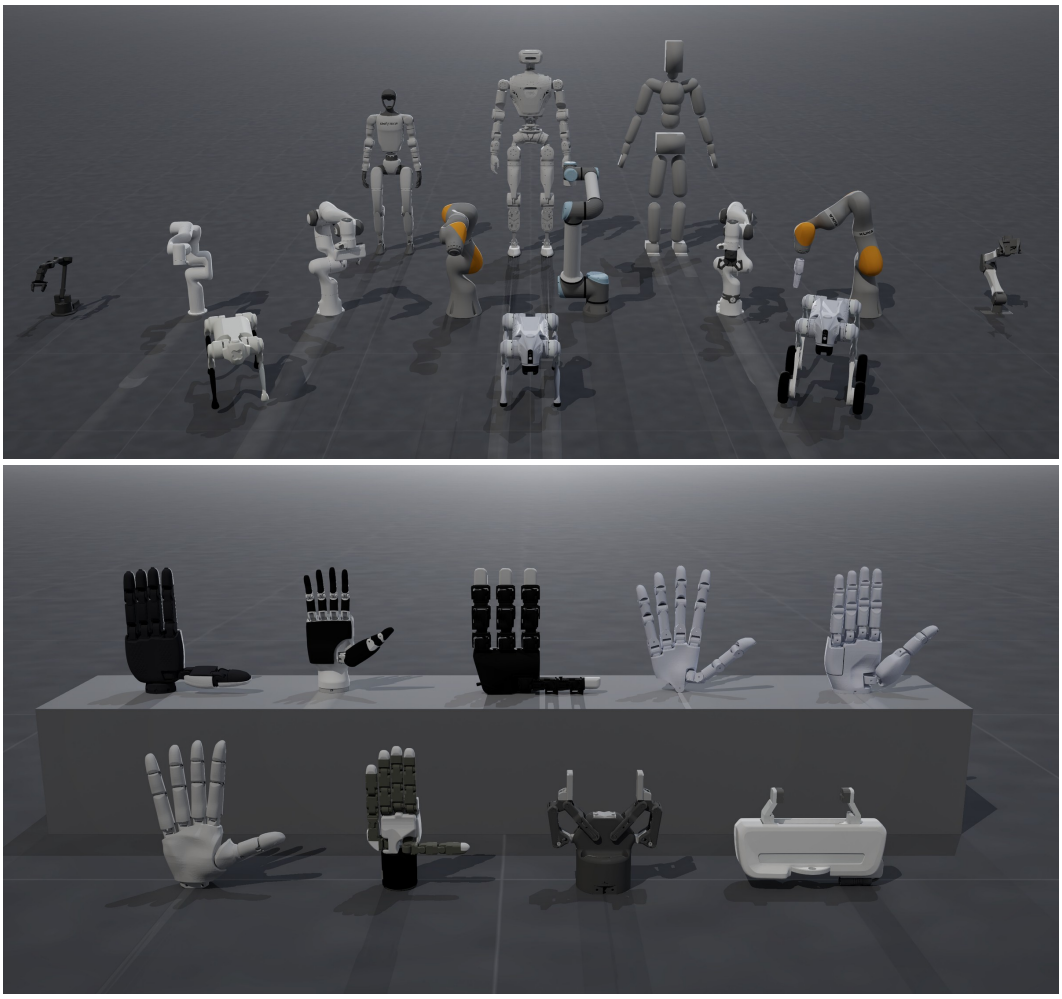


Figure 4: Robot embodiments, dexterous hands and grippers.

A General Capabilities of Eden

The main paper focuses on one design principle: a task is defined once and executed across the robot-learning lifecycle. Eden also includes the broader infrastructure needed to construct, train, evaluate, and deploy robot-learning systems. We summarize these capabilities here to document the scope of the released framework, rather than as separate contributions of this paper. Unless stated otherwise, they use the same configuration, entity, and registry interfaces described in the main text.

Tasks, robots, and assets. Eden includes task families spanning locomotion, whole-body tracking, rigid manipulation, and deformable interaction, together with ports of established robot-learning benchmarks. Tasks can be instantiated across a library of robot embodiments and scenes through the same entity and configuration interfaces. Assets are exposed through registries rather than task-specific loaders, allowing robots, objects, terrains, and reconstructed scenes to be reused across tasks. Table 5 summarizes the coverage of the evaluated release, and Appendix H specifies the native tasks in full.

Training and policy interfaces. The same vectorized environment can be consumed by multiple learning stacks, including `rsl_rl` and `rl_games`. Eden also provides adapters for pretrained policies and task suites whose observation and action conventions differ from its native tasks. These adapters translate at the policy boundary rather than introducing a second environment implementation. Checkpoints are stored together with the task configuration and learning metadata needed to reconstruct the environment in which they were trained.



Figure 5: Motion data. G1 poses retargeted from AMASS, LAFAN1, PHUMA and SEED; ARCTIC objects with MANO and four retargeted robot hands; OMOMO and HHI-Assist with SOMA-X bodies.

Motion and demonstration data. Eden provides a common representation for reference motion spanning single-actor motion, human–object interaction, and multi-actor sequences, with loaders for AMASS, LAFAN1, PHUMA, SEED, DexYCB, ARCTIC, OMOMO, and HHI. Retargeting backends map these sources to simulated embodiments, while filtering, resampling, and offline modifiers operate on the same motion representation. For demonstration collection, teleoperation drives the ordinary task action interface and the recorder manager captures trajectories from the same environment used for training and evaluation. Recorded trajectories can be exported to LeRobot format for imitation and offline learning.

Teleoperation and robot-learning tools. The framework includes interfaces for GELLO, Meta Quest, PICO, the Wuji glove, and SpaceMouse, together with motion retargeting, motion planning through cuRobo and OMPL, grasp sampling, system identification, pose estimation, and visualization. These utilities operate on the same scene entities and task configurations used by learning rather than defining separate representations of the robot or environment.

Scene construction and visual environments. Eden includes batched utilities for constructing manipulation scenes, including support-surface-aware placement that samples collision-free object poses directly on supporting geometry rather than relying on drop-and-settle initialization. Reconstructed environments can be represented by 3D Gaussian splats and composited with simulated robots and objects. The renderer supports fixed or per-environment cameras and shared or per-environment Gaussian geometry, allowing reconstructed scenes to participate in the same batched environment interface as conventional simulated scenes. These facilities are used by the manipulation, PolaRiS, and simulated-data workflows included with the release.

Physics and sensors. Through Genesis, Eden exposes rigid-body, MPM, SPH, PBD, and FEM simulation through the same environment configuration interface, together with the corresponding coupling and contact settings. These are Genesis capabilities, not new physics methods introduced by Eden. Eden adds the Mochi solver, our reimplementation in Genesis' kernel language of the physics engine of Project SuperDex (Meta Platforms, Inc., 2026), added to Genesis as an add-on solver (§3.3). Mochi assembles rigid and articulated bodies, tetrahedral solids, thin shells with self-contact, elastic rods, and the penalty contact between them into one implicit Newton solve per step, so a robot manipulates a deformable object without a rigid–soft coupler (Figure 7, bottom). The original engine steps one scene per engine instance; the reimplementation solves every environment of a batch in one solve, so deformable scenes run in Eden's batched environments like Genesis' own solvers. An entity moves to Mochi by taking a Mochi material, and the scene by setting



Figure 6: Simulated robots in Gaussian-splat scenes, per-environment object splats, support-surface placement, Gaussian-splat benchmark (PolaRiS) tasks.

`EnvOptions.mochi_options`; a Mochi scene may hold only Mochi and kinematic entities, and the combination is validated at construction. Appendix G measures Mochi against Genesis' rigid solver and IPC at engine level. The framework also exposes RGB, depth, normal, segmentation, tactile, IMU, and LiDAR sensing through its scene and sensor interfaces.

Deployment. The evaluated release includes hardware backends for Unitree G1, Go1, and Go2, LimX Oli, Franka Panda/FR3, and RoboTera XHand, together with the MuJoCo backend used for sim-to-sim validation. All implement the common `RobotState`/`RobotCommand` boundary described in Appendix D. Adding a backend therefore supplies robot-specific state acquisition, initialization, actuator mapping, and command delivery without introducing a separate observation or action implementation for each task.



Figure 7: Top: Genesis MPM sand, SPH liquid and PBD cloth with rigid robots. Bottom: the Mochi solver. A Franka folds a shell t-shirt; deformable FinRay fingers on a Franka fold around an apple and lift it; an RJ45 plug latches into a socket while trailing a self-colliding elastic-rod cable. In each Mochi scene the robot, the deformable bodies and their contacts are one implicit solve.

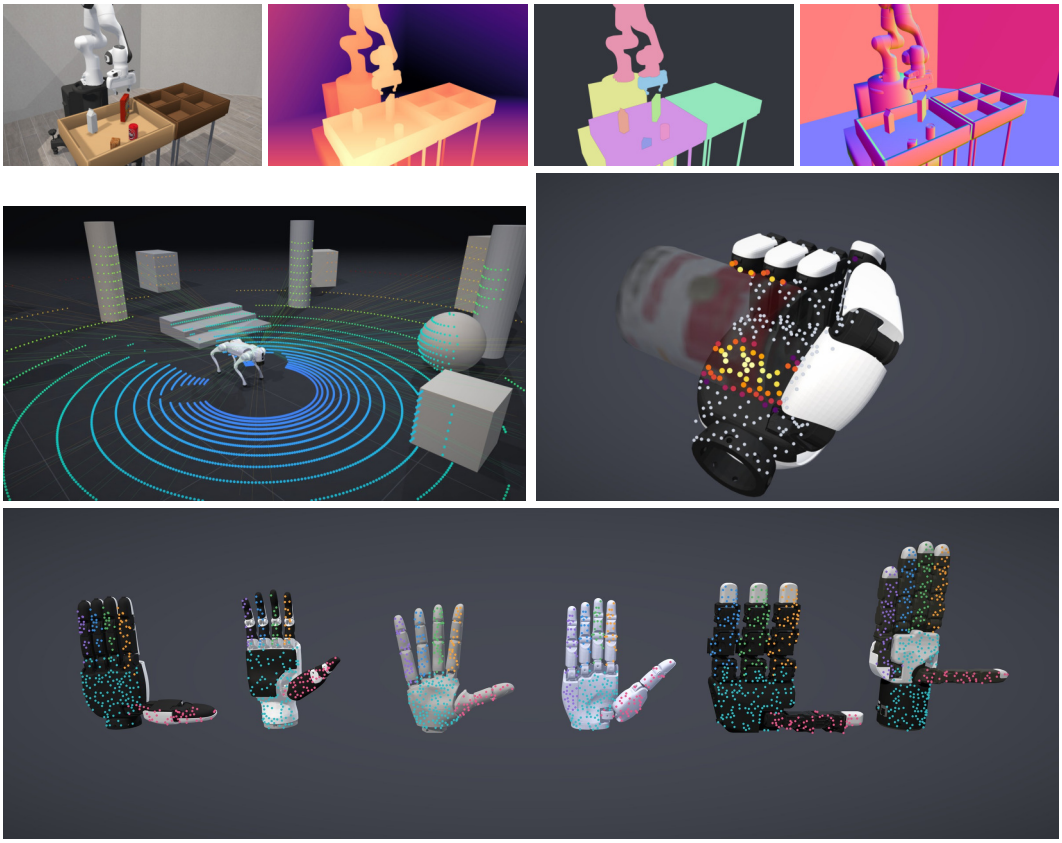


Figure 8: Sensors: RGB, depth, segmentation and normal cameras; LiDAR; tactile taxels.

Capability	Evaluated release
Registered terms	227
Native task configurations	44
Robot embodiments	51
Pretrained-policy adapters	14
Deployment backends	7
Motion-data sources	AMASS, LAFAN1, PHUMA, SEED, DexYCB, ARCTIC, OMOMO, HHI
Teleoperation	GELLO, Meta Quest, PICO, MANUS Gloves, Wuji Gloves, SpaceMouse Gamepad, Keyboard
Learning / data	rsl_rl, rl_games, LeRobot
Physics exposed from Genesis	Rigid, MPM, SPH, PBD, FEM, IPC coupling
Physics implemented for Eden	Mochi (Appendix G), ComFree

Table 5: **Scope of the evaluated Eden release.** These counts and capabilities document the breadth over which the abstractions in the main paper are used; they are not separate contributions or benchmark claims.

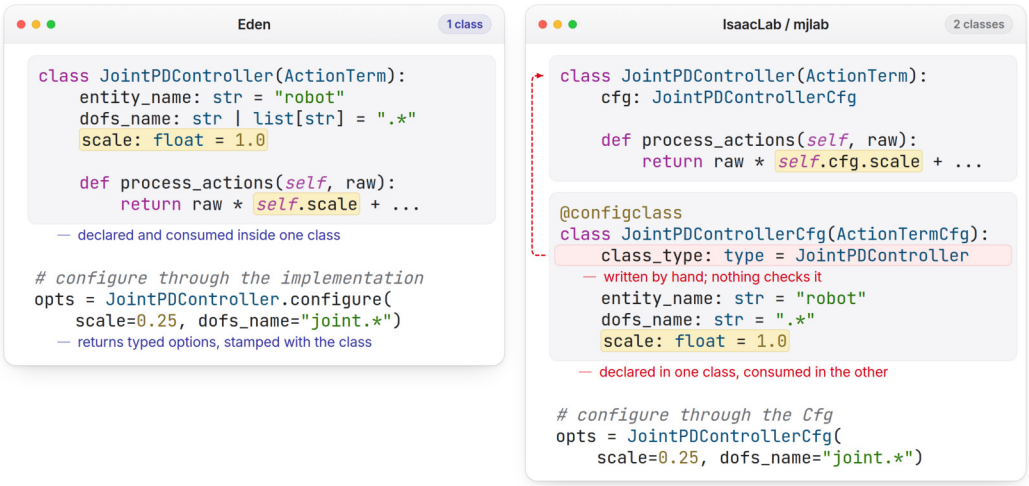


Figure 9: **Term configuration in Eden and in the separate-class design.** *Left*: in Eden, the term declares its own parameters and defaults, and `configure()` derives a typed, serializable options object from it. *Right*: in IsaacLab and mjlab, a term's implementation and its configuration class are written separately and linked by a `class_type` pointer, so a change to a parameter or default must be made in both.

B Configuration Without Drift

Eden, IsaacLab (Mittal et al., 2023), and mjlab (Zakka et al., 2026) all use manager-based environments composed from reusable terms. The distinction we make is not that Eden requires less configuration code—IsaacLab's `@configclass`, for example, is already compact—but where the configuration schema comes from. In IsaacLab and mjlab, a term's implementation and configuration are separate classes linked by a `class_type` reference. In Eden, the implementation itself defines the schema from which `configure(...)` constructs a typed options object (Figure 9).

One source for behavior and configuration. A term declares its configurable parameters, types, and defaults on the implementation that consumes them. `configure(...)` derives the corresponding options object from this definition rather than requiring the same fields to be restated in a second class. Renaming a parameter or changing its default therefore changes the configuration in the same place. The purpose of the single-class design is this single source of truth, not reduced verbosity.

Validation before execution. Configuration errors are checked in two stages. Fields that can be validated locally are checked when the options object is constructed; unknown fields are rejected rather than silently stored. References that depend on the assembled scene—such as entity, link, or DOF names—are checked when the environment is built. This catches configuration errors before the first environment step while still allowing terms to refer to objects that do not exist until scene construction.

Configuration remains data. Although its schema is derived from the implementation, the result of `configure(...)` is data rather than a live runtime object. Options can therefore be nested into an `EdenConfig`, inspected, modified, and serialized. Each serialized options object records the concrete component type needed to reconstruct it, so heterogeneous and polymorphic components round-trip without requiring a separate task-specific reconstruction path. This is what allows a saved `EdenConfig` to identify the task that is later loaded for evaluation or deployment.

Trade-off. Deriving schemas dynamically trades some static tooling for a single source of truth. In particular, `configure(**kwargs)` provides weaker static autocomplete and type checking for keyword arguments than a concrete dataclass constructor. Eden instead checks these fields at configuration and build time. We view this as a deliberate trade-off: the design prevents the

implementation and its configuration schema from evolving independently, at the cost of relying more heavily on runtime validation.

C Manager-Based Architecture Details

Eden decomposes the environment step into independently configurable managers, following the paradigm established by IsaacLab (Mittal et al., 2023) and adopted by mjlab (Zakka et al., 2026). Each manager inherits from `ManagerBase` and orchestrates a set of **terms**, registered computation units that inherit from `ManagerTermBase` and implement `compute()`. This appendix gives the step pipeline, the individual managers, the entity system, and the registry; Figure 2 shows how they compose.

C.1 Step Pipeline

Each call to `RLEnvBase.step()` executes the following fixed order, condensed from the twenty phases documented in Eden's `docs/guides/environments.md` (recorder hooks, the opt-in final-observation snapshot, and the evaluation-only command visualization are omitted):

1. **Action.** The *action manager* parses the policy output, slices it into per-term segments, and each `ActionTerm` maps its segment to entity DOFs.
2. **Simulate.** Command terms first advance their references (e.g. a motion clip's frame). The action terms then apply their processed actions (e.g. PD targets) to Genesis entities, and the physics engine advances by d substeps (decimation). Command terms then refresh their cached robot state on the post-physics state so that metric, termination, and reward terms grade the current step.
3. **Metric.** When configured, the *metric manager* evaluates its terms over the post-physics state, accumulating environment-level statistics (e.g. success flags) for logging and curriculum feedback.
4. **Observation tick.** Stateful observation terms (integrators, counters, latches) advance their per-step state exactly once. Observation `compute()` is a read, because it also runs at build, at reset, and for the final-observation snapshot.
5. **Terminate.** The *termination manager* evaluates all termination terms and produces per-environment `terminated` and `timeout` masks.
6. **Reward.** The *reward manager* evaluates all reward terms, applies per-term weights and optional temporal masks, scales by Δt, and reduces to a scalar reward per environment.
7. **Reset.** Environments flagged by the termination manager are reset: the *event manager* executes `reset`-mode terms, and all managers reset their internal buffers for the affected environments; a `post_reset` hook then runs in the declared manager order so terms that depend on a reset scene (e.g. cached reference state) see the final post-reset state.
8. **Curriculum.** When configured, the *curriculum manager* adjusts environment parameters (reward weights, command ranges) based on training progress.
9. **Command.** The *command manager* decrements per-environment timers and resamples expired commands.
10. **Event.** The *event manager* executes `interval`-mode terms based on per-term timers.
11. **Observe.** The *observation manager* evaluates all observation terms, applies noise, clipping, and scaling, updates history buffers, and returns the observation dictionary.

Resets occur *after* reward and termination computation but *before* observation, so returned observations for reset environments reflect the fresh post-reset state while the final reward of the terminated episode is correctly captured.

C.2 Manager Details

Observation manager. Observations are organized into named *groups* (e.g. `policy`, `critic`), each containing observation terms. Each term supports per-term noise injection, clipping, multiplicative scaling, and temporal history stacking with a configurable `history_length`. When all terms in a group are one-dimensional and history-free, the manager uses a pre-allocated concatenation

buffer for zero-copy assembly. Groups may independently enable or disable observation corruption, allowing asymmetric actor-critic architectures.

Reward manager. Each reward term computes a per-environment scalar with an associated `weight`. The manager evaluates all terms in a single vectorized pass, producing a $(\text{num_envs}, \text{num_terms})$ matrix that is element-wise multiplied by weights, optionally masked by temporal windows (`range_s`), and reduced to a per-environment reward scaled by Δt. Per-term episode sums are accumulated for logging and curriculum feedback, and reward groups can be logged as sums.

Action manager. The action manager maintains a routing table mapping slices of the policy output to individual `ActionTerm` instances, each bound to a specific entity and DOF set (specified by name with glob or regex patterns). Built-in terms include implicit and explicit PD controllers, task-space controllers (differential-IK and operational-space), and binary discrete actions; action modifiers (rate limits, low-pass filters, clipping) compose declaratively. The two-phase design, `compute()` for parsing and `apply_actions()` for actuation, lets the physics decimation loop re-apply the same action across substeps.

Event manager. Events trigger in three modes: `startup` (once at initialization), `reset` (when environments reset), or `interval` (periodically during training). Each term declares its mode, an optional priority for execution ordering, and for interval mode a randomized time range between firings.

Command manager. Command terms generate goal signals (target velocities, end-effector poses) resampled on a per-environment timer, decoupling command frequency from episode structure.

Termination manager. Each term is tagged as either a *termination* (early stop) or a *timeout* (episode time limit), and the manager OR-reduces each category separately for correct value bootstrapping at truncation boundaries.

Curriculum manager. Curriculum terms modify environment parameters, typically reward weights or command ranges, based on training progress, enabling stage-based progressions or performance-adaptive adjustments.

Motion manager. The motion manager provides built-in support for motion imitation tasks, loading reference trajectories from motion capture datasets (AMASS, LAFAN1, and others) and filling a structured buffer with target DOF positions, velocities, and link poses. Key features include heterogeneous motion bins with per-bin sampling weights, adaptive sampling via an exponential moving average of per-bin failure rates, and cross-morphology retargeting (Appendix A).

Metric and recorder managers. The metric manager tracks environment-level statistics for logging and curriculum feedback. The recorder manager captures per-step state, actions, and sensor readings at four configurable hook points in the step pipeline (pre-step, post-step, pre-reset, post-reset), enabling trajectory dataset generation for offline learning with conditional export (success-only or stratified) and LeRobot conversion.

C.3 Entity System

Eden provides four entity classes, dispatched from the entity options and material type:

- `RigidEntity`: rigid and articulated bodies with DOF-level control. Exposes joint positions, velocities, link poses, Jacobians, and contact forces; configurable actuator properties (k_p, k_d, armature, force limits) per DOF.
- `ParticleEntity`: particle-based deformable solids, cloth, and fluids simulated with the MPM, SPH, and PBD solvers. Position and rotation are baked into the morph at creation time, so `set_pos`/`set_quat` raise. Material options span MPM elastic, elasto-plastic, and liquid; SPH liquid; and PBD cloth, elastic, liquid, and particle models.

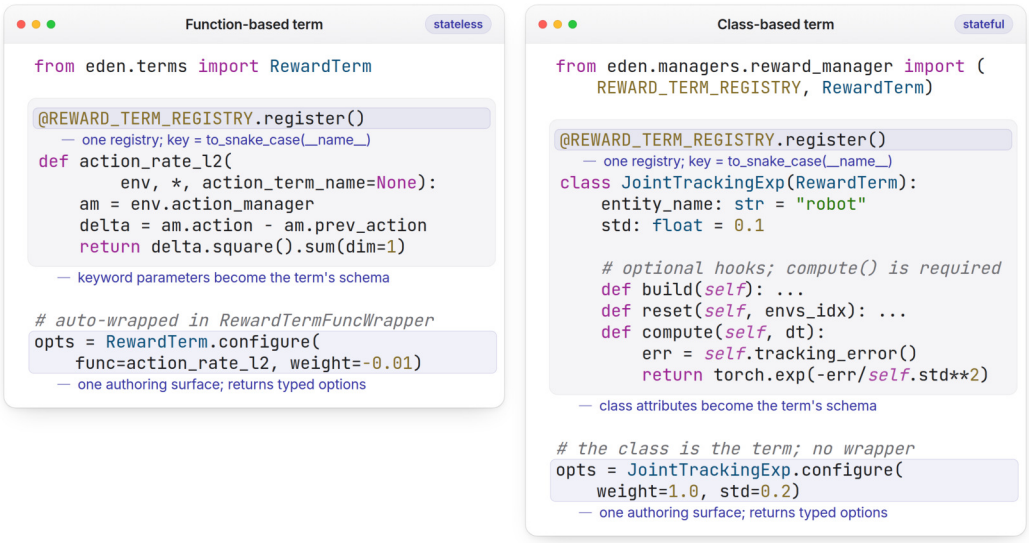


Figure 10: The two term flavors share one registry and one authoring interface. A **function-based** term (left, stateless) is a plain function that, once `@register()`-ed, is wrapped in the manager's `*TermFuncWrapper`; a **class-based** term (right, stateful) subclasses the manager's term base and adds `build`/`reset`/`compute` lifecycle hooks. Both register through the same decorator and are parameterized through the same `.configure()` call, which returns a serializable, type-stamped options object.

- `FEMEntity`: mesh-based finite-element soft bodies (elastic, cloth, muscle) used for the IPC-coupled deformable tasks, with Young's modulus E, Poisson ratio ν, and density ρ exposed.
- `Terrain`: procedurally generated or mesh-based terrain surfaces (a `RigidEntity` specialization) with optional support-surface sampling for contact-aware placement (Appendix A).

Dispatch is by the declared options: `TerrainOptions` yields a `Terrain`, MPM, SPH, or PBD materials a `ParticleEntity`, FEM materials a `FEMEntity`, and everything else a `RigidEntity`. Entity options are declared as named fields on a `SceneOptions` object. Eden discovers all entity options, topologically sorts them by their declared attachment dependencies (so a parent entity is always instantiated before any child attached to it, with cycles reported as errors), instantiates each entity, attaches children to parent links, compiles the scene for GPU-parallel execution, and runs post-build initialization. Cameras and sensors are configured analogously through dedicated `cameras_options` and `sensors_options` fields and wrapped by parallel `Camera` and `Sensor` classes that share the same `ConfigurableMixin` interface but are not subject to material-based dispatch.

C.4 Registry System

Eden maintains one `TermRegistry` per manager (thirteen in total, including the noise-model, observation-modifier, and action-modifier registries). Terms are registered via decorator and can be either function-based (stateless, automatically wrapped in the manager's `*TermFuncWrapper`, a subclass of `ManagerTermFuncWrapperBase`) or class-based (stateful, inheriting from the appropriate term base class with `compute()`, optional `build()`, `reset()`, and `post_reset()`); the registry derives each term's key as `to_snake_case(__name__)`, accepts no alias, and raises on duplicates at import (Figure 10). A term is parameterized via `TermClass.configure(**kwargs)`, the same `ConfigurableMixin` interface used everywhere else (Appendix B), producing serializable option objects that can be reused across tasks. Because the lookup is shared by built-in and user-defined terms alike, registering a term from anywhere in the user's codebase makes it available to the configuration exactly as a library term. Asset, robot, and task registries, which are looked up only by key, do accept an explicit `name=` alias.

D HARDWARE DEPLOYMENT: A CANONICAL INTERFACE AND ENTITY REPLACEMENT

A trained policy is a closed loop over an environment's manager stack: observations are assembled, the network maps them to an action, and the action manager turns that action into PD targets or torques. The usual way to put such a policy on hardware is to re-implement that loop against a vendor SDK, which can reintroduce mismatches in observation and action processing between training and the real robot (mismatched scaling, history stacking, frame conventions, or PD-target computation). Eden instead keeps the managers unchanged and replaces the entity backend, which contains the robot-specific code (SDK session, sensor readout, start-up sequence, motor write). Two mechanisms make this possible: a common state and command interface that every robot driver implements, and *entity replacement*, which lets unmodified observation and action terms read from hardware as if it were the simulator.

State and command interface. Two dataclasses mediate every backend. `RobotState` carries a timestamp, the base orientation quaternion `base_quat` (`wxyz`), body-frame angular velocity and linear acceleration (the IMU channels), per-DOF position/velocity/torque, an optional full-DOF vector, and an `extra` dictionary for backend-specific channels (end-effector pose, gripper width, safety mode). `RobotCommand` carries per-DOF target position, velocity, torque, and PD gains k_p/k_d. Against these structs a backend implements exactly five abstract methods: `connect` and `close` (open/tear down the SDK session), `init_sequence` (the per-robot start ramp), `read_state` $\rightarrow$ `RobotState`, and `send_payload(RobotCommand)`. Everything else (observation construction, the policy forward pass, action post-processing, PD-target synthesis, the command manager, a fixed-rate control-loop limiter, IMU yaw alignment, optional target low-pass filtering, and recording) is implemented once in `DeploymentBase` and shared by all robots.

The shared step loop. `DeploymentBase.step(action)` runs the same pipeline as simulation. It first calls `action_to_payload`, which invokes the environment's *own* `ActionManager.compute` on the policy output and then permutes each action term's processed targets from the term's local DOF order into the backend's DOF order through an index map precomputed at construction, emitting a `RobotCommand` whose k_p/k_d are the robot preset's defaults scaled by a single `deploy_kp_kd_scale` parameter (ramped from 0.2 to 1.0 when first bringing up a policy; the RoboTera XHand firmware keeps its own gains and does not take these). It then calls `send_payload`, sleeps on a `RateLimiter` pinned to the policy's control frequency (`control_dt` $=$ `sim_dt` $\times$ `decimation`), advances the command manager, and reads state back into an observation. Because the action is post-processed by the same `ActionManager` used in training, scaling, clipping, and PD-target computation are bit-for-bit the training path; only the final motor write is robot-specific.

Entity replacement. Observation and action terms never address a robot directly; they read state through the `Entity` getter interface (`get_pos`, `get_quat`, `get_ang`, `get_dofs_pos`, `get_links_pos`, `get_jacobian`, ...). At construction, `DeploymentBase` replaces the simulated entity in `env.entities[name]` with a `RobotStateEntity` wrapper; because terms re-resolve the entity by name each step and cache only derived indices (§3.2), no term holds a stale handle. The wrapper implements the same getter interface: it copies the entity's static schema once (DOF names and indices, default pose and gains, forward vector, fixed-base flag), and each step `update(state)` writes the latest `RobotState` into pre-allocated buffers that the base-link and DOF getters return. Critically, it retains the original simulated entity as a *kinematic model*: link-level getters (`get_links_pos`/`quat`/`vel`/`ang`) and dynamics getters such as `get_jacobian` and `get_mass_mat` delegate to the sim entity *after* `update` has synced its joint configuration from the measured DOFs. Genesis forward kinematics therefore reproduces exactly the link frames and Jacobians the policy was trained against, now evaluated at the real joint angles. No observation or action term is aware it is running on hardware.

Frame alignment and start-up. `reset()` spins `init_sequence` in a rate-limited loop, ramping PD gains and interpolating from the measured pose to the policy's initial reference pose on a legged robot (the interpolation to the reference before the policy starts is what stabilized the first seconds of the G1 deployments in Appendix I), or executing a blocking joint-space move to the home

configuration on an arm, until the operator hands control to the policy. When `auto_yaw_align` is set (the default; a no-op on a fixed-base arm whose base orientation is identity), the wrapper captures the robot's world-frame yaw from `base_quat` and installs a yaw-removal rotation, so world-frame observations live in the yaw-aligned frame the policy saw in training; body-frame quantities are untouched, and the step is disabled when an external estimator supplies a yaw-consistent base. For a fixed-base arm with no IMU, `read_state` reports an identity base orientation and the wrapper holds the training spawn pose.

Resolution and registry. Each backend self-registers into a `DEPLOYMENT_REGISTRY` and declares a robot→backend binding via `bind_deployment(robot_cls, name)`; `resolve_deployment_for` walks the robot options' method-resolution order so DOF variants and subclasses inherit a binding unless they override it. `eden deploy -task <name>` then resolves the backend from the robot bound to the task's scene entity, so the same task configuration that trains a policy also deploys it, with no deployment-specific configuration. Every backend's import is guarded, so a missing vendor SDK disables only that backend and Eden installs identically on workstations without robot SDKs. Seven backends ship with the evaluated release: Unitree G1 (23- and 29-DoF variants), Unitree Go1, Unitree Go2, LimX Oli, Franka Panda/FR3, RoboTera XHand, and a MuJoCo backend that drives the identical loop against a MuJoCo model of the robot for sim-to-sim evaluation (`eden deploy -mujoco`) without re-implementing any term. The UR10e workcell of Table 30 in Appendix I is driven by the fine-tuned VLA's own inference server and is not an Eden deployment backend.

What sharing does and does not remove. For a user moving a task from training to a robot, the shared loop removes three things: the second implementation of observation and action processing, the parity check between that copy and training, and the upkeep of both as the task changes. It leaves four things, all made explicit by the interface. (i) The backend itself: the seven shipped backends are 237–549 non-comment lines (Go1 237, Go2 252, XHand 332, G1 349, Franka 498, LimX Oli 549; the MuJoCo backend 494). (ii) Sensors the robot lacks: a quantity the hardware cannot measure is not zeroed silently; the deployable G1 policies are trained without base linear velocity (one configuration flag removes the term), or `estimate_lin_vel` runs a contact-aided Kalman filter over IMU and leg odometry and feeds the estimate to the same observation term. (iii) Channels and commands: an action term whose command the robot has no port for must be listed in `ignore_action_terms` or construction raises; a training-time command term that needs an operator (a random velocity sampler) is replaced through `command_overrides` by an ordinary command term of the same shape. (iv) The physical gap: sharing the pipeline does not make the simulator match the robot; dynamics mismatch (armature, gains, friction, control rate) diverges a trajectory while the pipeline stays bit-exact, so the DOF mapping, gains, and physical behavior are verified against the MuJoCo sim-to-sim backend before hardware.

Limitations. Entity replacement lowers the maximum control rate in exchange for reusing the training code. Because every step runs the full manager stack (observation assembly, the policy forward pass, and action post-processing) in the Python inference loop, the achievable control frequency is bounded by that per-step overhead rather than by the network alone. The loop is paced to the policy's control rate (50 Hz for the whole-body policies here) and reports every tick that overruns it, so a loop that falls behind is visible rather than silently slower. The G1 trials of Appendix I predate that reporting, so their achieved rate is not reported; the loop is meant for development and bring-up, not for high-rate reactive control. One robot-specific observation: a robot whose onboard controller closes its own PD law once per received packet on its parallel-linkage joints (the LimX Oli's waist; the G1's ankles and, on the 29-DoF build, its waist roll and pitch) must be commanded at a rate well above the policy rate, or the intermittent packets excite the joint. A shared command republisher therefore re-sends the latest target at the wire rate independently of the policy (1 kHz on the Oli, 500 Hz on the G1, following the vendor references); the Go2 backend republishes at 500 Hz for parity with Unitree's own examples, although it has no parallel joints. The intended workflow is to use entity replacement while the policy's inputs and outputs are still changing, since no processing code is reimplemented, and, once they are fixed, to port the loop to a faster implementation without managers where higher control rates are required.

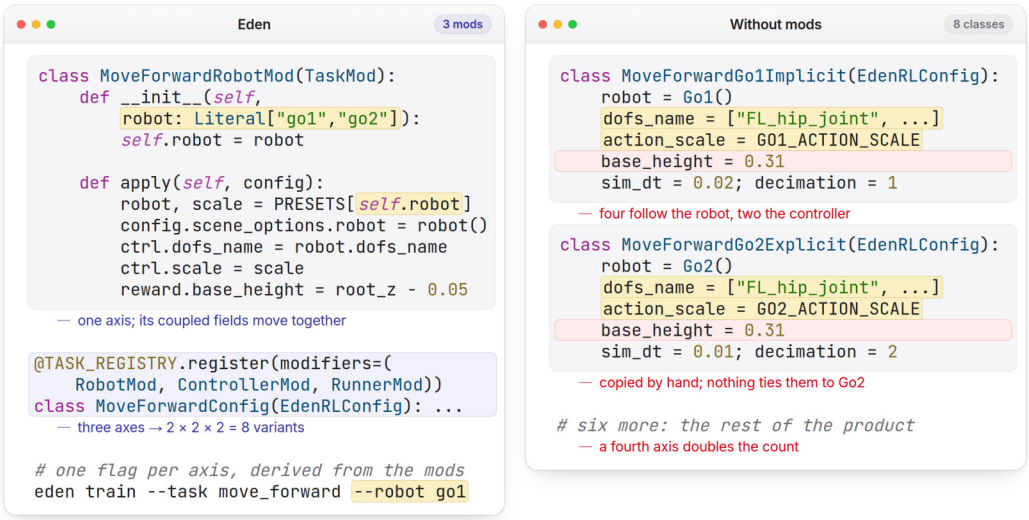


Figure 11: **Task variants compose along independent axes.** *Left*: Eden represents robot, controller, and runner choices as independent typed `TaskMods` applied to one task configuration; three binary axes produce eight variants without duplicating the task. *Right*: representing the same Cartesian product with one class per combination duplicates both the task definition and fields that must change together.

E Composable Task Variants and the CLI

TaskMods. Beyond the inheritance hierarchy of `EdenRLConfig`, Eden exposes *TaskMods*: typed, composable presets that mutate a config at build time. A `TaskMod` is deliberately a plain Python class rather than a Pydantic model, because it is logic with a few parameters, not user-facing data; `EdenConfig` is the data model and the mod operates on it. Its `__init__` declares CLI-tunable parameters as positional-or-keyword arguments with type annotations (optionally `Annotated[T, "help text"]`, forwarded to argparse) and non-CLI constructor state as keyword-only parameters, and it implements `apply(config)` $\rightarrow$ `config`. Mods attach to a task by passing instances to the registration decorator, `@TASK_REGISTRY.register(modifiers=(...))`, so they live on the registry rather than on the config class, and `TASK_REGISTRY.build(name, **kwargs)` applies them in order on a freshly constructed config, so in-place mutation inside `apply` cannot leak across builds. When the same mod class appears twice on a task (a two-arm setup with one robot mod per arm), `.with_prefix("left")` / `.with_prefix("right")` namespaces that instance's flags (`--left_robot`, `--right_robot`). The reference `move_forward` task ships three orthogonal mods, `Robot` (Go1 / Go2), `Controller` (implicit / explicit PD; retimes `sim_dt` and decimation), and `Runner` (rsl_rl / rl_games; rewrites the `runner_options` schema), that combine freely into the full Cartesian product of variants; the velocity-tracking family adds robot-parametric presets and an observation-preset axis. The mod set is discoverable without importing the task through the AST-based registry scan.

CLI. The `eden` entrypoint is a curated command table (not a plugin registry: the set is closed) with twenty subcommands, and it introspects each task's TaskMods to expose them as flags on a single argparse parser. The commands most relevant to this paper: `eden list` (registered tasks and their mod flags), `eden terms` (search the term catalog with `--grep`, list it per kind, or emit JSON), `eden train --task <name> [--mod-flag ...]` (dispatching to rsl_rl or rl_games from the task's `runner_options`; `--config` loads a saved configuration and compares by exact class), `eden inference` (roll out a checkpoint or a registered pre-trained policy), `eden deploy --task <name>` (bind the task to its registered deployment backend, or `--mujoco` for sim-to-sim), `eden smoke [--profile]` (build a task, step it, and attribute per-term cost), `eden preprocess` (build and shard the motion cache), `eden lerobot convert` (recorder output to LeRobot v3), `eden retarget {omniretarget,soma,dex}`, `eden pose gvhmr`, and the teleoperation bridges (`gello`, `metaquest`, `pico`, `wuji`).

F ENGINE-LEVEL THROUGHPUT (NOT A FRAMEWORK CLAIM)

The measurements in this appendix compare simulators, not frameworks: each framework's throughput is its engine's, and Eden owns none of these numbers. They are reported for completeness and because the agentic benchmark of §4.1 reports wall-clock, of which simulation speed is a covariate.

F.1 CROSS-SIMULATOR G1 THROUGHPUT

We benchmark rigid-body throughput on the Unitree G1 (29 DoF). Eden, mjlab (Zakka et al., 2026), Isaac Lab (Mittal et al., 2023), and Isaac Gym (Makoviychuk et al., 2021) share the same robot, per-link convex-mesh collision geometry, and MJX-parity Newton settings; we report mean and sample standard deviation of steady-state throughput over $n = 10$ trials per batch size B (Fig. 12). Eden (Genesis) reaches 1.4 M env-steps/s at $B = 8192$ versus 0.99 M (mjlab, MuJoCo Warp), 0.63 M (Isaac Lab, PhysX), and 0.40 M (Isaac Gym, PhysX).

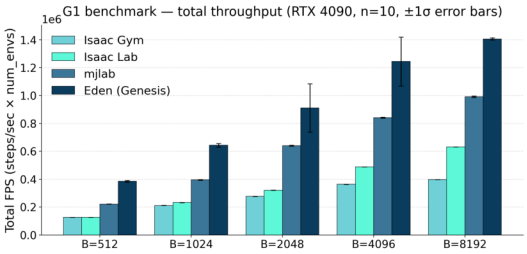


Figure 12: **G1 throughput across simulators (engine-level).** Environment steps/s vs. batch size; bands show ±1 s.d. over $n = 10$ trials on one RTX 4090.

F.2 END-TO-END TRAINING THROUGHPUT ON BEYONDMIMIC

The G1 benchmark above steps physics alone. We also train the BeyondMimic single-clip tracker (Liao et al., 2025) end to end in each framework, so the measured rate includes the full manager stack and the PPO update. IsaacLab runs the upstream BeyondMimic implementation, mjlab its bundled G1 tracking task, and Eden `eden.tasks.beyondmimic`. All three track the same Gangnam Style clip with 4096 environments, a 0.005 s physics step at decimation 4, and PPO with 24 steps per environment per iteration, 5 epochs, and 4 minibatches, on one NVIDIA L40S. The reward terms and weights follow BeyondMimic in all three; the contact penalty is IsaacLab's undesired-contacts term and a self-collision term in mjlab and Eden. Each framework has one run, with its own seed.

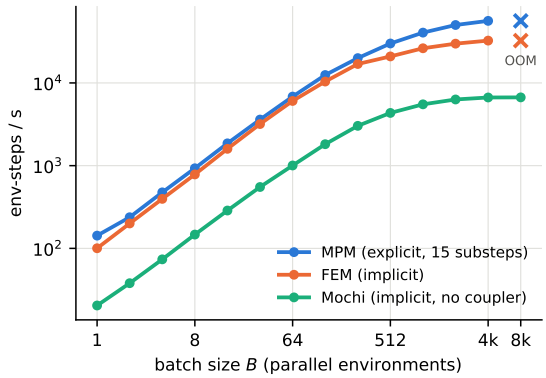
At 15,000 iterations the three reach a similar train reward (Table 6): 40.1 for Eden, 39.3 for IsaacLab, and 37.9 for mjlab. The Eden port therefore reaches the reward of the upstream implementation. Eden trains at 110.6 k env-steps/s, against 105.3 k for mjlab and 75.4 k for IsaacLab, and reaches 15,000 iterations in 3.78 h, against 4.02 h and 5.55 h. The ordering matches the physics-only G1 benchmark, so enabling the full BeyondMimic manager stack does not change it. With one seed per framework, the reward difference is not a ranking.

Framework	Engine	Train reward @ 15k it.	Env-steps/s	Wall-clock to 15k it.
IsaacLab	PhysX	39.3 ± 0.5	75.4 k	5.55 h
mjlab	MuJoCo Warp	37.9 ± 0.5	105.3 k	4.02 h
Eden	Genesis	40.1 ± 0.3	110.6 k	3.78 h

Table 6: **BeyondMimic end-to-end training (engine-level throughput).** Single-clip G1 tracking of the Gangnam Style clip, 4096 environments, identical PPO settings, one run per framework on one NVIDIA L40S. Train reward is the mean episode return over iterations 14,900–14,999 (± s.d. over those iterations). Env-steps/s is the training-loop rate (collection plus PPO update) averaged over the first 15,000 iterations (1.47 B env-steps).

F.3 RIGID–DEFORMABLE COUPLING THROUGHPUT

This benchmark exercises two-way rigid–deformable coupling: a Franka Panda closes its gripper on a 4 cm elastic cube and lifts it. Genesis can simulate the cube with three solvers, and Fig. 13 runs all three on one scene. MPM is explicit MLS-MPM with 560 particles (`grid_density`=128) and 15



B	MPM	FEM	Mochi
1	143	100	20
8	930	783	147
64	6.8 k	6.1 k	1.0 k
512	30.0 k	20.9 k	4.3 k
1,024	40.6 k	26.2 k	5.5 k
2,048	50.1 k	29.8 k	6.3 k
4,096	56.3 k	32.5 k	6.7 k
8,192	OOM	OOM	6.7 k

Figure 13: **Three solvers for the same Franka–elastic-cube grasp (engine-level).** Environment steps/s vs. batch size B on one RTX 4090, mean over $n = 10$ grasp-and-lift episodes ($n = 3$ for Mochi at $B = 8192$; 4k and 8k are 4096 and 8192); bands are ± 1 s.d. (narrower than the lines), $\times$ marks the first B that ran out of GPU memory. Every rate is of a grasp that holds the cube in every environment of every episode. Only the deformable solver changes; the rates are not at matched accuracy (see text).

substeps per frame; FEM is Genesis' implicit FEM solver (linear corotated) with one substep. Each is coupled to the rigid solver by Genesis' legacy coupler. Mochi (Appendix A) integrates the robot and the cube in one implicit Newton solve (stable neo-Hookean, at most 8 Newton iterations per step, one substep) with no coupler. FEM and Mochi integrate the identical 212-vertex, 594-tet mesh. Everything else is shared: the tendon-free Panda MJCF, its PD gains, the grasp and lift joint targets, a 1 N closing force per finger, the cube ($E = 5 \times 10^4$ Pa, $\nu = 0.3$, $\rho = 1000$ kg/m^3, friction 1.0), fp32, and a frame step $\Delta t = 5$ ms. The one difference in control is that Mochi does not clamp drive forces, so its arm runs without the Panda's torque limits. All three run on one Genesis build (the Mochi fork, `6836c9ca`, pinned by Eden `69c18a97`). One process per solver and B runs 10 episodes, each from `scene.reset()`: 100 untimed frames close the gripper, and 200 timed frames lift the hand by 13.5 cm. At the end of each episode the cube's rise is compared with the hand's in every environment: the minimum ratio over all environments and episodes is 0.96 for MPM, 0.64 for FEM, and 0.99 for Mochi, so no environment drops the cube, and the FEM cube slips in the grasp.

MPM is the fastest at every B: 143 env-steps/s at $B = 1$ and 56.3 k at $B = 4096$. FEM runs 100 and 32.5 k. Mochi runs 20 at $B = 1$, a fifth of FEM's rate, and 6.7 k at $B = 4096$, a fifth of FEM's there too. MPM and FEM run out of the 24 GB at $B = 8192$; Mochi still fits at $B = 8192$, but its rate has stopped growing (6.69 k against 6.67 k at $B = 4096$). The rates are not at matched accuracy. At this step Genesis' FEM solver overstates the self-weight sag of a soft block by 306% (Appendix G, finding 8), and the legacy coupler corrects velocities but not positions. The MPM substep (0.33 ms) is 2.1 times the step Genesis suggests for this grid (0.16 ms). Appendix G measures what Mochi's implicit contact costs and buys on the related scenes; this appendix reports rate only.

F.4 Batched RGB rendering throughput

This benchmark measures the camera path. Following ManiSkill3 (Tao et al., 2024), every environment is a Cartpole driven by a random cart force and rendered at every step by either one 640×480 RGB camera or three 320×180 RGB cameras; the rate counts environment steps per second including the render and the read of the image tensor. Both sides load the same Cartpole URDF and use the same environment-local camera poses and 60° vertical field of view. Eden renders through Genesis' batch renderer in rasterizer mode; Isaac Lab (Mittal et al., 2023) renders through its `TiledCamera` sensor on Isaac Sim's RTX renderer. The two renderers are not image-equivalent (a rasterizer against a ray-tracing pipeline), so the comparison is of throughput at the image sizes a policy consumes, not of image quality. The Eden harness drives the Genesis scene directly after `en.init()`, so the manager layer is not on this path. With one 640×480 camera, Eden peaks at 41.0 k env-steps/s at $B = 128$ and runs out of GPU memory at $B = 1024$; Isaac Lab peaks at 2.35 k env-steps/s at

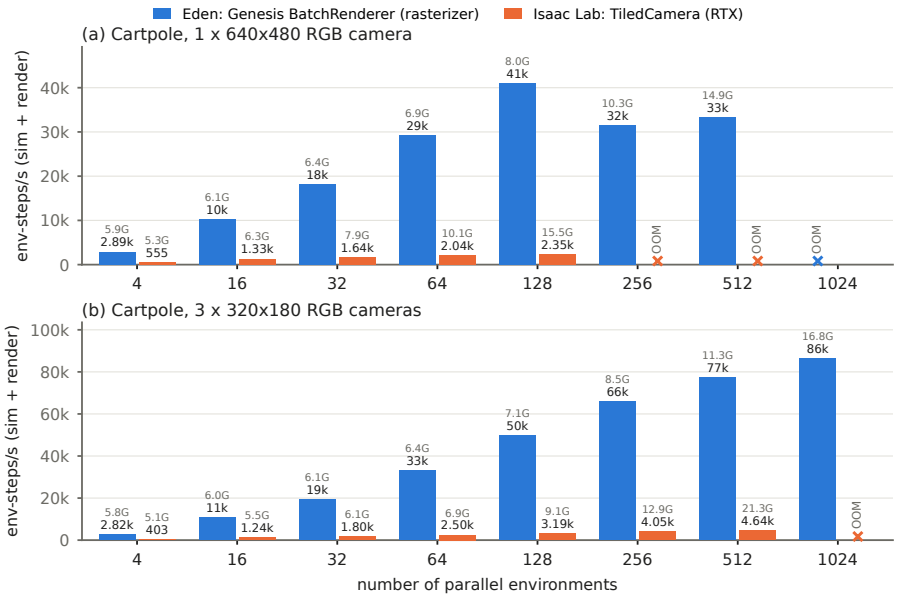


Figure 14: **Batched RGB rendering throughput on Cartpole (engine-level).** Environment steps/s, physics and one RGB render per step, vs. the number of parallel environments, under the protocol of ManiSkill3 (Tao et al., 2024) (their Fig. 2). Each bar is labeled with its rate and its peak GPU memory in GiB (device-wide peak minus idle usage); $\times$ marks a batch size that ran out of GPU memory. Mean of 3 trials on one RTX 4090.

$B = 128$ and runs out of GPU memory from $B = 256$. With three 320×180 cameras, Eden reaches 86.3 k env-steps/s at $B = 1024$ (16.8 GiB) and Isaac Lab 4.64 k env-steps/s at $B = 512$ (21.3 GiB), running out of GPU memory at $B = 1024$ (Fig. 14).

G Engine-Level: Contact Fidelity of Rigid, Mochi, and IPC Solvers

A task configured in Eden can run under three contact and integration stacks whose physical guarantees are ordered in theory. Genesis' constraint-based rigid solver treats penetration as a compliant-constraint modelling choice. The MochiSolver assembles inertia, elasticity, joint drives, and a smooth signed-distance penalty contact into one implicit Newton solve per step, so penetration is bounded by the penalty stiffness. Incremental Potential Contact (IPC, through libuipc (Li et al., 2020)) uses a log-barrier contact energy with continuous collision detection in the line search, so every iterate is intersection-free by construction. This appendix measures whether the observed quality follows that order, what each step costs in wall-clock, and where the order breaks.

Where the three stacks come from, and how a task switches between them. The rigid solver and the IPC coupler are Genesis'. The MochiSolver is ours: a reimplementation in Genesis' Quadrants kernel language of the Mochi physics engine of Project SuperDex (Meta Platforms, Inc., 2026) (C++, Apache 2.0), integrated into Genesis as an add-on solver. The original steps one scene per engine instance and parallelizes environments through Gymnasium vector environments; the reimplementation batches every environment into one solve, so it runs inside Eden's batched environments like Genesis' own solvers. A task selects a stack with two configuration changes and no term change: the entity's material (a Genesis rigid material, the same material with an IPC coupling mode, or a `Mochi*MaterialOptions`) and the matching option on `EnvOptions` (none, `coupler_options = IPCCouplerOptions(...)`, or `mochi_options = MochiOptions(...)`). `EnvOptions` validates the combination at construction: the IPC coupler is rejected with more than one environment, because it has no per-environment reset, and a scene with a Mochi entity may hold only Mochi and kinematic entities.

What is Eden's and what is Genesis'. This is an engine-level measurement. Eden's part is the selection surface above, and the MochiSolver as an add-on; the rigid and IPC physics are Genesis' and libuipc's. The study drives the Genesis scene API those options configure, because its metrics read solver state (libuipc's affine-body buffers) that no Eden term exposes. It does not evaluate trained policies. The MochiSolver is not in upstream Genesis. It is measured on an Eden development branch that adds the MochiSolver, with the Genesis fork that branch pins, which also carries the rigid solver, the FEM solver, and the IPC coupler, so all three tiers run from one build.

Scenes and protocol. Eleven scenes each isolate one mechanism against a ground truth that does not depend on any simulator. The five rigid scenes are a cube drop (rest height), a five-cube stack (zero interface overlap), a Coulomb slide on a 20° incline ($a = g(\sin\theta - \mu\cos\theta)$, $\mu = 0.2$), static friction on the same incline ($\mu = 0.6$, zero displacement), and a 2 cm cube dropped 1.25 m onto a fixed 1 cm plate, which it hits at 5 m/s, the continuous-collision test. The two articulated scenes are an undamped, contact-free three-link pendulum (energy conserved) and a Franka Panda that closes on a 4 cm, 64 g cube and lifts it under an open-loop joint schedule (zero in-hand slip and tilt). The four FEM scenes are a soft block under self-weight (sag $\rho g H^2/2E$ from linear elasticity), a soft ball drop, a rigid cube on a soft slab, and a soft cube on a soft slab (zero overlap in each). No body starts with a velocity, because an IPC-only body does not take an initial velocity from Genesis; motion comes from gravity and, in the grasp, the joint schedule. Resting bodies start 1.5 mm clear of their support, outside IPC's barrier range $\hat{d} = 1$ mm, so no tier starts in contact. Every scene runs at $h \in \{0.5, 1, 2.5, 5, 10, 20\}$ ms (the grasp from 1 ms), in fp64, at batch 1. Four ablations follow: libuipc's default Newton tolerance, Mochi's BDF2 integrator, Mochi's default iteration count, and the CPU backend for Rigid and Mochi. The study comprises 284 cells. 283 completed; the IPC pendulum at $h = 20$ ms is recorded as crashed, because libuipc aborts the process on a NaN energy in its line search. All cells ran on NVIDIA RTX A6000 nodes, one cell per GPU.

One yardstick for every tier. No solver's self-reported contact depth enters a metric. Each tier returns the state it integrated, as rigid-link transforms and soft-body vertex positions, and the same geometric functions measure overlap in that state for all three: box corners against planes, separating-axis box–box distance, and mesh vertices inside boxes and inside deformed tetrahedral meshes. For IPC this state is read from libuipc's own buffers, because Genesis' mirror of an IPC body is not the integrated state (finding 7 below). Genesis' FEM solver splits boundary-only tetrahedra and Mochi does not, which yields 449 against 275 vertices on the same block. Mochi is therefore given the FEM solver's final mesh, so all three tiers integrate identical elements (vertices/tetrahedra: block 449/1425, ball 964/3772, slab 378/1119, soft cube 132/344). Timing covers `scene.step()` alone, synchronised per step, with the first five steps discarded. Table 7 lists the only choices a tier makes.

Findings. *(1) On interpenetration the theoretical order holds without exception.* IPC's peak overlap is exactly zero in all ten contact scenes at every h from 0.5 to 20 ms. Mochi's is 0.35–2.6 mm, bounded by its penalty stiffness, and unlike Rigid's it does not grow with h: it rises by at most 0.04 mm from one h to the next, and on the impact scenes it is largest at the finest step. Rigid's is 0.35–27 mm on the rigid scenes. It is largest at $h = 20$ ms on the stack, the slide, and the static-friction scene, but it peaks at 10 ms on the cube drop and at 0.5 ms on the plate impact. On deformables it is unbounded: at $h = 20$ ms the soft ball's deepest vertex reaches 155 mm below the floor, and the ball comes to rest with its centre below the floor. *(2) IPC pays for zero overlap with a gap.* IPC contacts settle 0.3–1.0 mm apart inside $\hat{d}$, so a resting cube sits 0.83 mm high. Rigid is closest to the exact contact height (-0.004 mm). At $h = 5$ ms Mochi overlaps at rest by 0.13–0.58 mm on the rigid scenes and 0.33–1.4 mm on the FEM scenes, and the Franka's fingers rest 1.2 mm into the grasped cube. The light tunnel cube is the exception, resting 0.06 mm clear of the plate, inside Mochi's 1 mm contact threshold. *(3) A smaller step does not prevent Rigid's contact failures.* On the plate impact the cube ends below the plate at $h = 20$ ms, by discrete tunnelling of about 100 mm per step. It also ends below the plate at $h = 0.5$ and 1 ms. There the compliant contact (time constant 10 ms) lets the cube sink more than 15 mm, half the combined 3 cm of plate and cube, and past that midpoint the contact stops resisting: the cube leaves the plate's underside at 1.3–1.4 m/s and falls on under gravity. At 2.5 ms it stops at 14 mm, and at 5 and 10 ms it rebounds from its first contact step. Only the 2.5–10 ms runs survive. *(4) On friction the order reverses.* At $h = 5$ ms the slide-acceleration error is 0.02% for Rigid, 0.07% for Mochi, and 5.2% for IPC (11% at 10 ms). Over 1.8 s of static friction Rigid creeps 1.5 mm at $h \leq 5$ ms (2.9 and 5.7 mm at 10 and 20 ms), and Mochi and IPC 6.1–6.7 mm at every h;

Table 7: The three solver stacks. All run from one Genesis build (the fork pinned by Eden `feat/mochi-solver`), in fp64, at batch 1. A tier may choose only the rows below; every physical parameter (geometry, density, stiffness, friction, gravity, control schedule) is one shared specification. "Rigid" names the default Genesis stack, so on the FEM scenes it is Genesis' FEM solver coupled to the rigid solver by the default legacy coupler.

	Rigid	Mochi	IPC
rigid contact	constraint solver, compliant contact, time constant $\max(0.01, 2h)$ (default)	smooth SDF penalty, 10^9 Pa/m, 1 mm threshold (default)	libuipc barrier, $\hat{d} = 1$ mm, CCD in the line search
integration	Genesis approximate implicit-fast	one implicit Newton solve per step (backward Euler, ≤ 8 iterations)	implicit Newton, tolerance 10^{-4} (libuipc default 5×10^{-2})
rigid bodies	reduced coordinates	reduced coordinates	affine bodies ($\kappa = 100$ MPa)
articulation	reduced coordinates	reduced coordinates	Genesis predicts, IPC corrects (external articulation)
FEM	Genesis FEMSolver, implicit, corotated linear*	Mochi tetrahedral FEM, stable Neo-Hookean	libuipc, stable Neo-Hookean
FEM–ground	FEM solver floor boundary	penalty	barrier
FEM–rigid	legacy coupler: FEM surface-vertex velocity projected against the rigid SDF, opposite impulse to the rigid link; no position correction	penalty, same Newton solve	barrier, same Newton solve
FEM–FEM	none	penalty	barrier

*The implicit FEMSolver has no Hessian for stable Neo-Hookean. Both models reduce to linear elasticity at small strain, which covers the scene with an exact elastic answer (`fem_rest`, peak strain $\rho g H/E \approx 2\%$).

Table 8: Headline quantity against its exact answer, and the peak overlap anywhere in the run (value / overlap in mm; the pendulum has no contact), at $h = 5$ ms. Overlap is measured the same way for every tier from the state that tier integrated (IPC's own affine-body and FEM buffers, not Genesis' mirror of them). All three tiers integrate identical tetrahedral meshes. Every rung of the 0.5–20 ms ladder is in Figure 15.

scene	quantity	exact	Rigid	Mochi	IPC
`rigid_drop`	rest height error [mm]	0	-0.00427 / 6.7	-0.134 / 1.2	0.832 / 0
`rigid_stack`	worst box-box overlap [mm]	0	1.76 / 1.8	0.603 / 0.89	0 / 0
`rigid_slide`	slide accel. error [%]	0	-0.0206 / 1.6	-0.0678 / 0.43	-5.21 / 0
`rigid_stick`	creep (exact 0) [mm]	0	1.46 / 0.8	6.74 / 0.62	6.71 / 0
`rigid_tunnel`	final height above plate [mm]	≥ 0	-0.00427 / 13	0.0605 / 1.1	0.977 / 0
`artic_pendulum`	energy drift [% of peak KE]	0	-9.68 / –	-78.5 / –	-1.17 / –
`artic_grasp`	in-hand slip [mm]	0	3.82 / 0.95	2.95 / 1.3	0.706 / 0
`fem_rest`	sag error vs linear elasticity [%]	0	306 / 2.2	-1.53 / 0.52	-20.1 / 0
`fem_drop`	peak ground penetration [mm]	0	57.5 / 57	2.01 / 2	0 / 0
`fem_rigid`	peak rigid-soft overlap [mm]	0	3.06 / 3.1	1.89 / 1.9	0 / 0
`fem_fem`	peak soft-soft overlap [mm]	0	39.1 / 39	2.23 / 2.2	0 / 0

both regularise Coulomb friction below 0.01 m/s by default. *(5) libuipc's default Newton tolerance produces a false stick.* At 5×10^{-2} the $\mu = 0.2$ cube is correct at 1 ms, but its acceleration falls 82% short at 5 ms and it sticks at 10 ms (Table 10). The main IPC tier therefore runs at 10^{-4}. *(6) IPC's articulated path does not reproduce articulated dynamics.* On the pendulum at $t = 0.1$ s, Rigid's and Mochi's tip positions at 0.5 ms agree to 0.04 mm, and each converges towards the other as h shrinks. IPC's tip is 6.9–9.3 mm from them at every h and approaches 6.9 mm, not zero, as h shrinks. Its 1.2% energy drift at 5 ms (Table 8) is not conservation: across the ladder the drift ranges from -26% to $+30\%$, and within the 5 ms run the energy deviates by up to 35% of the peak kinetic energy. Contact through the same path works: IPC has the best grasp, with 0.19 mm slip and 0.5° tilt at 1 ms, against 3.0 mm / 13.6° for Mochi and 3.8 mm / 7.4° for Rigid. *(7) Genesis' mirror of an IPC body is not the integrated state.* After the coupler copies IPC's pose into Genesis, the rigid solver still applies one gravity step, so `get_pos()` on an IPC-only body reads gh^2 low (0.25 mm at 5 ms). That offset is indistinguishable from penetration. On articulated bodies the mirror is 0.7–79 mm off, most at coarse

Table 9: Real-time factor at $h = 5$ ms, batch 1, each tier on its faster Genesis backend. On the rigid and articulated scenes the CPU backend is 1.8–2.5× faster than the GPU backend for Rigid and 4–63× for Mochi, and its trajectories match the GPU's to 2×10^{-8}. On the FEM scenes the GPU backend is faster except for Mochi on `fem_rigid`, and Genesis' FEM solver itself does not reproduce across backends (vertex positions differ by up to 0.1 m). libuipc runs on CUDA regardless of the Genesis backend. Timing covers `scene.step()` alone and is synchronised per step. Run-to-run noise is about ±15%.

scene	Rigid	Mochi	IPC
`rigid_drop`	12 (CPU)	35 (CPU)	1.52 (GPU)
`rigid_stack`	8.68 (CPU)	26.6 (CPU)	0.6 (GPU)
`rigid_slide`	15.3 (CPU)	29.9 (CPU)	0.42 (GPU)
`rigid_stick`	14.4 (CPU)	31.3 (CPU)	0.821 (GPU)
`rigid_tunnel`	6.08 (CPU)	24.4 (CPU)	0.947 (GPU)
`artic_pendulum`	11.8 (CPU)	31 (CPU)	0.24 (GPU)
`artic_grasp`	7 (CPU)	4.57 (CPU)	0.16 (GPU)
`fem_rest`	0.196 (GPU)	1.24 (GPU)	0.879 (GPU)
`fem_drop`	0.203 (GPU)	0.206 (GPU)	0.389 (GPU)
`fem_rigid`	0.203 (GPU)	0.736 (CPU)	0.282 (GPU)
`fem_fem`	0.198 (GPU)	0.507 (GPU)	0.286 (GPU)

Table 10: Ablations. At libuipc's default Newton tolerance (5×10^{-2}, a velocity) the sliding cube is still correct at $h = 1$ ms, but its acceleration falls 82% short at 5 ms and it sticks at 10 ms. Mochi's default 4 Newton iterations under-resolve sustained sliding at $h \geq 5$ ms, and its BDF2 integrator reduces the energy lost by backward Euler.

scene	quantity	variant	$h = 1$ ms	5 ms	10 ms
`rigid_slide`	slide accel. error [%]	IPC (tol. 10^{-4})	-6.3e-06	-5.21	-11.1
		IPC, libuipc default tol.	-0.011	-81.6	-100
`rigid_stick`	creep [mm]	IPC (tol. 10^{-4})	6.71	6.71	6.71
		IPC, libuipc default tol.	5.46	5.46	5.46
`artic_grasp`	in-hand slip [mm]	IPC (tol. 10^{-4})	0.189	0.706	1.07
		IPC, libuipc default tol.	0.181	0.69	1.06
`artic_pendulum`	energy drift [% of peak KE]	Mochi (BE, 8 it.)	-44.7	-78.5	-97.9
		Mochi, BDF2	-31.8	-56.8	-66.3
`rigid_slide`	slide accel. error [%]	Mochi (BE, 8 it.)	-0.00249	-0.0678	0.0378
		Mochi, default 4 it.	-0.00249	-8.26	-29.2

h. (8) The default Rigid stack is not a usable deformable baseline. Genesis' FEM solver matches linear elasticity only at 0.5 ms: the sag error is 3% there, 16% at 1 ms, and 306% at 5 ms. On the ball drop it inverts up to 1,270 elements. The legacy coupler projects each FEM surface vertex's velocity against the rigid SDF and applies the opposite impulse to the rigid link, but it corrects no position, so an overlap that forms is not pushed out; at 10 and 20 ms the rigid cube reaches the floor through the slab. It has no FEM–FEM contact, so at every h the soft cube passes through the soft slab to the floor (a peak cube–slab overlap of 39–40 mm up to 10 ms and 29 mm at 20 ms). Mochi's sag is within 1.5–1.7% of linear elasticity at every h. IPC's is 20% short at every h, with the same elastic parameters and mesh as Mochi; its cause was not isolated. *(9) The backend matters as much as the tier for batch-1 speed.* Mochi on the CPU backend is the fastest tier on every non-FEM scene except the grasp, at 24–35× real time (Table 9). IPC is 23–130× slower than Mochi there. It is competitive only on FEM, where it is the fastest tier on the ball drop and every tier except Mochi on the resting block runs below real time. *(10) Mochi's backward Euler is strongly dissipative.* It loses 78% of the pendulum's peak kinetic energy at 5 ms, and BDF2 reduces that to 57% at no measured cost.

What had to change to run the comparison. Each item is a behaviour a user of these stacks meets. libuipc 0.0.24 reversed its revolute-joint sign convention, and Genesis' IPC coupler, at the pinned fork and upstream, still compensates for the old one. With pyuipc 0.0.28 every revolute joint of an `external_articulation` robot therefore turns the wrong way inside IPC: the IPC-side Panda

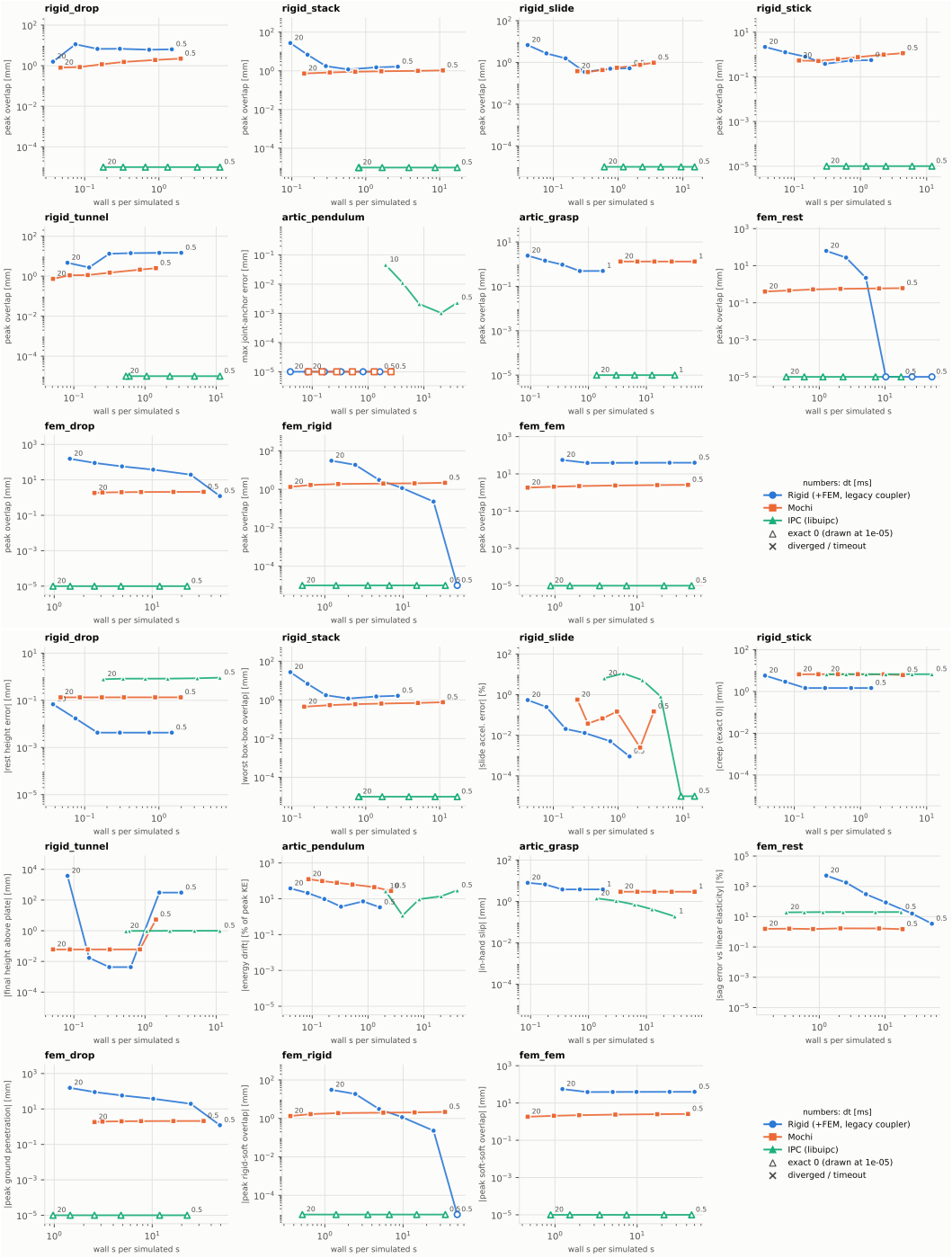


Figure 15: **Accuracy against cost, per scene.** Top: peak interpenetration (the pendulum panel shows the joint-anchor error). Bottom: error against the scene's exact answer. Each line is one tier walking its h ladder; the numbers are h in ms; lower-left is better; exact zeros are drawn hollow at 10^{-5} mm. Genesis GPU backend, so every rung is timed the same way.

hand rises to $z = 0.32$ m while Genesis lowers it to 0.06 m. The study undoes the reversal at runtime. The coupler also refuses `set_qpos` on such a robot and initialises its joint reference to zero, and the Panda's zero pose has interpenetrating fingers, so the start pose is written through the rigid solver

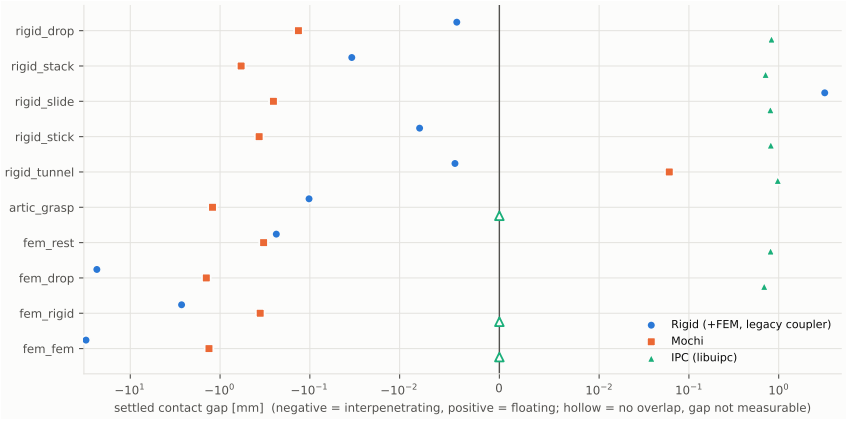


Figure 16: **Settled contact state at $h = 5$ ms.** The signed gap of each scene's primary contact at rest. Negative values are interpenetration and positive values are separation. Hollow markers are one-sided probes (a mesh inside a volume), which report no overlap but cannot measure a gap.

before the coupler builds. The runtime sign correction is part of the measured IPC articulation path. It restores the joint direction, as the grasp shows, but it cannot be excluded as a contributor to finding 6.

Caveats. Batch 1 is not where Rigid's and Mochi's GPU paths are designed to operate, and IPC's coupler does not support per-environment reset, so a batched comparison is a separate benchmark. Each tier is measured at one contact parameterisation (Mochi's default penalty, $\hat{d} = 1$ mm, Rigid's default time constant); the h ladder explores the step, not those knobs. Genesis' SAP coupler, which adds FEM–rigid and FEM–FEM contact to the Rigid tier, was not measured. An earlier run started resting bodies 0.5 mm from their support, inside $\hat{d}$. That pre-compressed IPC's barrier and launched the top cube of the IPC stack 166 mm at $h = 0.5$ ms. Every rest-contact scene was rerun at 1.5 mm for all tiers; only the rerun is reported. Code, raw traces, and every cell are in `eden-benchmark/fidelity_bench/solvers` and `results/fidelity_solvers`; the tables and figures are generated from them and imported unedited by `scripts/import_solver_fidelity.sh`.

H Task Specifications

We document a selection of the task configurations counted in Table 5. Each subsection follows the option groups of the config: the scene, the command, the action, observation, reward and termination managers, and the events and curricula. Table 11 gives the registered name that builds each task. The task figures show four-frame montages of task execution.

Conventions shared by the native tasks. Physics is the Genesis rigid-body solver (Authors, 2024). The control period Δ_c is 0.02 s (50 Hz) on nine tasks and 0.05 s (20 Hz) on in-hand reorientation and bimanual piano. The default action term maps $a \in \mathbb{R}^A$ to joint position targets $q^\star = q_0 + s \odot a$, where q_0 is the default pose and s a per-joint scale, and a PD controller tracks $q^\star$; a task that deviates says so. The four humanoid tasks and quadruped handstand set their gains as in BeyondMimic (Liao et al., 2025), $k_p = I\omega_n^2$ and $k_d = 2\zeta I\omega_n$, from the reflected joint inertia I, a natural frequency ω_n and damping ratio $\zeta = 2$, and derive the scale as $s_j = 0.25\,\tau_{\max,j}/k_{p,j}$.

The reward manager returns $r_t = \Delta_c \sum_i w_i\, r_i$ over the terms in each task's table; a negative w_i marks a penalty. A tracking term has the form $\exp(-e/\sigma^2)$ when the error e is squared and $\exp(-e/\sigma)$ when it is not.

Actor and critic read separate observation groups on every task but piano and extreme parkour, where one group serves both. Noise corrupts the actor group; unless a term says otherwise it is drawn from $\mathcal{U}[-n, n]$, and each term lists its half-width n. The critic group repeats the actor's quantities without noise and appends privileged state. Quadruped handstand is the exception: its critic copies of the

actor terms are noisy as well. Frame stacking, where used, applies to the actor group alone, except on in-hand reorientation.

Table 11: **The Twelve tasks.** The second column is the name given to `eden train -task`, followed by the TaskMod flag that picks the embodiment when it is not the default. A counts actions. Observation sizes are the full input of each group, stacked history included; the vision task also feeds a 72×128 RGB image to both groups. Episodes are measured in control steps. ‡Trained by the project example, not `eden train` (Appendices H.4 and H.12); the parkour student is distilled without a critic, and its depth input is a 58×87 frame.

	registered task	embodiment	A	actor	critic	rate	episode
Velocity tracking	`velocity_tracking_terrain -robot oli`	LimX Oli humanoid	31	1445	292	50 Hz	1000
Motion tracking	`beyondmimic`	Unitree G1 humanoid	29	160	286	50 Hz	500
Multi-clip motion tracking	`general_mimic`	Unitree G1 humanoid	29	428	554	50 Hz	250
Hand-object tracking	`contrack`‡	UFACTORY xArm7 + XHand	19	228	228	50 Hz	255
Humanoid skateboarding	`husky`	Unitree G1 humanoid (23 DoF)	23	395	215	50 Hz	1000
Dexterous lifting	`DexLift`	Franka Panda + XHand	18	84	84	50 Hz	300
In-hand reorientation	`reorient`	Wuji Hand 1	20	207	403	20 Hz	1000
Bimanual piano	`piano -song chpn-p7.mid`	two Wuji Hand 2, floating wrists	52	1212	1212	20 Hz	761
Vision-based pick-up	`LiftYCBRGB`	Franka Panda + Robotiq 2F-85	7	33 + RGB	40 + RGB	50 Hz	300
Quadruped handstand	`handstand`	Unitree Go1	12	45	93	50 Hz	500
Extreme parkour, teacher	`extreme_parkour_teacher`‡	Unitree Go2	12	753	753	50 Hz	1000
Extreme parkour, student	`extreme_parkour_student`‡	Unitree Go2 + depth camera	12	753 + depth	—	50 Hz	1000

H.1 VELOCITY TRACKING

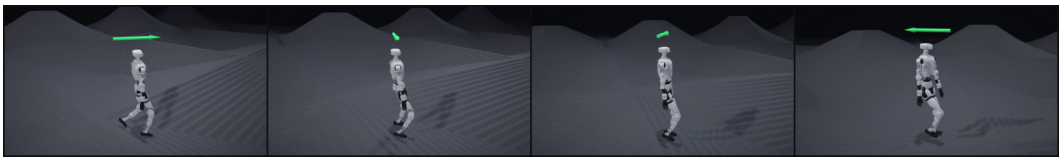


Figure 17: **Velocity tracking.** The humanoid follows four scripted velocity commands in turn across rough terrain; the floating arrow shows the current command.

Scene. The robot is a LimX Oli with 31 actuated joints: 6 in each leg, 7 in each arm, 3 in the waist and 2 in the neck. The ground is a 10×10 grid of 8×8 m terrain tiles. Each column holds one terrain type (flat, pyramid stairs, inverted stairs, pyramid slope, inverted slope, uniform height noise, waves), and each row is harder than the one before. No curriculum moves the robot across rows.

Command. The command $c = (v_x^\star, v_y^\star, \omega_z^\star)$ is a body-frame base velocity, redrawn every $\mathcal{U}[3, 8]$ s from $v_x^\star, v_y^\star \sim \mathcal{U}[-1, 1]$ m/s and $\omega_z^\star \sim \mathcal{U}[-0.5, 0.5]$ rad/s. A tenth of the environments are commanded to stand still. For thirty percent, the yaw rate is replaced by a heading $\psi^\star \sim \mathcal{U}[-\pi, \pi]$, from which $\omega_z^\star = 0.5\,(\psi^\star - \psi)$ is computed and clipped to the same range.

Actions. Joint position targets for all 31 joints, with scales from 0.11 to 0.31 rad and an implicit PD controller at $\omega_n = 2\pi \cdot 5$ Hz. Episodes are 20 s long (1000 steps).

Observations. One actor frame holds the command (3), the base angular velocity (3, $n = 0.2$), gravity expressed in the base frame (3, $n = 0.05$), joint positions as offsets from q_0 (31, $n = 0.01$), joint velocities times 0.05 (31, $n = 1.5$), the last action (31), and terrain heights relative to the base on a 17×11 grid at 0.1 m pitch, times 0.2 (187, $n = 0.1$). Five stacked frames give the actor 1445 inputs. The critic reads a single frame plus the base linear velocity, 292 inputs.

Rewards. Table 12.

Terminations. The time limit; a base tilt beyond 70° from vertical; the base coming within 3 m of the terrain border, which counts as a time-out rather than a failure.

Events and curricula. On reset, the base is displaced by up to 0.5 m and given a random yaw. Pushes arrive every $\mathcal{U}[1, 3]$ s and change the base velocity by at most 0.5 m/s horizontally, 0.4 m/s vertically, 0.52 rad/s about roll and pitch and 0.78 rad/s about yaw. Once per environment at startup, the foot friction is drawn from $\mathcal{U}[0.3, 1.2]$, the torso center of mass is shifted by up to 2.5 cm horizontally and 3 cm vertically, and the joint targets receive a bias of up to 0.015 rad. A command curriculum widens

the ranges twice: at iteration 5000 to $v_x^\star \in [-1.5, 2]$ m/s and $\omega_z^\star \in [-0.7, 0.7]$ rad/s, and at iteration 10000 to $v_x^\star \in [-2, 3]$ m/s.

Table 12: **Reward terms, velocity tracking.** Base velocities v, ω are in the body frame; g^{t} is unit gravity in the torso frame; z_f and v_f are the height and velocity of foot f. The posture width σ_j equals 0.05 on all joints when the command is zero, and lies between 0.08 and 0.6 per joint otherwise. The clearance and slip terms are active only under a nonzero command.

reward term	w_i	r_i
track linear velocity	2.0	$\exp\big(-(\lVert v_{xy}^\star - v_{xy}\rVert^2 + 2v_z^2)/0.25\big)$
track yaw rate	2.0	$\exp\big(-((\omega_z^\star - \omega_z)^2 + 0.05\lVert\omega_{xy}\rVert^2)/0.5\big)$
upright torso	1.0	$\exp\big(-\lVert g_{xy}^{\mathrm{t}}\rVert^2/0.2\big)$
default posture	1.0	$\exp\big(-\frac{1}{A}\sum_j (q_j - q_{0,j})^2/\sigma_j^2\big)$
torso roll/pitch rate	-0.05	$\lVert\omega_{xy}^{\mathrm{t}}\rVert^2$
joint-limit violation	-1.0	summed distance beyond 90% of each joint range
action rate	-0.1	$\lVert a_t - a_{t-1}\rVert^2$
action smoothness	-0.05	$\lVert a_t - a_{t-1}\rVert^2 + \lVert a_t - 2a_{t-1} + a_{t-2}\rVert^2$
joint acceleration	-6.25×10^{-4}	$\lVert\dot{q}_t - \dot{q}_{t-1}\rVert^2$
swing-foot clearance	-2.0	$\sum_f \lvert z_f - 0.125\rvert\, \lVert v_{f,xy}\rVert$
foot slip	-0.1	$\sum_f \lVert v_{f,xy}\rVert^2\, \mathbb{1}[z_f < 0.05]$
self-contact	-1.0	count of self-contacts

H.2 Motion tracking

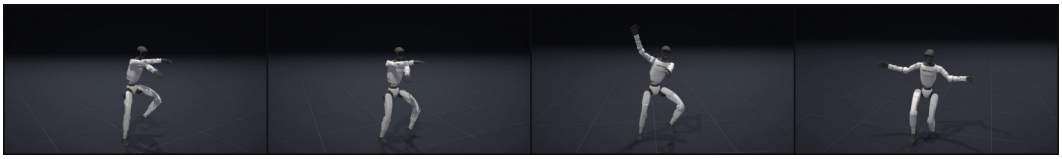


Figure 18: **Motion tracking.** Four poses from one episode in which the humanoid follows a reference dance clip.

Scene. A 29-joint Unitree G1 on flat ground, with self-collision off. The task follows the Beyond-Mimic formulation (Liao et al., 2025).

Command. The command is a single retargeted motion-capture clip: a 32 s dance captured at 60 Hz and resampled to 50 Hz. Episodes begin at a phase drawn uniformly over the clip, in the reference state perturbed by noise. At the end of the clip a new phase is drawn and the robot is reset onto the reference. Fourteen bodies are tracked (pelvis, torso, and both hips, knees, ankles, shoulders, elbows and wrists), with the torso as anchor. Before the relative body errors are taken, the reference bodies are brought to the robot's anchor: they are rotated by the heading difference and shifted to the anchor's horizontal position.

Actions. Joint position targets for all 29 joints, with scales from 0.07 to 0.55 rad, tracked by the explicit PD law $\tau = k_p(q^\star - q) - k_d\dot{q}$ at $\omega_n = 2\pi \cdot 10$ Hz. Episodes are 10 s long (500 steps).

Observations. The actor's 160 inputs are the reference joint positions and velocities ($29 + 29$); the reference anchor position (3, $n = 0.25$) and orientation, as the first two rotation-matrix columns (6, $n = 0.05$), both in the robot's anchor frame; the base linear (3, $n = 0.5$) and angular (3, $n = 0.2$) velocity; joint positions as offsets from q_0 (29, $n = 0.01$) and joint velocities (29, $n = 0.5$); and the last action (29). The critic appends the anchor-relative positions (42) and orientations (84) of the tracked bodies, for 286 inputs.

Rewards. Table 13.

Terminations. The time limit; an anchor height more than 0.25 m from the reference's; a difference above 0.8 m/s^2 between the vertical components of the robot's and the reference's projected gravity, which is not normalized; an ankle or wrist height more than 0.25 m from the reference's.

Events. The reset state is perturbed around the reference: the base pose by up to $5\,\mathrm{cm}$ horizontally, $1\,\mathrm{cm}$ vertically, $0.1\,\mathrm{rad}$ in roll and pitch and $0.2\,\mathrm{rad}$ in yaw; the base velocity by up to $0.5\,\mathrm{m/s}$ horizontally, $0.2\,\mathrm{m/s}$ vertically, $0.52\,\mathrm{rad/s}$ in roll and pitch and $0.78\,\mathrm{rad/s}$ in yaw; each joint by up to $0.1\,\mathrm{rad}$. Velocity pushes with the same bounds occur every $\mathcal{U}[1, 3]\,\mathrm{s}$. At startup each environment draws a foot friction from $\mathcal{U}[0.3, 1.2]$, a pelvis center-of-mass shift of up to $2.5\,\mathrm{cm}$ in x and $5\,\mathrm{cm}$ in y and z, and a joint-target bias of up to $0.01\,\mathrm{rad}$.

Table 13: **Reward terms, motion tracking.** A hat marks the reference; a tilde marks the reference after it is brought to the robot's anchor. p, R, v, ω are world-frame position, orientation and linear and angular velocity, $\angle(\cdot, \cdot)$ is the angle between two orientations, and mean_b averages over the 14 tracked bodies.

reward term	w_i	r_i
anchor position error	0.5	$\exp\big(-\lVert p_\mathrm{a} - \hat{p}_\mathrm{a}\rVert^2/0.3^2\big)$
anchor orientation error	0.5	$\exp\big(-\angle(R_\mathrm{a}, \hat{R}_\mathrm{a})^2/0.4^2\big)$
body positions	1.0	$\exp\big(-\mathrm{mean}_b\lVert p_b - \tilde{p}_b\rVert^2/0.3^2\big)$
body orientations	1.0	$\exp\big(-\mathrm{mean}_b\angle(R_b, \tilde{R}_b)^2/0.4^2\big)$
body linear velocities	1.0	$\exp\big(-\mathrm{mean}_b\lVert v_b - \hat{v}_b\rVert^2/1.0^2\big)$
body angular velocities	1.0	$\exp\big(-\mathrm{mean}_b\lVert \omega_b - \hat{\omega}_b\rVert^2/3.14^2\big)$
action rate	-0.1	$\lVert a_t - a_{t-1}\rVert^2$
joint-limit violation	-10.0	summed distance beyond 90% of each joint range
self-contact	-10.0	self-contacts; zero here, since self-collision is disabled

H.3 Multi-clip motion tracking



Figure 19: **Multi-clip motion tracking.** Four frames from one evaluation episode on the LAFAN1 clip `fightAndSports1_subject1`, at clip times 142.2, 144.0, 145.2 and $146.6\,\mathrm{s}$. The episode starts at $141\,\mathrm{s}$. For the figure the time limit is lifted and the episode is stopped after $7\,\mathrm{s}$ without a termination. The camera follows the robot.

Scene. A 29-joint Unitree G1 on flat ground, with self-collision off, as in motion tracking (Appendix H.2). One policy tracks a library of clips instead of a single clip. The reward, observation and termination terms follow the `g1-bones-deploy` motion tracker of ProtoMotions (Tessler et al., 2025), and the action and randomization setup follows BeyondMimic.

Command. The default library is the 40 clips of LAFAN1 (Harvey et al., 2020) retargeted to the G1, $147\,\mathrm{min}$ captured at $30\,\mathrm{Hz}$ and resampled to $50\,\mathrm{Hz}$. A dataset flag switches to AMASS (Mahmood et al., 2019) retargeted to the G1. Each clip is shifted vertically so that the robot's lowest point in the first frame is at or above the floor. At each reset a clip m is drawn with probability w_m, and a start time within the clip is drawn from $1\,\mathrm{s}$ bins. After each PPO iteration the clip weights are recomputed from the episodes that ended in that iteration. An episode succeeds if the clip plays to its end, fails if a termination fires first, and is dropped if the time limit ends it mid-clip. The failure rate f_m of each clip is capped at 200 times the mean rate, the capped rates are normalized to give w_m, and the result is mixed with the uniform distribution as $w_m \leftarrow 0.9\,w_m + 0.1/N$ for N clips. Within a clip, the weight of a bin is an exponential moving average, with rate 0.001 per control step, of the terminations that occurred in that bin, plus 0.1 divided by the clip's bin count, normalized per clip. The policy sees the reference 1, 2, 4 and 8 frames ahead (20 to $160\,\mathrm{ms}$). Past the end of a clip the reference holds its last frame with zero velocity. The tracked bodies, the anchor and the relative-body transform are those of motion tracking.

Actions. Residual joint targets around the reference, $q^\star = \hat{q} + s \odot a$, where $\hat{q}$ is the reference joint position at the current frame; motion tracking uses q_0 in place of $\hat{q}$. The scales and the explicit PD law are those of motion tracking. Episodes are 5 s long (250 steps).

Observations. The actor's 428 inputs are the 160 of motion tracking plus the reference at the four future frames: joint positions and velocities ($116 + 116$), and anchor position (12, $n = 0.25$) and orientation as the first two rotation-matrix columns (24, $n = 0.05$), both in the robot's current anchor frame. The critic appends the anchor-relative positions (42) and orientations (84) of the tracked bodies, for 554 inputs.

Policies. PPO with 512-256-128 ELU actor and critic MLPs and a state-independent action standard deviation initialized to 1. Each iteration collects 24 steps from each of 4096 environments. The policy shown was trained on two GPUs with 4096 environments each for 42,500 iterations; the success and failure counts are summed over GPUs before each weight update.

Rewards. Table 14.

Terminations. The time limit; an anchor height error above $0.25\, g(\hat{z}_\mathrm{a})$ m; the projected-gravity test of motion tracking with threshold 0.8; an ankle or wrist height error above $0.25\, g(\hat{z}_\mathrm{a})$ m. Here $g(\hat{z}_\mathrm{a}) = 1 + 0.75\,\mathrm{clip}\big((0.5 - \hat{z}_\mathrm{a})/0.5, 0, 1\big)$, where $\hat{z}_\mathrm{a}$ is the reference anchor height in meters, so the height gates widen from 0.25 to 0.44 m as the reference anchor descends from 0.5 m to the floor. The end of a clip does not end the episode.

Events. The reset perturbations, velocity pushes and startup randomization of motion tracking.

Table 14: **Reward terms, multi-clip motion tracking.** Notation as in Table 13. A future term compares the robot's current state with the reference $k \in \{1, 2, 4, 8\}$ frames ahead and averages r_i over the k whose frame lies inside the clip.

reward term	w_i	r_i
anchor position, orientation	0.5, 0.5	as in Table 13
body positions, orientations	1.0, 1.0	as in Table 13
body linear, angular velocities	1.0, 1.0	as in Table 13
future anchor position, orientation	0.25, 0.25	anchor terms against frame k ahead
future body positions, orientations	0.5, 0.5	body pose terms against frame k ahead
action rate	-0.1	$\|a_t - a_{t-1}\|^2$
joint-limit violation	-10.0	as in Table 13
self-contact	-10.0	self-contacts; zero here, since self-collision is disabled

H.4 HAND-OBJECT MOTION TRACKING

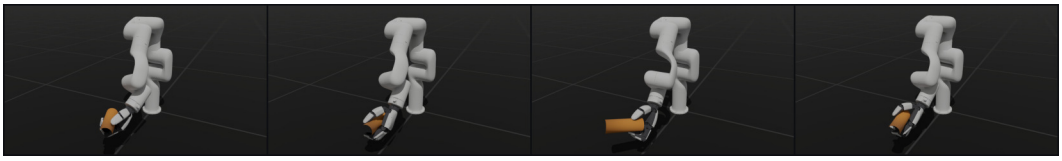


Figure 20: **Hand-object motion tracking.** Four frames from one evaluation episode on the cylinder clip, at reference times 0.44, 1.92, 3.12 and 4.32 s. The episode starts from reference frame 22 and ends at the end of the clip. The hand holds the cylinder off the ground and turns it in the grasp as the reference prescribes. The object is recolored for the figure.

Scene. A port of ConTrack (Liang et al., 2026). The robot is a fixed-base UFACTORY xArm7 with a RobotEra XHand welded to its flange, 7 arm joints and 12 finger joints. Eden builds the rig from its own xArm7 and XHand models, which the weld merges into one kinematic tree. The handedness of the clip selects the left or right hand; the clip shown uses the left. The object is the clip's mesh as a free body with density 200 kg/m^3, here a cylinder 5.5 cm in diameter and 16 cm long (76 g). Friction is 2.0 on the robot and the object, and torsional friction is on. The simulator steps every 5 ms with

two substeps, and each action is held for four steps, so the control rate is 50 Hz instead of the source's 60 Hz.

Command. The command is one reference clip from the ConTrack release. A clip holds the 19 joint positions, the object pose, and a contact flag and an object-frame contact point for each of 15 finger segments. The loader merges the segments into the 10 tracked finger links: each proximal segment stays alone, and the middle and distal segments of a finger, which one link spans, are OR-merged with their points averaged. The clip shown is 100 frames at 20 Hz (5 s), retimed to 50 Hz (248 frames). Poses are interpolated and contacts take the nearest frame. Episodes start at a reference frame drawn by a failure-biased sampler over 0.5 s bins of the clip, which stands in for the source's library of cached simulated reset states. One policy is trained per clip, as in the source.

Actions. Residual joint targets around the reference, $q^\star = \hat{q} + s \odot a$, where $\hat{q}$ is the reference joint position at the current frame. The scale is $s = 0.01$ rad on the arm and 0.25 rad on the fingers, and the runner clips actions to ± 1. An implicit PD controller tracks $q^\star$ with fixed gains, $k_p = 1000$, $k_d = 50$ and a 50 N m effort limit on the arm and $k_p = 100$, $k_d = 10$ and 1 N m on the fingers; the gains are not derived from joint inertia. The time limit is the clip length plus 0.1 s (255 steps), so an episode ends at the clip's end first.

Observations. Actor and critic read identical groups of 228 inputs. No term is noisy, and the critic receives no privileged state. The groups hold the arm and finger joint positions $(7 + 12)$ and velocities $(7 + 12)$; the object position (3), orientation quaternion (4) and linear and angular velocity $(3 + 3)$, in the world frame; and a flag for each tracked finger link in contact with the object (10). The reference part holds the arm and finger joint positions $(42 + 72)$ and the object position (18) and quaternion (24) at the current and next five reference frames, the reference contact flags of the current frame (10) and the clip phase (1).

Policies. The config sets no runner, and `eden train` does not build this task. The project example (`examples/projects/mimic/contrack`) trains it with the source's constrained PPO, installed as a separate package. Each reward term belongs to a task group r_g, a style group r_s or a penalty group r_p (Table 15). A two-headed critic estimates the returns of $r_g + r_p$ and $r_s + r_p$, and the policy gradient uses the advantage $wA_g + (1 - w)A_s$ with $w = \sigma(\lambda)$. After each update λ moves by $0.05\,(\alpha - \hat{J}_g / J_g^\star)$ with $\alpha = 0.9$, where $\hat{J}_g$ is the mean task return of the last 100 episodes and $J_g^\star$ the largest $\hat{J}_g$ of the last 20 updates. Actor and critic are 256-256-128 MLPs.

Rewards. Table 15.

Terminations. The time limit; the end of the clip; an object position error above p_{th} or an orientation error above θ_{th}.

Events and curricula. Each reset writes the reference state at the start frame to the arm, the fingers and the object, without noise. A curriculum sets the object-pose thresholds from an exponential moving average, with rate 0.05 per episode, of the fraction of episodes that end at the clip's end rather than at the pose gate. As the average rises from 0.9 to 0.999 the thresholds tighten linearly from $(p_{\text{th}}, \theta_{\text{th}}) = (0.1\,\text{m}, 1.0\,\text{rad})$ to $(0.05\,\text{m}, 0.5\,\text{rad})$. No physical parameter is randomized.

Table 15: **Reward terms, hand-object motion tracking.** A hat marks the reference at the current frame. p_o, R_o, v_o, ω_o are the object's world-frame position, orientation and linear and angular velocity. mean_j averages over the 7 arm or 12 finger joints, and mean_k over vector components. Contact terms run over the 10 tracked finger links ℓ. c_ℓ and $\hat{c}_\ell$ flag simulated and reference contact of link ℓ with the object, and d_ℓ is the distance between the simulated contact point in the current object frame and the reference point in the reference object frame. Accelerations are finite differences over one control step.

reward term	w_i	r_i
task, r_g		
object position	500	$\exp\big(-\mathrm{mean}_k(p_o-\hat{p}_o)_k^2/0.05^2\big)$
object orientation	500	$\exp\big(-\angle(R_o,\hat{R}_o)^2/0.5^2\big)$
style, r_s		
arm joints	10	$\exp\big(-\mathrm{mean}_j(q_j-\hat{q}_j)^2/0.5^2\big)$
finger joints	10	$\exp\big(-\mathrm{mean}_j(q_j-\hat{q}_j)^2/0.5^2\big)$
contact match	100	$\mathrm{mean}_\ell\,\mathbb{1}[c_\ell \wedge \hat{c}_\ell]$
contact distance	100	mean of $\exp(-d_\ell^2/0.03^2)$ over links with $c_\ell \wedge \hat{c}_\ell$; 0 if there are none
penalty, r_p		
arm joint velocity	-1	$1-\exp\big(-\mathrm{mean}_j \dot{q}_j^2\big)$
finger joint velocity	-1	$1-\exp\big(-\mathrm{mean}_j \dot{q}_j^2\big)$
object linear velocity	-1	$1-\exp\big(-\mathrm{mean}_k v_{o,k}^2\big)$
object angular velocity	-0.1	$1-\exp\big(-\mathrm{mean}_k \omega_{o,k}^2\big)$
arm joint acceleration	-10	$1-\exp\big(-\mathrm{mean}_j \ddot{q}_j^2/50^2\big)$
finger joint acceleration	-10	$1-\exp\big(-\mathrm{mean}_j \ddot{q}_j^2/50^2\big)$
object linear acceleration	-100	$1-\exp\big(-\mathrm{mean}_k \dot{v}_{o,k}^2/50^2\big)$
object angular acceleration	-10	$1-\exp\big(-\mathrm{mean}_k \dot{\omega}_{o,k}^2/50^2\big)$

H.5 HUMANOID SKATEBOARDING

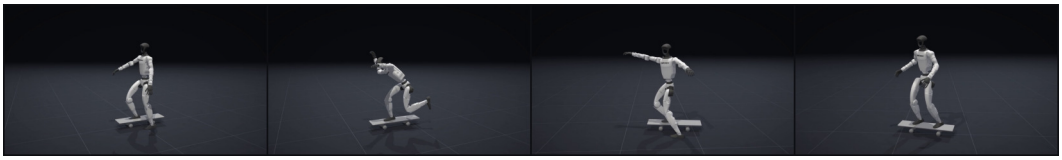


Figure 21: **Humanoid skateboarding.** From left to right: cruising upright, crouching to push, carving, and regaining balance on the board.

Scene. A port of HUSKY (Han et al., 2026). The robot is a Unitree G1 with 23 actuated joints; its wrists are locked. The skateboard is an unactuated free body with seven joints (deck tilt, two truck rolls, four wheels), and an equality constraint ties truck roll to deck tilt, so the board turns when the rider leans. HUSKY adds an adversarial motion prior to shape the push; this config does not, and Table 16 is the whole learning signal.

Command. A target board speed $v^\star \sim \mathcal{U}[0, 1.5]$ m/s, drawn once per episode, and a heading change $\Delta\psi^\star \sim \mathcal{U}[-\pi/4, \pi/4]$ measured from the board. A phase clock with a 6 s period divides each cycle into a push, $[0, 0.4)$, where one foot pushes off the ground; a push-to-steer transition, $[0.4, 0.5)$; a steer, $[0.5, 0.95)$, where both feet stay on the deck and the rider leans into the turn; and a steer-to-push transition, $[0.95, 1)$. Every push-to-steer transition ends with a fresh $\Delta\psi^\star$, anchored to the board's heading at that moment. In both transitions, 14 links follow targets interpolated from where they were when the transition began to recorded push and steer poses. When $v^\star < 0.1$ m/s the clock stops and a stand-still term takes over.

Actions. Joint position targets for the 23 joints, with scales from 0.35 to 0.55 rad and an implicit PD controller at $\omega_n = 2\pi \cdot 10$ Hz. Episodes are 20 s long (1000 steps).

Observations. One actor frame holds the command (2), the pelvis heading (1), the phase (1), the base angular velocity (3, $n = 0.2$), the projected gravity (3, $n = 0.05$), joint positions as offsets from

q_0 (23, $n = 0.01$), joint velocities (23, $n = 1.5$) and the last action (23); five stacked frames give 395 inputs. The critic reads one frame and adds privileged state: base linear velocity, heading error, the board's pose and velocity relative to the pelvis, deck-tilt and truck joint angles, the 14 link targets of the transitions, ground and deck contact forces at each foot, and a one-hot encoding of the contact phase, for 215 inputs.

Rewards. Table 16.

Terminations. The time limit; a pelvis tilt beyond 70°; both feet off the board; contact at the pelvis, a knee, a shoulder, an elbow or a wrist.

Events. Each reset offsets every joint by up to 0.01 rad. At startup each environment draws friction coefficients for the robot from $\mathcal{U}[0.3, 1.6]$, the feet from $\mathcal{U}[0.3, 1.8]$ and the deck from $\mathcal{U}[0.8, 2.0]$; it shifts the torso center of mass by up to 2.5 cm horizontally and 3 cm vertically, and the deck's by up to 2 cm horizontally and 1 cm vertically. A horizontal push of at most 0.5 m/s hits the base every $\mathcal{U}[5, 10]$ s.

Table 16: **Reward terms, humanoid skateboarding.** A phase-specific term is gated by the indicator of its phase. v^{b} is the board's velocity in its own frame, ψ and ψ^{b} the rider's and the board's headings, γ the deck tilt, and γ_{ref} the tilt that would bring the board to $\psi^{\star}$ by the end of the steer phase. The push foot is the one that contacts the ground during the push.

reward term	w_i	r_i
push		
board speed	3.0	$\exp\big(-((v^{\star}-v_x^{\mathrm{b}})^2+(v_y^{\mathrm{b}})^2+(v_z^{\mathrm{b}})^2)/0.25\big)$
rider–board heading	1.0	$\exp\big(-\lvert\psi-\psi^{\mathrm{b}}\rvert/0.25\big)$
push-foot flight time	3.0	$\mathbb{1}[\,0.1 < \text{air time} < 0.5\,\mathrm{s}\,]$
flat push-foot landing	0.5	$\mathbb{1}$[push foot flat on the ground]
steer		
both feet on deck	3.0	$2\cdot\mathbb{1}$[both feet on the deck] $-\,\mathbb{1}$[push foot on the ground]
steering pose	1.5	$\exp\big(-\mathrm{mean}_j(q_j-q_j^{\mathrm{steer}})^2/0.2\big)$
foot markers	1.0	$\exp(-\bar{d}/0.1)$, $\bar{d}$ the mean distance from each foot to its deck marker
board heading	5.0	$\exp\big(-\lvert\psi^{\mathrm{b}}-\psi^{\star}\rvert/0.02\big)$
deck tilt	4.0	$\exp\big(-\lvert\gamma-\gamma_{\mathrm{ref}}\rvert/0.02\big)$
transitions		
link positions	10.0	$\exp\big(-\mathrm{mean}_{\ell}\lVert p_{\ell}-p_{\ell}^{\mathrm{target}}\rVert^2/0.05\big)$
link orientations	10.0	$\exp\big(-\mathrm{mean}_{\ell}\angle(R_{\ell},R_{\ell}^{\mathrm{target}})^2/0.1\big)$
push foot grounded	−0.5	$\mathbb{1}$[push foot on the ground]
every phase		
four wheels down	0.5	$\mathbb{1}$[all four wheels in contact]
level deck	3.0	$\exp(-\lvert\gamma\rvert/0.05)$, except while steering
stand still	1.0	$\exp\big(-\mathrm{mean}_j(q_j-q_{0,j})^2/0.1\big)$, if $v^{\star}<0.1$
joint-limit violation	−5.0	summed distance beyond the soft joint limits
action smoothness	−0.1	$\lVert a_t-a_{t-1}\rVert^2+\lVert a_t-2a_{t-1}+a_{t-2}\rVert^2$
joint velocity	-10^{-3}	$\lVert\dot{q}\rVert^2$
joint acceleration	-6.25×10^{-4}	$\lVert\dot{q}_t-\dot{q}_{t-1}\rVert^2$
joint torque	-10^{-6}	$\lVert\tau\rVert^2$
self-contact	−10.0	$\mathbb{1}$[any self-contact]

H.6 DEXTEROUS LIFTING

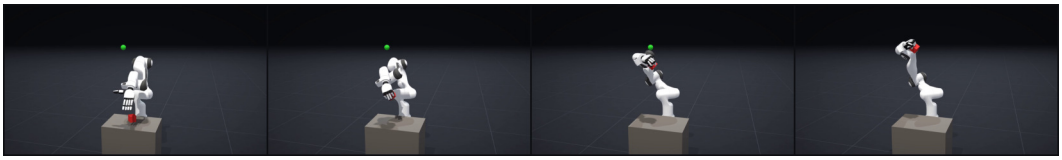


Figure 22: **Dexterous lifting.** The hand closes on the cube, lifts it off the pedestal and raises it to the goal marker.

Scene. The `LiftBox` workcell with its parallel gripper swapped for a 12-DoF left RobotEra XHand, mounted on a 7-DoF Franka Panda. The object is a 6 cm, 43 g cube resting on the table; the task is to lift it to a goal and keep it there.

Command. One goal per episode, drawn in the arm's base frame from $x \in [0.18, 0.40]$, $y \in [-0.15, 0.15]$ and $z \in [0.665, 0.865]$ m. A curriculum grows the sampling box from a single point toward these bounds (see Events and curricula).

Actions. Six actions give an end-effector step of $0.05\tanh(a)$ m in position and $0.1\tanh(a)$ rad in orientation. A damped least-squares solve ($\lambda = 0.1$) turns the step into joint increments, each clipped to 0.2 rad, which the arm's PD controller tracks. The remaining twelve actions set hand joint targets with scale 0.2 rad under $k_p = 50$, $k_d = 5$. Episodes are 6 s long (300 steps).

Observations. One group of 84 inputs serves actor and critic: arm joint positions and velocities ($7 + 7$), hand joint positions and velocities ($12 + 12$), the goal (3, $n = 0.01$) and the cube pose, as a position (3, $n = 0.01$) and a quaternion (4, $n = 0.01$), both expressed relative to the arm base; the mean of the five fingertip positions, used as the grasp point (3); the vector from each fingertip to the cube (15); and the last action (18).

Rewards. Table 17.

Terminations. The time limit; the cube dropping more than 4 cm below the tabletop.

Events and curricula. Resets shift the cube by up to 6 cm in x and 12 cm in y, give it a random yaw, and offset the arm and hand joints by up to 0.05 and 0.02 rad. Physical parameters stay fixed. The goal curriculum tracks success, meaning the cube came within 8 cm of the goal at least once in an episode, averaged over the latest 4096 episodes. Above 0.7 it raises the level by 0.05 times the fraction of episodes that finished; below 0.4 it lowers it by the same amount. A second schedule raises two penalty weights (Table 17).

Table 17: **Reward terms, dexterous lifting.** d is the grasp-point-to-cube distance, h the cube's rise above its resting height and d_g its distance to the goal. The gate $G = \mathbb{1}[h > 0.02]\,\mathbb{1}[d < 0.06]$ enables the goal terms only for a cube that is both raised and held. Arrows give the scheduled weights, which change at iterations 1200 and 1800.

reward term	w_i	r_i
approach, coarse	5.0	$\min(1 - \tanh(d/0.3),\ 0.7)$
approach, fine	3.0	$1 - \tanh(d/0.05)$
fingertip proximity	2.0	$\mathrm{mean}_{\mathrm{tips}} \exp\big(-\lVert p_{\mathrm{tip}} - p_{\mathrm{cube}}\rVert^2 / 0.06^2\big)$
height gained	15.0	$\mathrm{clip}(h, 0, 0.3)$
raised and held	5.0	G
reach goal, coarse	25.0	$G\,(1 - \tanh(d_g/0.3))$
reach goal, fine	10.0	$G\,(1 - \tanh(\max(d_g - 0.02, 0)/0.05))$
arm still at goal	-6.0	$(1 - \tanh(d_g/0.05))\,\lVert \dot{q}_{\mathrm{arm}} \rVert^2$
action rate	$-0.01 \to -0.04 \to -0.1$	$\lVert a_t - a_{t-1} \rVert^2$
arm joint speed	$-0.01 \to -0.2 \to -0.5$	$\sum_j \max(\lvert \dot{q}_j \rvert - 0.5, 0)^2$
joint-limit violation (arm; hand)	-1.0; -1.0	summed distance beyond the soft joint limits

H.7 In-hand reorientation

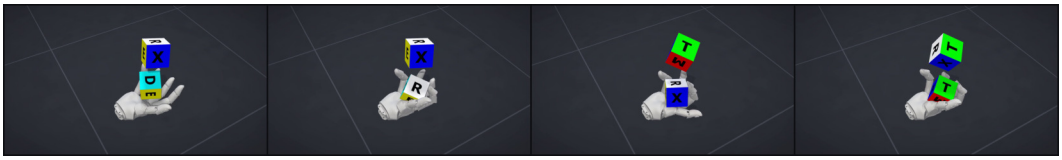


Figure 23: **In-hand reorientation.** Within one episode the hand turns the cube through several distinct orientations without releasing it.

Scene. A right Wuji Hand 1, 20 actuated joints with 4 per finger, fixed palm-up. A free cube of 5.4 cm and 120 g sits on the palm. The `-hand left` TaskMod builds the mirrored task on the left hand.

Command. A stream of target orientations, each drawn uniformly from $SO(3)$ in a wrist-fixed frame. A target is reached once the geodesic error e has stayed under 0.2 rad for five steps in a row; twenty steps after that, the next target is drawn and the cube is not reset.

Actions. Twenty joint offsets from a grasp pose, forming $\mathrm{clip}(q_0 + 0.5\,\mathrm{clip}(a, -1, 1))$ within 90% of each joint range, low-pass filtered as $q_t^\star = 0.5\,q_t^{\mathrm{raw}} + 0.5\,q_{t-1}^\star$. For the first 0.4 s the hand is held at its default pose. The implicit PD controller uses the vendor's per-joint gains, k_p between 0.18 and 0.69 N m/rad. Episodes are 50 s long (1000 steps).

Observations. One actor frame holds joint positions mapped to $[-1, 1]$ over the soft joint limits, i.e. 90% of each range (20, $n = 0.06$), the joint-position error to the current target (20), the cube position in the wrist frame (3, $n = 0.008$ m), the target-to-cube rotation in a 6D encoding (6, zero-mean Gaussian noise with standard deviation 0.05), and the last action (20). With probability 2% per step, the cube pose is overwritten with random values. Three stacked frames give the actor 207 inputs. The critic stacks the same three frames free of noise and adds three frames of joint positions, joint velocities and fingertip positions, plus the cube's linear and angular velocity, the palm orientation, the progress through the current target, the drop counter, the active disturbance and the randomized physical parameters, for 403 inputs.

Rewards. Table 18.

Terminations. The time limit, or the cube staying outside a box around the palm and fingers. A drop counter adds 1 per step outside the box and subtracts 0.5 per step inside it; the episode ends when it hits 10.

Events and curricula. Each reset tilts gravity by a pitch from $\mathcal{U}[-0.4, 0.1]$ rad, which plays the role of wrist pitch; offsets three of the four joints of every finger by $\mathcal{U}[-0.3, -0.1]$ rad; and places the cube within 1 cm of a fixed point above the palm with a uniformly random orientation. At startup, per environment: hand friction from $\mathcal{U}[0.5, 1.0]$; k_p scaled by $\mathcal{U}[0.75, 1.5]$ and k_d by $\mathcal{U}[0.5, 2]$; armature scaled by a log-uniform factor in $[0.75, 1.3]$; joint damping from $\mathcal{U}[0, 0.002]$; a target bias of up to 0.01 rad; hand link masses scaled by $\mathcal{U}[0.4, 1.5]$ and the cube mass by $\mathcal{U}[0.4, 1.6]$. From 3 s into an episode, the cube is kicked in a uniformly random direction every $\mathcal{U}[0.6, 1.8]$ s, at a speed from $\mathcal{U}[0.05, v_{\max}]$ m/s with $v_{\max} = 0.05 + 0.1\,\rho\,\kappa$. The ramp ρ rises linearly from 0 at iteration 250 to 1 at iteration 4000. The adaptive factor $\kappa \in [0.05, 1]$ starts at 0.05, increases after episodes that reach 800 steps and decreases after episodes that end sooner.

Table 18: **Reward terms, in-hand reorientation.** e is the geodesic error to the current target; k counts the steps the cube has remained on target since reaching it.

reward term	w_i	r_i
orientation error	15	1 if $e \le 0.2$, else $\max(0,\ 1 - (e - 0.2)/\pi)$
hold on target	11.4	k while $e < 0.2$, during the 20 steps that follow a reached target
fingertip grip	0.5	$\mathbb{1}$[a distal link touches the cube; neither the palm nor a proximal link does]
grasp pose	−0.2	$\sum_j (q_j - q_{0,j})^2$
action smoothness	−1.0	$\lVert a_t - a_{t-1}\rVert^2 + \lVert a_t - 2a_{t-1} + a_{t-2}\rVert^2$
normalized torque	−24	$\sum_j (\tau_j/\tau_{\max,j})^2$
fingertip slip	−0.3	$\sum_{\text{tips in contact}} \lVert v_{\text{tip}} - v_{\text{cube}}\rVert$
cube outside box	−500	$0.4\,c \cdot \mathbb{1}$[cube outside the box], with c the drop counter
finger–finger contact	−1	count of contacts between fingers

H.8 BIMANUAL DEXTEROUS MANIPULATION



Figure 24: **Bimanual dexterous manipulation.** A handover, from left to right: both hands open above the bar on the table; the robot's left hand lifts the bar; both hands hold it; the left hand has let go and the right hand holds the bar alone.

Scene. A table stands at the center of the scene. A LimX Oli humanoid, fitted with a Sharpa hand on each arm, stands facing the center of the table. A single object is placed on the tabletop with a randomized pose at every reset. Figure 24 shows the two hands handing a bar from one to the other.

Actions. Joint-angle targets for both arms and both hands, $A = 58$: 7 joints in each arm and 22 in each Sharpa hand. The 14 arm actions are joint-angle deltas; the 44 finger actions are absolute joint angles.

Observations for one arm. Observations include the object's current pose and delta to its next goal pose in the palm frame, proprioception states such as the current joint positions and velocities, and object geometry and contact information.

Rewards for one arm. Table 19.

Table 19: **Reward terms for one arm, bimanual dexterous manipulation.** Weights are SimToolReal's per-step scales $\Delta_c w_i$ ($\Delta_c = 1/60$ s). z is the object's rise above its release height, ℓ a latch set once $z > 0.07$ m, and d the largest keypoint distance between the object and goal poses, with $\bar{d}$ its minimum so far. A success is d within tolerance for 10 steps with object pose the goal asks for.

reward term	$\Delta_c w_i$	r_i
lift		
lift height	20	$\mathrm{clip}(z, 0, 0.5)\,(1-\ell)$
lift bonus	300	$\mathbb{1}[\ell$ set this step], once per episode
object goal tracking		
keypoint progress	200	$G\,\ell\,\max(0, \bar{d} - d)$
goal reached	1000	$\mathbb{1}$[success]
contact		
hold or release	0.05	$b\,r_{\mathrm{sup}} + (1-b)\,g\,r_{\mathrm{clr}}$

H.9 BIMANUAL PIANO



Figure 25: **Bimanual piano.** Two Wuji hands play Chopin's Prelude Op. 28 No. 7, keys that are pressed are drawn in green.

Scene. After RoboPianist (Zakka et al., 2023): two Wuji Hand 2 hands, 20 finger joints each, above an 88-key keyboard. No arm is modeled; each hand hangs from a floating wrist actuated in all six axes. A key registers as pressed at 87% of its travel.

Command. The piano roll, i.e. the set of keys that should be held at each step. The `-song` TaskMod loads a MIDI score, here Chopin's Prelude Op. 28 No. 7; without it the task plays a shorter default

score. The prelude has 167 notes on 25 distinct keys and never more than nine keys down together. With 1 s of silence prepended it spans 761 control steps (38 s).

Actions. Twenty-six per hand, 52 in all. Six move the wrist relative to a rest pose above the keys, within 0.04 m forward, 0.03 m up, a lateral travel computed from the score's key span (0.16 m for the right hand and 0.25 m for the left on this piece), and 0.15, 0.15 and 0.30 rad of rotation; a PD controller on the floating base tracks them. Twenty set finger targets $q_{\text{rest}} + 0.5\,a$ under $k_p = 50$, $k_d = 5$.

Observations. One noise-free, unstacked group of 1212 inputs serves both actor and critic: the piano roll at the current and next ten steps (11×88), every key's travel (88), the last action (52), and per hand the finger joint positions relative to rest (20), finger joint velocities times 0.1 (20), the wrist pose relative to rest (6), and the wrist's linear and angular velocity times 0.1 (6).

Rewards. Table 20.

Terminations, events and curricula. The episode ends when the score does. Nothing is randomized and no curriculum runs.

Table 20: **Reward terms, bimanual piano.** The shaping function $g(e; b, m)$ is 1 for $e \le b$ and falls off as a Gaussian in $e - b$, reaching 0.1 at $e = b + m$. $u_k \in [0, 1]$ is key k's travel and $\mathcal{K}_t$ the keys due at step t. The fingertip term pairs each key due in the coming five steps with its nearest fingertip.

reward term	w_i	r_i
correct keys	1	$\frac{1}{2}\,\text{mean}_{k\in\mathcal{K}_t}\, g(1-u_k; 0.05, 1) + \frac{1}{2}\,\mathbb{1}[\text{no key} \notin \mathcal{K}_t \text{ pressed}]$
fingertip over key	1	$\text{mean}_k\, g(d_k; 0.01, 0.1)$, with d_k the nearest fingertip's distance to key k
mechanical power	-5×10^{-4}	$\sum_j \lvert\tau_j \dot{q}_j\rvert$ over finger and wrist joints
distance from rest	-0.02	$\lVert q - q_{\text{rest}}\rVert^2$ plus the squared wrist translation and rotation
action rate	-10^{-3}	$\lVert a_t - a_{t-1}\rVert^2$

H.10 VISION-BASED PICK-UP

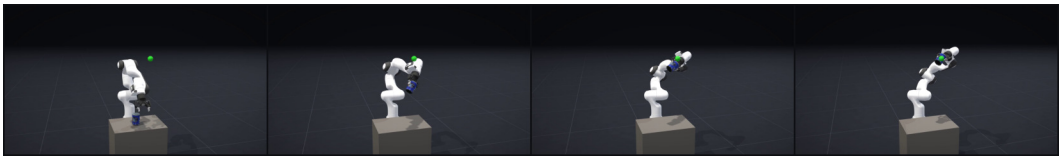


Figure 26: **Vision-based pick-up.** Acting on wrist-camera RGB, the arm grasps a YCB object on the pedestal and carries it to the goal marker.

Scene. A 7-DoF Franka Panda with a Robotiq 2F-85 gripper and a wrist-mounted RGB camera, rendered by the Madrona batch renderer. Each environment is assigned one of eight YCB objects (Calli et al., 2015) for the whole run, with the eight in equal proportion: master chef can, tomato soup can, tuna fish can, pudding box, gelatin box, potted meat can, foam brick, apple. The object's pose is hidden from the actor.

Command. The goal, its curriculum and the penalty schedule are shared with dexterous lifting (Appendix H.6).

Actions. The six end-effector actions of dexterous lifting, plus one gripper action that closes the gripper when positive and opens it otherwise. Episodes are 6 s long (300 steps).

Observations. The actor's 33 vector inputs are the arm joint positions and velocities ($7 + 7$), gripper joint positions (6), the goal (3, $n = 0.01$), the point midway between the finger pads (3) and the last action (7); it also receives a 72×128 RGB frame from the wrist camera, with an $80°$ field of view. The critic sees the same frame and adds the object position and quaternion, for 40 vector inputs. Actor and critic each have their own image encoder: two convolutions (5×5, stride 2, 16 channels; 3×3, stride 2, 32 channels; ELU) and a spatial soft-argmax yielding 32 keypoints (64 features), concatenated with the vector inputs before the MLP.

Rewards. Table 21.

Terminations and events. As in dexterous lifting, without the hand joints. Mass, friction, lighting, textures and the camera pose are fixed.

Table 21: **Reward terms, vision-based pick-up.** Symbols follow Table 17, except that d is measured from the midpoint of the finger pads and h from each object's own resting height. The approach, height, raised-and-held, goal and arm-still terms are those of Table 17, with the same weights, and are omitted here. A grasp term, with $c \in [0, 1]$ the gripper closure, takes the place of the fingertip term.

reward term	w_i	r_i
grasp	2.0	$\mathbb{1}[d < 0.03]\, c$
action rate	$-0.01 \to -0.04 \to -0.1$	$\lVert a_t - a_{t-1} \rVert^2$
arm joint speed	$-0.01 \to -0.2 \to -0.5$	$\sum_j \max(\lvert \dot{q}_j \rvert - 0.5, 0)^2$
joint-limit violation (arm)	-1.0	summed distance beyond the soft joint limits

H.11 QUADRUPED HANDSTAND

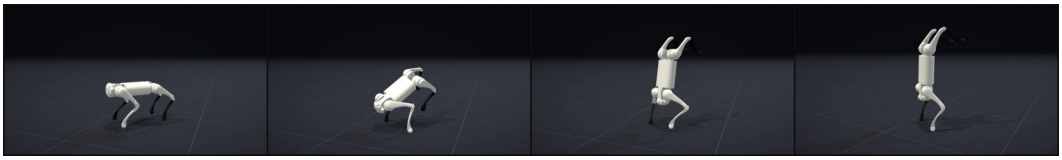


Figure 27: **Quadruped handstand.** Starting on four legs, the quadruped lifts its hind legs and holds an inverted stance on its front legs.

Scene. A port of the MuJoCo Playground Go1 handstand (Zakka et al., 2025): a 12-joint Unitree Go1 standing on flat ground. The goal state is a balance on the front legs with the body's forward axis pointing at the ground and the torso 0.55 m high.

Actions. Actions are clipped to $[-1, 1]$ and integrated into the joint targets, $q^\star_t = \mathrm{clip}(q^\star_{t-1} + 0.3\, a_t)$, where the outer clip is to 90% of each joint range and $q^\star$ restarts from q_0 each episode. An implicit PD controller tracks $q^\star$. Episodes are 10 s long (500 steps).

Observations. The actor's 45 inputs are the base linear (3, $n = 0.1$) and angular (3, $n = 0.2$) velocity, the projected gravity (3, $n = 0.05$), joint positions as offsets from q_0 (12, $n = 0.01$), joint velocities (12, $n = 1.5$) and the last action (12). The critic repeats these 45 inputs, noise included, and adds noise-free body-frame base linear and angular velocity, the world-frame angular velocity, absolute joint positions, joint velocities, joint torques and the base position, for 93 inputs.

Rewards. Table 22.

Terminations. The time limit; the body z-axis tipping past 0.25 below the horizontal, i.e. its world z component dropping under -0.25; a front hip, thigh or calf coming within 5 cm of the ground.

Events. Resets move the base by up to 0.5 m horizontally with a random yaw and draw each base-velocity component from $\mathcal{U}[-0.5, 0.5]$. The dynamics are not randomized; the Playground original randomizes floor friction, joint friction loss, armature, link masses and the torso center of mass.

Table 22: **Reward terms, quadruped handstand.** z is the torso height and e_x the body's forward axis in world coordinates; the pose term covers the six rear-leg joints.

reward term	w_i	r_i
torso height	1.0	$\exp\big(-(0.55-\min(z,0.55))\big)$
nose-down orientation	1.0	$\big(\frac{1}{2}\,e_x\cdot(0,0,-1)+\frac{1}{2}\big)^2$
rear foot low	-0.1	$\mathbb{1}[\text{a rear foot below } 3\,\text{cm}]$
joint-limit violation	-0.5	summed distance beyond 90% of each joint range
rear-leg pose	-0.1	$\sum_{j\in\text{rear}}(q_j-q_{0,j})^2$

H.12 EXTREME PARKOUR

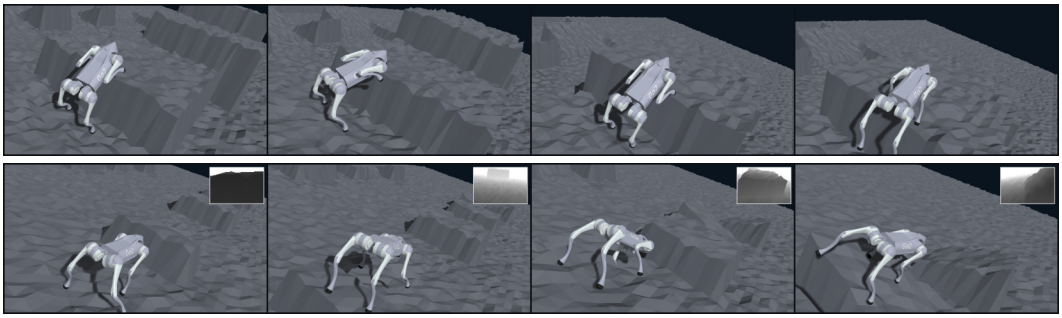


Figure 28: **Extreme parkour.** One evaluation episode per row, frames left to right. Top: the teacher climbing steps 0.45 m high, on row 9. Bottom: the depth student on the demonstration course, on row 9: it approaches the hurdle, has the first step in view, climbs a step and crosses the gap. The inset is the depth frame the student received at that step, with near surfaces dark.

Scene. A port of Extreme Parkour (Cheng et al., 2024) to a 12-joint Unitree Go2 in two phases, each a registered task. The privileged teacher (`extreme_parkour_teacher`) reads a terrain height scan. The student (`extreme_parkour_student`) reads a forward depth image and is distilled from the teacher. The terrain is a 10×20 grid of 16×4 m cells with a 5 mm vertical resolution and a rough surface. Row i sets the difficulty $d=i/9$. Each cell holds one course: gaps $0.1+0.7d$ m wide over a pit 0.2–1.0 m deep, hurdles up to $0.15+0.25d$ m high, steps $0.1+0.35d$ m high, inclined stepping stones over a pit, or a flat lane. Every course places six obstacles along $+x$ and marks eight goal waypoints: the start, one per obstacle and the end. The teacher's columns repeat gap, hurdle, flat, step and stones four times, at a 0.08 m horizontal resolution. The student's grid is 0.1 m. Its row holds 4 gap, 4 hurdle, 1 flat, 4 step and 4 stones columns and 3 columns of a fixed demonstration course: a hurdle, two steps, a gap and further steps, 0.3–0.45 m high. The student also narrows the lateral goal offset of the corridor courses from ±0.4 to ±0.1 m, so the goals stay in the camera's view.

Command and curriculum. The command term holds a forward speed $v_x\sim\mathcal{U}[0.3,0.8]$ m/s, resampled every 6 s, and the goal waypoints of the robot's cell. A goal counts as reached after the base stays within 0.2 m of it for more than 0.1 s. Env k walks one column for the whole run. Its row starts at $\mathcal{U}\{0,1,2\}$. At each reset the row moves up when the robot covered more than $0.8\,v_xT$ from the spawn point and down when it covered less than $0.4\,v_xT$, with $T=20$ s the episode length. A robot that clears the top row restarts at a random row.

Actions. The default position action term with a uniform scale $s=0.25$ drives an implicit PD controller with $k_p=40$ and $k_d=1$, and the torque limits are 35 N m at the hips and 40 N m elsewhere. The gains are not derived from joint inertia. Actions are clipped to ±4.8. The student applies each target one control step (20 ms) late, while its observation carries the undelayed action. Episodes are 20 s long (1000 steps).

Observations. One noise-free group of 753 inputs serves both the teacher's actor and its critic. Its slots are fixed: proprioception (53), the height scan (132), explicit privileged state (9), latent privileged state (29) and a history of ten proprioception frames (530). Proprioception is the base

angular velocity times 0.25 (3), roll and pitch (2), the yaw error to the current and next goal (2), the commanded v_x (1), a one-hot flat-or-obstacle course flag (2), joint positions as offsets from q_0 (12), joint velocities times 0.05 (12), the last action (12) and a latched foot-contact flag minus 0.5 (4), plus three slots held at zero. The scan samples 12×11 terrain heights over $x \in [-0.45, 1.2]$ m and $y \in [-0.75, 0.75]$ m in the base's yaw frame as $\mathrm{clip}(z_{\text{base}} - z_{\text{terrain}} - 0.3, -1, 1)$. The explicit block is the body-frame base velocity times 2, padded with six zeros. The latent block is the added base mass, the base center-of-mass offset, the mean foot friction ratio and the 24 PD gain ratios, which stay at 1 because the gains are not randomized. The history frames zero the yaw-error slots. The scan and the yaw errors refresh every fifth control step.

The student adds two groups. The first is a depth frame from a 106×60 camera with an $87°$ horizontal field of view, mounted on the base at $(0.33, 0, 0.08)$ m and pitched $20°$ down. The batch renderer renders it for all envs at once. The frame is cropped by two rows at the bottom and four columns at each side, clipped to $[0, 2]$ m, resized bicubically to 58×87 and mapped to $d/2 - 0.5$. It refreshes every fifth control step, and the policy receives the previous refresh, a 100 ms latency. The second group is a gate that is 1 where the yaw error to the current goal is below 0.6 rad.

Policies. Neither config sets a runner, and `eden train` does not build these tasks. The project example (`examples/projects/locomotion/extreme_parkour`) trains both phases on the RMA scheme (Kumar et al., 2021) of the source. The teacher actor reads proprioception, a 32-dimensional encoding of the scan, the explicit block and a 20-dimensional latent. At act time an estimator trained alongside the actor replaces the explicit block with its own prediction from proprioception. During rollouts the latent comes from an encoder of the privileged block. A history encoder is regressed onto that latent, and every 20th iteration the history encoder supplies it instead. The critic reads all 753 inputs. The example adds a mirror-symmetry loss that the source does not use. The student replaces the scan encoding with the output of a convolutional encoder on the depth frame followed by a GRU. The same network predicts the goal yaw error, which replaces the true one wherever the gate is 1. Its actor starts as a copy of the teacher's. The student is trained with DAgger (Ross et al., 2011): it drives the environment, and the frozen teacher labels every step. The loss is the distance to the teacher's action plus the yaw-prediction error.

Rewards. Table 23. The student config inherits the teacher's reward terms, but distillation does not read them.

Terminations. The time limit; roll or pitch beyond 1.5 rad; the base dropping below -0.25 m, i.e. into a pit; the last goal reached. Every termination is treated as a truncation, so the value bootstraps at every episode end, as in the source.

Events. Resets place the robot at its cell's spawn point facing $+x$ and scale the default joint positions by $\mathcal{U}[0.95, 1.05]$. At startup the foot friction is scaled by $\mathcal{U}[0.6, 2.0]$, the base mass shifted by $\mathcal{U}[-1, 3]$ kg and the base center of mass by $\mathcal{U}[-0.02, 0.02]$ m per axis. Every 8 s the planar base velocity is set to $\mathcal{U}[-0.5, 0.5]$ m/s per axis. The student also draws a per-env camera pitch offset from $\mathcal{U}[-5°, 5°]$ at startup.

Table 23: **Reward terms, extreme parkour.** $\hat{g}$ is the unit vector from the base to the current goal, ψ_g its heading and ψ the base heading. An obstacle cell is any cell but a flat lane. The last two terms, the uprightness weight and its obstacle-cell scale are not in the source reward set. The example adds them to correct a forward-leaning, limping gait that the port converged to without them.

reward term	w_i	r_i
goal velocity	1.5	$\min(v_{xy} \cdot \hat{g},\, v_x)/v_x$
goal heading	0.5	$\exp(-\lvert\psi_g - \psi\rvert)$
vertical velocity	-1.0	v_z^2 in the body frame, halved on obstacle cells
roll/pitch rate	-0.05	$\omega_x^2 + \omega_y^2$
uprightness	-2.0	$\lVert g_{xy}\rVert^2$ of the projected gravity, times 0.3 on obstacle cells
joint acceleration	-6.25×10^{-4}	$\lVert\dot{q}_t - \dot{q}_{t-1}\rVert^2$
collision	-10.0	base, thigh and calf links with contact force above 0.1 N
action rate	-0.1	$\lVert a_t - a_{t-1}\rVert$
torque rate	-10^{-7}	$\lVert\tau_t - \tau_{t-1}\rVert^2$
torque	-10^{-5}	$\lVert\tau\rVert^2$
hip pose	-0.5	$\sum_{j\in\text{hips}}(q_j - q_{0,j})^2$
joint pose	-0.04	$\lVert q - q_0\rVert^2$
stumble	-1.0	$\mathbb{1}$[a foot's horizontal contact force exceeds $4\times$ its vertical]
foot on edge	-1.0	feet in contact within 5 cm of a terrain edge, from row 4 up
feet air time	1.0	$\sum_{\text{feet}}(t_{\text{air}} - 0.35)$ at touchdown
contact balance	-0.5	variance across feet of each foot's contact duty

H.13 HUMAN-HUMAN INTERACTION



Figure 29: **Human-human interaction.** From left to right: the supporter (orange) bends over the seated recipient (blue), takes hold of them, lifts them to a crouch and holds them once they stand.

Scene. A port of the help-up task of AssistMimic (Shibata et al., 2026). One environment holds two SMPL humanoids on flat ground, a supporter and a recipient, each with 69 actuated joints. The recipient has 0.7 times the supporter's mass, and its lower body is weakened to half the PD gains and an 80 N m effort limit, so it cannot rise without help. The original adds an adversarial motion prior; this config does not, and Table 24 is the whole learning signal.

Command. A paired clip from Inter-X, drawn uniformly from 8 clips at each reset; both humanoids start on its first frame. In six clips the recipient starts on the floor.

Actions. Joint position targets for each humanoid's 69 joints ($A = 138$), following PHC Luo et al. (2023): each body's three actions form a rotation vector scaled by π. The control rate is 30 Hz, and episodes are 10 s long (300 steps).

Observations. Each humanoid reads its own group of 1452 inputs. The first 934 are PHC's proprioception and imitation features. The rest cover the interaction: the partner's body states in the agent's root and wrist frames, wrist contact flags and forces for both agents, the last action and a role flag. Each role has its own actor, initialized from PHC, and one critic serves both.

Rewards. Table 24.

Terminations. The time limit; the end of the clip; either humanoid's mean body distance to its reference exceeding 0.25 m.

Events. Nothing is randomized. Half of the resets start from a stored state of a past high-return rollout instead of the clip's first frame.

Table 24: **Reward terms, human-human and human-human-scene interaction.** The supporter's reward is the mean of its own reward and the recipient's, so recipient terms enter it at half weight. r^{im} is PHC's imitation reward, a weighted sum of exponentials of body position, rotation and velocity errors. d_w is the distance from supporter wrist w to the nearest recipient upper-body joint, and F_w the contact force on that wrist.

reward term	w_i	r_i
supporter		
imitation	0.5	r^{im}
hand contact	0.5	$\sum_w \mathbb{1}[d_w \le 0.4]\,\big(3.75\,e^{-2.5\,d_w}\,\min(e^{\lvert F_w\rvert-1}, 1) + 0.05\big)$
recipient head height	1.0	$\text{clip}(z_{\text{head}}, 0, 2)$
recipient imitation	0.5	r^{im} of the recipient
recipient torque (scene only)	0.25	$\exp\big(-\sum_j \lvert\tau_j^{\text{rc}}\rvert/150\big)$
power	-7.5×10^{-4}	$\sum_j \lvert\tau_j \dot{q}_j\rvert$
recipient power	-10^{-3}	$\sum_j \lvert\tau_j^{\text{rc}} \dot{q}_j^{\text{rc}}\rvert$
recipient		
imitation	1.0	r^{im}
head height	1.0	$\text{clip}(z_{\text{head}}, 0, 2)$
power	-2×10^{-3}	$\sum_j \lvert\tau_j \dot{q}_j\rvert$

H.14 HUMAN-HUMAN-SCENE INTERACTION



Figure 30: **Human-human-scene interaction.** From left to right: the supporter (orange) approaches the recipient (blue) lying on a bed, reaches under their shoulders, raises their trunk and brings them to sit on the edge of the bed.

Scene. AssistMimic's bed task (Shibata et al., 2026): the scene of Appendix H.13 with a fixed $1.9 \times 1.0 \times 0.42$ m bed added. The recipient is weaker than in help-up: its hips are limited to 20 N m and its trunk to 40 N m.

Command. A paired clip from HHI-Assist, drawn uniformly from 10 clips that each start with the recipient lying on the bed.

Actions. As in Appendix H.13.

Observations. As in Appendix H.13, with dynamic reference retargeting. While the pair is within 1.3 m, each of the supporter's hand targets is shifted by the recipient's tracking error at the nearest recipient body, so the supporter follows where the recipient actually is.

Rewards. Table 24, including the recipient-torque term, which pays the supporter for taking the recipient's weight.

Terminations and events. As in Appendix H.13.

H.15 ROBOSUITE MANIPULATION BENCHMARK

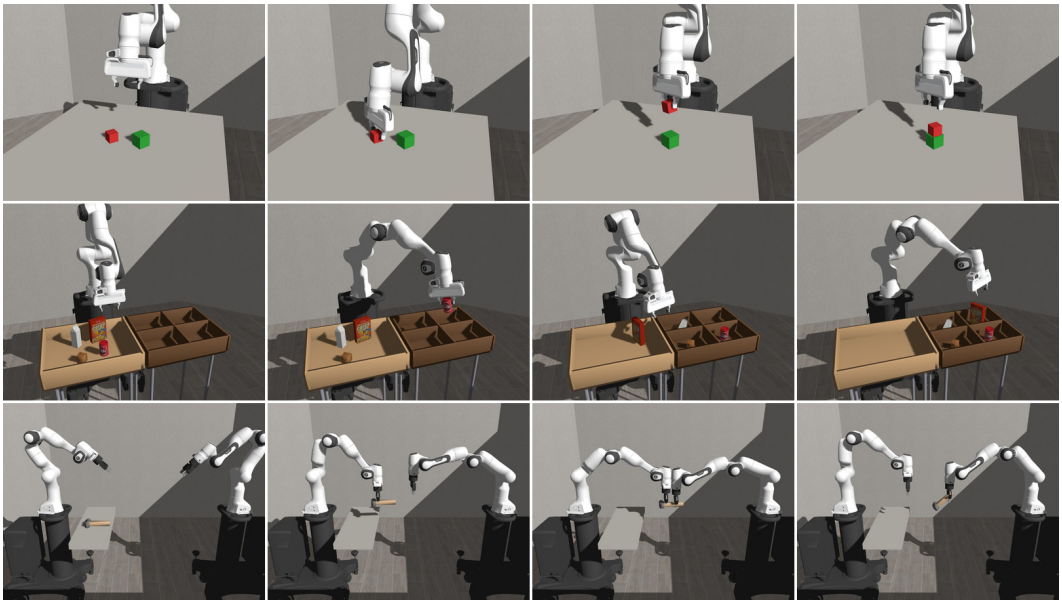


Figure 31: **robosuite port of Eden.** One row per task, four frames each, from one scripted episode that acts only through the task's action space; the tasks ship no trained policy. From top to bottom: `robosuite_stack`, the red cube stacked on the green one; `robosuite_pick_place`, all four objects moved to their quadrants of the target bin; `robosuite_two_arm_handover --robot droid`, the first arm lifting the hammer off the table and carrying it to the midline, both arms holding it, and the second arm holding it alone. Each episode ends with the task's success metric satisfied. The handover uses the DROID rig (Franka arm and Robotiq 2F-85) because the Franka Hand's grip lets the hammer slide out before the second arm can take it. The camera is moved closer than the registered one.

The seven tasks of Table 25 port the manipulation environments of robosuite (Zhu et al., 2020) (`eden.tasks.robosuite`). Unlike SIMPLER and PolaRiS, they are training tasks and register reward terms. Arena geometry, robot placement, object placement ranges, reward multipliers and success predicates take robosuite's values. Each task module lists these values and the deviations described below. Every task takes `--robot`, which selects an arm with a welded two-finger gripper from the composite registry: `panda_franka_hand` (robosuite's default and the default here), `droid`, `ur5e_2f85` or `ur10e_2f85`. A swap replaces the rig in each robot slot of the scene and retargets the two action terms of each arm. No other term names the robot, because each term reads the arm, the gripper, the pad links and the joint lists from the rig in its scene slot when the environment is built. Three tasks take a variant flag for robosuite's variant environments. `--object` and `--nut` keep one object and remove the others' entities and terms, and `--arrangement` places the two arms facing each other (`opposed`) or side by side (`parallel`). Four further robosuite tasks (Door, Wipe, ToolHang, TwoArmTransport) are ported but not yet registered.

Scene. Each arm stands on robosuite's Rethink pedestal with its base 0.912 m above the floor. The single-arm tasks place it 0.56 m behind the table centre, and PickPlace places it at $(-0.5, -0.1)$ m. The table is 0.8×0.8 m with its top at 0.8 m. NutAssembly raises the top to 0.82 m, and TwoArmHandover uses a 0.8×0.3 m table offset 0.45 m toward the first arm. PickPlace replaces the table with two bins, and TwoArmPegInHole has no table. The floor and walls are robosuite's arena mesh, loaded as a kinematic entity that renders but does not collide; an invisible plane provides the ground contact. The mesh objects, the bins, the nuts and the plate are robosuite's MJCF files, and the pot and the hammer are re-authored from robosuite's procedural objects. Each is converted for Genesis when it is loaded. The cubes and the pegs are primitives. The simulator steps every 5 ms with two substeps, and each action is held for ten steps, so the control rate is robosuite's 20 Hz. A rig whose gripper declares a finer substep, such as the Robotiq 2F-85, raises the substep count to five when it is swapped in.

Table 25: **robosuite tasks.** A counts actions and "obs." the size of the actor group; the critic group has the same terms without noise. Every task runs at 20 Hz for 200 steps (10 s). Success is the predicate robosuite checks; "released" means that the gripper's two pads are not both touching the object.

registered task	variant flag	A	obs.	success
`robosuite_lift`	—	7	49	cube centre 4 cm above the table
`robosuite_stack`	—	7	62	4 cm cube lifted, touching the 5 cm cube, released
`robosuite_pick_place`	`--object`	7	79	milk, bread, cereal and can each in its quadrant of the target bin, released
`robosuite_nut_assembly`	`--nut`	7	59	each nut within 3 cm of its peg per horizontal axis, below table + 5 cm, released
`robosuite_two_arm_lift`	`--arrangement`	14	97	pot bottom 10 cm above the table, tilt below 30°
`robosuite_two_arm_peg_in_hole`	`--arrangement`	12	93	hole centre within 6 cm of the peg axis, axial offset in $[-0.12, 0.14]$ m, axis cosine above 0.95
`robosuite_two_arm_handover`	`--arrangement`	14	94	hammer 10 cm above the table, held by the second arm alone

Command. No command term is registered.

Actions. Each arm has 7 actions, as robosuite's `OSC_POSE` controller with a gripper. The first six are an end-effector displacement $\delta = s \odot \tanh(a_{1:6})$, with $s = 5$ cm per unit in position and 0.5 rad per unit in orientation. Damped least squares, $\Delta q = J^\top (JJ^\top + \lambda I)^{-1}\delta$ with $\lambda = 0.1$ and each $|\Delta q_j|$ clamped to 0.3 rad, turns δ into joint offsets, and the arm tracks $q + \Delta q$. The displacement acts at the arm's flange, not at the point between the fingers where robosuite's controller acts. The two points are 10.3 cm apart on the Franka Hand, so a pure rotation moves the fingertips on an arc. The seventh action closes the gripper when positive and opens it otherwise. TwoArmPegInHole has no gripper actions, so $A = 12$. There, each arm starts with its fingers closed on its object (the peg or the plate), which is welded to its hand.

Observations. Each arm contributes robosuite's proprioception: the sine and cosine of the seven joint angles, joint velocities, the position of the midpoint between the two pads, the flange orientation, and the finger positions and velocities (32 values). Each object contributes its position and orientation in the frame of its asset file and the vector from the pad midpoint to it. TwoArmLift and TwoArmHandover add handle points. TwoArmPegInHole adds robosuite's peg–hole quantities: the axial offset, the off-axis distance and the axis cosine. Every task appends the last action. The actor's object-pose terms carry $\mathcal{U}[-0.005, 0.005]$ noise.

Rewards. Table 26. The grasp terms are robosuite's contact test: both pads of one arm touch the object. Genesis reports contacts per entity, not per geometry, so on TwoArmLift a pad counts on any part of the pot, where robosuite requires the arm's own handle.

Terminations and success. Tasks time out after 10 s by default (50 s in robosuite). Except for TwoArmPegInHole, they also terminate if any object falls 0.1 m below its initial supporting surface; robosuite terminates only at the horizon. Success is defined in Table 25.

Events and curricula. At reset each arm's joints are offset by $\mathcal{U}[-0.02, 0.02]$ rad about the home pose. robosuite draws from $\mathcal{N}(0, 0.02^2)$ instead. Objects are placed with robosuite's ranges and a uniform yaw. Lift uses ±3 cm about the table centre, Stack ±8 cm in x, NutAssembly one band per nut on the robot's side of the table, TwoArmLift ±3 cm and 180° ± 60°, and TwoArmHandover ±10 cm by ±5 cm. Where robosuite draws several objects from one region and rejects overlaps, each object here has its own sub-region. Stack keeps its two cubes in separate half-planes, and PickPlace places each object in its own quadrant of the source bin. Object sizes that robosuite resamples at each reset are fixed. For example, the peg radius is 2 cm, where robosuite samples from $[1.5, 3]$ cm. There are no curricula.

Table 26: **robosuite reward terms and weights.** Weights are robosuite's shaping multipliers. robosuite returns the largest stage, or only the success reward once it is reached; Eden sums the terms, which ranks the states in the same order. PickPlace and NutAssembly carry the five terms once per object. "Reach" is $1 - \tanh(k\,d)$ in the distance d from the pad midpoint to the object, with $k = 10$ unless stated.

task	terms (weight)
Lift	reach (1.0), both pads touching (0.25), lifted (2.25)
Stack	reach (0.25), both pads touching (0.25), lifted 4 cm (1.0), $1 - \tanh(d_{xy})$ to the target cube once lifted (0.5), stacked (2.0)
PickPlace, NutAssembly	per object: reach (0.1), both pads touching (0.35), lifted 10 cm (0.5), $1 - \tanh(10\,d_{xy})$ to its target once lifted (0.7), placed (1.0)
TwoArmLift	$\mathrm{clip}(h - 0.05, 0, 0.15)$ in the height h of the pot bottom above the table, while tilted below 30° (10.0); per arm, both pads touching (0.25) and reach to its handle (0.5); success (3.0)
TwoArmPegInHole	$1 - \tanh$ of the peg–hole distance, of the off-axis distance and of the axial offset (1.0 each), axis cosine (1.0), success (1.0)
TwoArmHandover	first arm: reach, $k = 1$ (0.25), pads touching (0.25); hammer lifted (0.5); second arm: reach once lifted, $k = 1$ (0.25), pads touching (0.5); success (2.0)

H.16 LIBERO MANIPULATION BENCHMARK

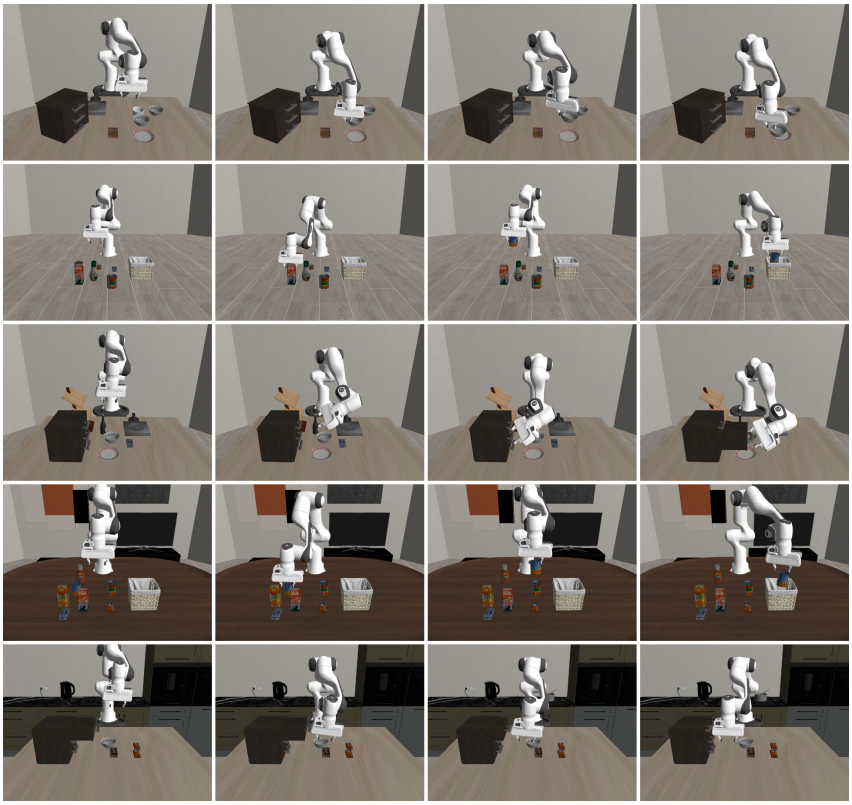


Figure 32: **LIBERO port of Eden.** Four frames from each of five episodes replaying official LIBERO demonstrations through the task's action space, with one episode per row and time progressing from left to right. From top to bottom: LIBERO-SPATIAL (black bowl onto the plate), LIBERO-OBJECT (alphabet soup into the basket), LIBERO-GOAL (opening the middle drawer), LIBERO-10 (alphabet soup and tomato sauce into the basket), and LIBERO-90 (closing the top drawer). Each replay reached its task's success condition. The cameras are chosen for visualization.

The five suites of Table 27 port 130 language-conditioned manipulation tasks from LIBERO (Liu et al., 2024) through one registered configuration, `libero` (`eden.tasks.libero`). A TaskMod selects the task from the catalog. The task's BDDL specification supplies its language instruction, scene entities, regions and logical goals.

Table 27: **LIBERO tasks.** Object and fixture types are listed across all tasks in each suite; workspace families are described below. Braces mark variable phrases in the illustrative instruction patterns; each task retains its original BDDL instruction.

suite	tasks	objects and fixtures	example instruction patterns
LIBERO-SPATIAL	10	black bowls, cookies, ramekins, plates, wooden cabinet, stove	Pick the akita black bowl {location} and place it on the plate
LIBERO-OBJECT	10	alphabet soup, BBQ sauce, butter, chocolate pudding, cream cheese, ketchup, milk, orange juice, salad dressing, tomato sauce, basket	Pick the {object} and place it in the basket
LIBERO-GOAL	10	black bowl, cream cheese, wine bottle, plate, wooden cabinet, stove, wine rack	Put the {object} {relation} the {target}; Open the {drawer}; Turn on the {fixture}
LIBERO-10	10	alphabet soup, butter, chocolate pudding, cream cheese, ketchup, milk, orange juice, tomato sauce, wine bottle, black bowl, basket, black book, frying pan, moka pot, plate, mugs (porcelain, red, yellow/white), desk caddy, stove, microwave, white cabinet, wine rack	Put both {objects} {relation} the {target}; Put the {object} in the {fixture} and close it
LIBERO-90	90	all LIBERO-10 object and fixture types, plus salad dressing, white bowl, wooden tray, yellow book, wooden cabinet and two-layer shelf	Pick up the {object} and put it {relation} the {target}; Open/close the {drawer}; Stack the {object} on the {target}

Scene. The robot is a Franka Emika Panda with seven arm joints and two finger joints. The task selects a tabletop, floor, kitchen, living-room or study workspace together with its fixtures and movable objects. Object and workspace MJCF assets are converted for Genesis when loaded; the tabletop and kitchen workspaces also reconstruct LIBERO's procedural table geometry. Physics runs at 500 Hz with a 2 ms timestep and one substep. Each action spans 25 physics steps, giving a 20 Hz control rate.

Command. No command term is registered. Each task is conditioned on the fixed natural-language instruction in its LIBERO BDDL specification.

Actions. $A = 7$. The first six components specify a relative end-effector pose in the world frame: three translation components and three axis-angle rotation components. They are clipped componentwise to $[-1, 1]$ and scaled by 0.05 m and 0.5 rad, respectively. An operational-space torque controller holds the resulting pose target over the control interval and recomputes feedback at each physics step. The seventh component controls the gripper: positive values close it and negative values open it. The sign of this command increments an internal normalized target by 0.01 per physics step, clipped to $[-1, 1]$ and mapped to position targets for the two fingers.

Observations. The policy receives two 128×128 RGB images, the seven arm joint positions and the two finger joint positions. The fixed external camera has a 45° field of view and the wrist camera 75°. No observation noise is applied.

Terminations and success. Success follows the official LIBERO goal predicates: all predicates in the task's BDDL goal must hold simultaneously. The spatial `On` and `In` predicates reproduce LIBERO's point-containment, support and contact conditions. For object-on-object `On`, the objects must touch, the source origin must be at least as high as the target origin, and their horizontal separation must be below 3 cm. The `Open`, `Close`, `Turnon` and `Turnoff` predicates test the designated joint states. Episodes end on success or timeout. Eden's default time limit is 605 control steps (30.25 s).

Events and curricula. At reset, each environment returns to its assigned saved initial state. The default configuration contains no domain-randomization events or curricula.

H.17 SIMPLER BRIDGEV2 EVALUATION

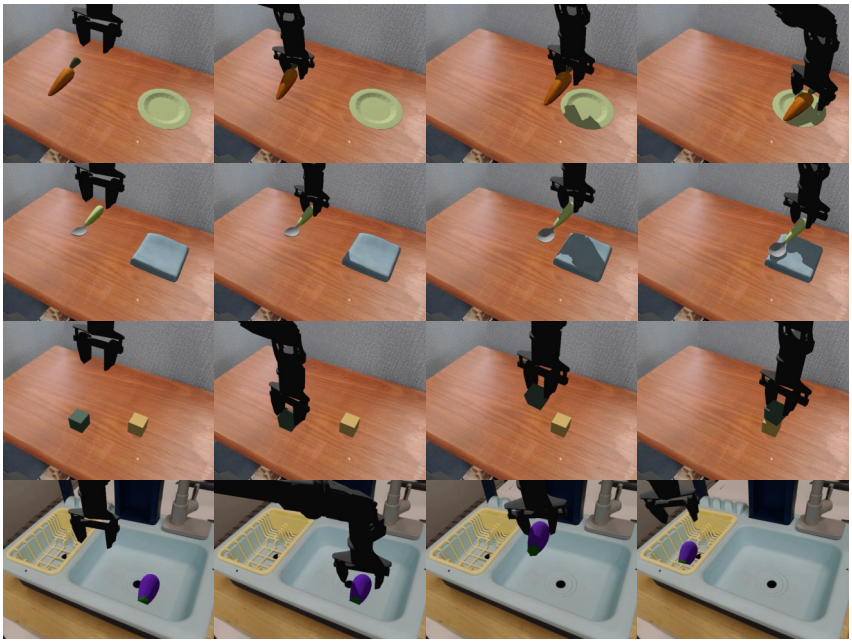


Figure 33: **SIMPLER BridgeV2 port of Eden.** One row per task, four frames each; from top to bottom, carrot on plate, spoon on table cloth, green cube on yellow cube, and eggplant from the sink into the basket.

The four tasks of Table 28 reproduce the BridgeV2 evaluation scenes of SimplerEnv (Li et al., 2024) (`eden.tasks.simpler`). They are built to evaluate a pretrained policy, not to train one, and register no reward terms. All four share one base config and differ only in their objects, camera and reset placement.

Table 28: **SIMPLER tasks.** The source object is moved relative to the target object; both cubes have 3 cm edges. The instruction is the task's only language string.

registered task	source	target	instruction
`put_spoon_on_table_cloth`	spoon	table cloth	"put spoon on table cloth"
`put_carrot_on_plate`	carrot	plate	"put carrot on plate"
`stack_green_cube_on_yellow_cube`	green cube	yellow cube	"put green cube on yellow cube"
`put_eggplant_in_basket`	eggplant	sink	"put eggplant in basket"

Scene. The robot is a fixed-base WidowX-250 with six arm joints and two finger joints. Its PD gains are not the stock values but a system identification fitted to the real BridgeV2 arm, and gravity compensation is on. In three tasks the source and target stand free on a ground plane that serves as the table. In the fourth the source starts inside a sink basin and is moved to the basket beside it. The simulator steps at 200 Hz with five no-slip iterations, and each action is held for 40 steps, so the control rate is 5 Hz.

Command. No command term is registered. Each task is conditioned on the fixed instruction in Table 28.

Actions. $A = 7$. The first six components are an end-effector displacement $\delta = s \odot \tanh(a_{1:6})$ in position and orientation, with $s = 0.5\,\text{m}$ and $0.5\,\text{rad}$ per unit. Damped least squares, $\Delta q = J^\top (JJ^\top + \lambda I)^{-1}\delta$ with $\lambda = 0.1$, turns δ into joint offsets, and the six arm joints track $q + \Delta q$. The seventh component is a binary gripper command: a positive value closes both fingers to 0, and any other value opens them to $\pm 0.037\,\text{m}$.

Observations. No observation terms are registered; the policy reads the RGB image of a single camera. The camera uses the intrinsics of the Logitech C920 with which the BridgeV2 demonstrations were recorded (640×480, focal length 623.6 px). The ground plane is not rendered: a real photograph of the Bridge setup, with the robot inpainted out, is composited in its place. In the sink task the sink is composited as well. The two table-top layouts share one extrinsic calibration and one backdrop, and the sink task has its own.

Terminations and success. No termination term is registered, so the evaluation loop decides when an episode ends. The three table-top tasks score success with one metric: the source's origin lies above the target's, within 0.03 m horizontally and 0.04 m vertically. `put_eggplant_in_basket`' success is detected when the eggplant's root position lies within the basket-region box.

Events and curricula. At reset the table-top tasks place the source and the target independently at $x \sim \mathcal{U}[0.2, 0.35]$ m, $y \sim \mathcal{U}[-0.15, 0.15]$ m on the table, each in its default orientation. The sink task places the eggplant at $x \sim \mathcal{U}[-0.064, -0.050]$ m, $y \sim \mathcal{U}[-0.027, 0.135]$ m in the sink's support-facet frame. There are no curricula.

H.18 PolaRiS DROID evaluation



Figure 34: **PolaRiS DROID port of Eden.** Four frames from each of three scenes, seen through the external camera; from top to bottom, tape into container, block stacking in the kitchen, and food bussing. Everything but the robot and the task objects is the Gaussian-splat reconstruction of the real scene.

The six scenes of Table 29 port the real-to-sim evaluation environments of PolaRiS (Jain et al., 2025) on the DROID (Khazatsky et al., 2023) platform (`eden.tasks.polaris`). Each scene is reconstructed from the real environment as a Gaussian splat, and the simulated robot and objects are composited over it. Each scene has two registered configs. The single-environment config is a non-RL splat config, so it has no reward, termination, command or curriculum manager at all. The `_batched` config keeps the same objects, cameras and actions, renders four environments by default in one batched pass with the splat composited into each, and carries the RL managers, which have no terms yet.

Scene. The robot is the DROID arm: a Franka Panda with a Robotiq 2F-85 gripper welded to its flange. Only the gripper pads collide, because a gripper with full collision geometry penetrates the mesh objects and ejects them. The arm starts from $q_0 = (0, -\pi/5, 0, -4\pi/5, 0, 3\pi/5, 0)$, a home pose calibrated to the splat cameras, and gravity compensation is on. The simulator steps every 4 ms with four substeps, so the gripper's tuned 2 ms contact time constant is not raised by Genesis's clamp at twice the substep, and each action is held for five steps, so the control rate is 50 Hz. Episodes last 30 s (1500 steps).

Command. No command term is registered. Each scene is conditioned on the instruction in Table 29.

Table 29: **PolaRiS scenes.** Every object is a mesh asset placed at a fixed pose taken from the PolaRiS scene. `tape_into_container` registers no instruction.

registered task	objects	instruction
`block_stack_kitchen`	green cube, wooden cube, green tray, pink bowl, purple plate, tomato, corn	"Place and stack the blocks on top of the green tray"
`food_bussing`	yellow bowl, grapes, ice cream, battery, blue cup	"Put all the foods in the bowl"
`pan_clean`	frying pan, sponge, bell pepper, blue mug, cola can, ketchup, mustard, sushi	"Use the yellow sponge to scrub the blue handle frying pan"
`move_latte_cup`	latte-art cup, cutting board, cleaner bottle, cola can	"put the latte art cup on top of the cutting board"
`organize_tools`	scissors, tape, two containers	"put the scissor into the large container"
`tape_into_container`	tape, wrench, container	—

Actions. $A = 8$. The first seven components specify absolute joint-angle targets in radians, $q^{\text{target}} = a_{1:7}$, which the Franka arm tracks using implicit PD position control. The eighth component controls the Robotiq gripper: $a_8 > 0.5$ closes the gripper by setting both driver-joint targets to 0.8 rad; otherwise, both targets are set to 0 rad to open it.

Observations. The policy receives external and wrist RGB images together with an eight-dimensional proprioceptive vector containing seven arm joint angles and normalized gripper closedness. The external camera renders at 1280×720, and the wrist camera at 320×180. The external camera is fixed in the scene, while the wrist camera follows the arm's flange. In both views, ground-plane pixels are replaced by the Gaussian-splat background.

Terminations and success. No environment termination terms are configured. Each task defines a success rubric with prerequisite-dependent reach, lift, and placement criteria. Satisfied criteria remain latched for the episode, and success is declared once all criteria have latched. The evaluation client ends the rollout on success or when its step budget is exhausted.

Events and curricula. None. Every object starts at its fixed PolaRiS pose.

I Hardware Trials

I.1 Unitree G1: whole-body tracking by entity replacement

Setup. Five BeyondMimic whole-body tracking policies (basic walking, short emote, sidekick, balancing, dynamic dancing) were trained in Eden from their reference clips and deployed on a physical Unitree G1 (29-DoF) under a gantry tether through the Unitree G1 deployment backend. The deployment loads the training configuration, replaces the simulated G1 with the hardware-backed entity, and runs the unchanged observation, action, and command managers in a loop paced to the policy's 50 Hz control rate. The achieved loop rate of these trials is not reported: they predate both the rate limiter's overrun reporting and the G1 backend's 500 Hz command re-send (Appendix D), so a tick that overran the 50 Hz schedule was not recorded. All online randomization (observation and action noise, dynamics randomization, motion resampling) is disabled at deployment; the one configuration difference from training is the deployable policy variant's removal of the privileged base-linear-velocity observation. Before the policy takes control, the start-up sequence interpolates from the measured joint configuration to the first reference frame, which removed the transient the earlier video showed. PD gains are the robot preset's defaults, ramped through `deploy_kp_kd_scale`.

Trials. Each of the five policies was run for ten trials (Table 4). Basic walking, short emote, sidekick, and balancing succeed in all ten. Dynamic dancing succeeds in seven; the three failures were attributed to a disconnected wired link and to excessive gantry-tether tension, and not to the policy or the deployment path. No observation, action, or command code was rewritten for any of the five policies.



Figure 35: **Real-world deployment.** *Top*: velocity tracking, LimX Oli. *Bottom*: motion tracking, Unitree G1.

I.2 UR10e + Robotiq 2F-85: simulated demonstrations for a VLA

Setup. The workcell is reconstructed as a 3D Gaussian splat and loaded in Eden alongside the UR10e + 2F-85 URDF; wrist and front cameras are placed at the calibrated extrinsics as Gaussian-splat cameras. One hundred demonstrations are collected in simulation with a GELLO leader arm (Wu et al., 2023) through Eden's device layer, recorded by the recorder manager, exported to LeRobot format, and used to fine-tune GR00T N1.7 (NVIDIA et al., 2025). The fine-tuned model is evaluated in simulation (3×30 episodes) and on the real workcell (30 trials), where the robot is driven by the VLA's own inference server; this experiment does not use an Eden deployment backend.

Controls. The same 100-demonstration collection was repeated in IsaacLab on the same Gaussian-splat scene (Isaac Sim's native splat rendering), giving 76.7% real and $81.1 \pm 5.1\%$ simulated success against Eden's 90.0% and $86.7 \pm 5.8\%$; a collection with a non-reconstructed background yielded a model that never succeeded on the real workcell; and the base checkpoint succeeds in neither domain (Table 30). Visual fidelity of the reconstruction is the dominant enabling factor; the pipeline comparison is what the IsaacLab row isolates.

Robot / task	Policy	Condition	Success (real)	Success (sim)
UR10e + 2F-85, pick-and-place	GR00T N1.7, fine-tuned on 100 Eden sim demos (3DGS scene)	sim-only fine-tuning	27/30 (90.0%)	$86.7 \pm 5.8\%$ (3×30)
	GR00T N1.7, fine-tuned on 100 IsaacLab sim demos (3DGS scene)	sim-only fine-tuning	23/30 (76.7%)	$81.1 \pm 5.1\%$ (3×30)
	GR00T N1.7, fine-tuned on non-3DGS sim demos	sim-only fine-tuning	0/30	—
	GR00T N1.7, base checkpoint	no fine-tuning	0/30	0%

Table 30: **UR10e pick-and-place.** Every fine-tuned policy was trained on simulated demonstrations only. The IsaacLab row uses the same Gaussian-splat scene reconstruction, so the comparison with the Eden row isolates the data-collection pipeline; the non-3DGS row shows that visual fidelity is the dominant enabling factor.

J Coding-Agent Benchmark

This appendix gives the protocol behind §4.1, the inputs every agent received, and results that the main text summarizes. The harness, task specifications, checkers, and documentation packages are released with the framework.

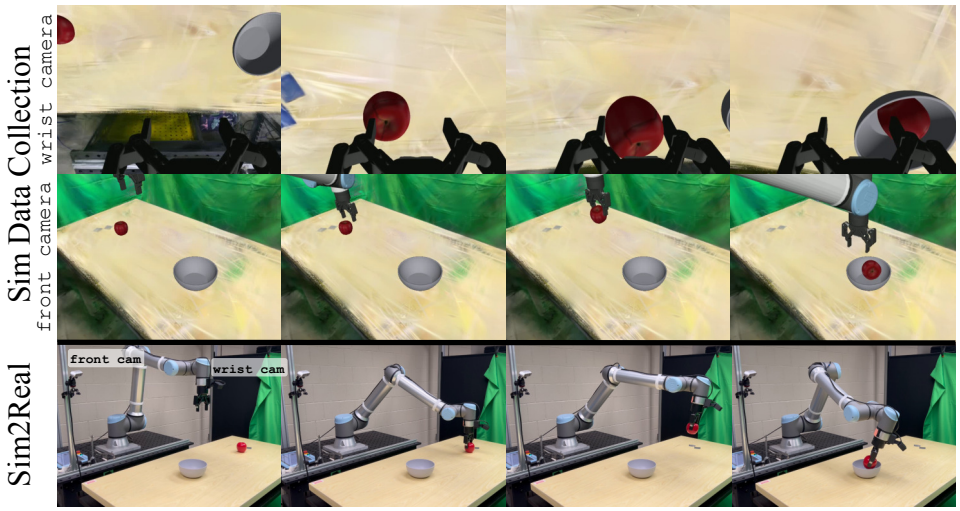


Figure 36: (top) Simulated teleoperation by GELLO (bottom) real execution of GR00T N1.7 fine-tuned on 100 demonstrations in Eden GS scene.

J.1 Protocol

Fixed before the first rollout. The task specifications, success configurations, checkers, per-framework documentation packages, model list, number of rollouts, and pass threshold are hashed into a manifest before the first scored rollout. Each checker is calibrated in both directions: a known-good solution must pass and a do-nothing rollout must fail. The backflip checker requires at least 30% of the net backward rotation to be accumulated in free flight, detected ballistically, and an apex of at most 1.20 m for a robot that stands at 0.30 m, so that a hop followed by a roll on the floor does not pass. This version of the backflip checker replaced an earlier one that scored whole-episode rotation and airborne time separately; the Claude Opus 5 and Haiku 4.5 backflip rollouts were scored by the earlier version and re-scored by this one, which changes no outcome.

Agent harness. Every rollout is a fresh agent session with no persistent memory or project context. The working directory exposes only the specification, the brief (Listing 1), the documentation package, the robot notes, and the checker specification, and the agent has no web access. Model, sampling settings, system prompt, tool set, and timeouts are identical across frameworks and recorded per rollout, and rollouts are ordered so that no framework's rollouts run back-to-back.

Budget and independent scoring. Each rollout has a hard 3 h wall-clock budget, and a rollout over budget is a failure. The agent must leave `train.py`, which accepts `--seed`, and `export_rollout.py`. The operator runs the export from the saved checkpoint and scores the result, so a trajectory written by the agent cannot pass. Single-episode tasks are evaluated deterministically (mean action, randomization off, nominal initial state), and the others as a success rate over re-sampled episodes; the Go1 spin and crawl start from perturbed states so that an open-loop replay fails. An authenticity gate, a trajectory validator plus a simulator-side step counter, rejects fabricated, hand-integrated, and replayed trajectories.

Tasks a framework cannot represent. A task whose physics a framework cannot represent is not run on that framework and is excluded from any cross-framework comparison. Deformable cutting requires a separable continuum material; it runs on Eden (Genesis MPM) and, as a reference cell, on IsaacLab through its experimental Newton MPM backend, and not on mjlab.

Measures. The primary measure is pass@k within the budget. The secondary measure is wall-clock to a working policy, split into agent-side time (reading, writing, and running code) and compute-side

time (training the agent launches) where the harness can separate them. Tokens are recorded for every rollout.

Task	Robot	Episodes	Success requires
`go1_spin_ccw`	Go1	5 perturbed starts	A commanded yaw rate of $+1.5$ rad/s in the commanded direction.
`go1_crawl_backward`	Go1	5 perturbed starts	Backward velocity tracking with a lowered posture.
`door_open`	arm	10 randomized	Opening a hinged door whose body and handle the agent authors.
`bin_pick`	arm + gripper	10 randomized	A grasp and lift; bin walls rule out sweeping.
`dough_cut`	arm + blade	1 episode	Separating a continuum (MPM) slab.
`go1_backflip`	Go1	1 episode	A full backward rotation, at least 30% of it in flight, and a settled landing.
`cube_push`	arm	10 randomized	Pushing a cube, without grasping it, into a goal region where it settles.
`g1_handstand`	G1	1 episode	An extended handstand on both hands, held for 2 s.

Table 31: The tasks. No arm ships with all three frameworks, so each manipulation task runs on an arm the framework ships and is scored in an arm-agnostic, start-relative way. Except for the backflip, the specification permits any control method (RL, trajectory optimization, planning with IK, hand-designed feedback) provided the robot is controlled through the simulator.

Documentation parity. Each framework's package is two files: a prose guide and a components reference, curated to a common outline and containing no worked task example (Table 32). Package sizes span a factor of 1.13. All agents also have the installed framework source.

Framework	Guide (KB)	Components (KB)	Total (KB)
Isaac Lab	47.1	15.7	62.8
mjlab	42.2	13.2	55.4
Eden	43.0	16.4	59.3

Table 32: Documentation packages supplied to the agent in each framework.

J.2 Additional results

Claude Opus 5.5. Table 33 gives every Claude Opus 5.5 cell, one rollout per cell, with wall-clock split into agent-side and compute-side time. It covers the five tasks of Table 2, cube pushing, the G1 handstand, and the backflip. Every entered cell passes. On deformable cutting, Eden reaches a working policy in 9.1 min; as reference points, IsaacLab 2.1.0, which has no MPM material, takes 40.1 min and standalone Newton MPM 49.5 min. Where Eden's total time is higher, as on the backflip (113.0 against 63.5 min for IsaacLab), the difference is compute-side training time (97.9 against 41.3 min), which depends on the schedule the agent chose; agent-side time is lower with Eden (15.1 against 22.2 min).

J.3 Exploratory configuration suite

The tasks of §4.1 each take hours and end in a trained policy. The suite instead measures many small edits to one shipped task, the flat-terrain G1 velocity-tracking task that all three frameworks ship. Its 36 tasks cover every manager: 27 core tasks, and 9 extension tasks of which three are drawn from each framework's own feature set. Six are debug tasks whose starter file carries an injected fault. A framework-neutral probe verifies each task by building it headless and asserting on manager state; no policy is trained. One agent session per framework receives all 36 tasks in shuffled order under a 60 min cap and ends with a verification sweep over every task. The sessions were run before the suite's protocol was frozen and are reported as exploratory.

Task	Framework	pass@1	Total (min)	Agent-side (min)	Compute-side (min)	Tokens (M)	Turns
Go1 spin	IsaacLab	1/1	41.3	21.5	19.8	3.61	104
	mjlab	1/1	29.8	5.9	23.9	2.70	67
	Eden	1/1	21.7	4.8	16.9	3.33	69
Go1 backward crawl	IsaacLab	1/1	23.2	9.7	13.5	2.21	76
	mjlab	1/1	14.5	6.9	7.6	1.96	50
	Eden	1/1	31.9	7.1	24.8	3.69	69
Door opening†	IsaacLab	1/1	79.1	77.7°	1.4°	3.56	95
	mjlab	1/1	15.4	12.1	3.2	3.18	81
	Eden	1/1	24.1	12.5	11.6	6.26	100
Bin picking†	IsaacLab	1/1	31.1	29.5°	1.6°	3.08	90
	mjlab	1/1	16.0	11.0	5.0	2.90	81
	Eden	1/1	13.1	9.4	3.7	3.86	72
Deformable cutting†	IsaacLab 2.1.0 (no MPM)	1/1	40.1	40.1°	0.0°	6.11	145
	mjlab	no separable material model; not entered					
	Eden	1/1	9.1	5.0	4.1	2.22	55
	Newton (standalone MPM)	1/1	49.5	17.3	32.2	7.12	147
Cube pushing	IsaacLab	1/1	34.2	15.1	19.2	2.50	76
	mjlab	1/1	26.1	6.9	19.2	2.08	52
	Eden	1/1	27.4	7.0	20.4	4.00	73
G1 handstand	IsaacLab	1/1	111.9	57.0	54.9	2.38	156
	mjlab	1/1	92.4	23.6	68.8	9.59	151
	Eden	1/1	90.7	15.5	75.2	7.58	115
Go1 backflip	IsaacLab	1/1	63.5	22.2	41.3	3.40	91
	mjlab	1/1	97.3	16.2	81.1	6.86	125
	Eden	1/1	113.0	15.1	97.9	7.66	109

Table 33: **Claude Opus 5.5 on eight tasks, one rollout per cell**, with wall-clock split into agent-side time (the agent reading, writing, and running code) and compute-side time (training runs launched by the agent). Every entered cell passes. Deformable cutting has no cross-framework comparison: the evaluated IsaacLab release has no MPM material, and Newton is a standalone reference entrant, not IsaacLab. †The agent authors the scene. °The harness could not separate agent-side from compute-side time in this cell (training ran in the foreground of an agent command), so the agent-side value includes training. With $n = 1$ per cell the table is descriptive, not a ranking.

Framework	Passed	Solve time, agent-side (s)	Probe runs to pass
IsaacLab	36/36	661	40
mjlab	36/36	630	40
Eden	36/36	608 (464–695)	38

Table 34: **Exploratory 36-task configuration suite**, one Claude Opus 5 session per framework under a 60 min cap, run before the suite's protocol was frozen. Passed: tasks the framework-neutral probe passes. Solve time: wall-clock to the last task's first pass, excluding command run time and the final verification sweep. Probe runs to pass: probe invocations up to each task's first pass, summed over tasks. In parentheses, the range over all 9 Eden sessions run under the same session rules.

Time. Every framework passes all 36 tasks, with similar agent-side solve time (Table 34): the nine Eden sessions span 464–695 s, a range that contains the IsaacLab and mjlab sessions.

Solution size. Table 35 counts the code each session left. Eden's solutions are 31% shorter than IsaacLab's and 33% shorter than mjlab's, and Eden has the smallest total in an earlier session per framework and in the reference solutions as well. Most of the difference is registration. Eden makes a configuration class a named task with a one-line decorator. IsaacLab's `gym.register` and mjlab's `register_mjlab_task` take about eight and seven lines per task, because they also name the environment class or runner and the RL configuration. Task bodies are closer: 490 lines in Eden against 525 in IsaacLab and 629 in mjlab.

Misconfiguration. Two of the six debug tasks assign a parameter through a misspelled field (`weigth` on a reward term; `resampling_time_rang` on the velocity command). Eden refuses both starters at import with a `ValueError` that names the misspelled field, the term, and the fields the term accepts. IsaacLab and mjlab accept both, and the probe detects each fault only from its consequence: the reward weight or resampling time is unchanged. On the other four debug tasks, all

Solutions	Framework	Imports	Registration	Task body	Total	Shortest on
Agent, latest	IsaacLab	262	279	525	1066	2
	mjlab	250	217	629	1096	4
	Eden	202	42	490	734	29
Agent, earlier	IsaacLab	271	279	478	1028	3
	mjlab	228	217	551	996	5
	Eden	202	43	469	714	27
Reference	IsaacLab	249	279	412	940	1
	mjlab	200	217	401	818	2
	Eden	194	40	337	571	32

Table 35: **Lines of code per solution, summed over the 33 suite tasks** that have a solution file in all three frameworks (t01–t03 are edits to the kit's variant files and are not counted). Agent, latest: the sessions of Table 34. Agent, earlier: one earlier session per framework, run under an earlier revision of the session rules that does not affect the code a session leaves. Reference: the reference solutions used to calibrate the probe, written by the authors. A line counts if it holds code; blank lines, comments and docstrings do not. Registration is the statement that makes a configuration available as a named task: a class decorator in Eden, a `gym.register` call in IsaacLab, a `register_mjlab_task` call in mjlab. Shortest on: tasks on which this framework's solution has the fewest lines, ties counted for each.

three frameworks fail the starter on the same probe check. This is a property of the starters, not of the agent sessions.

J.4 Agent brief and task specification

Listing 1: Agent brief (prompt) for the Go1 backflip task. The operator fills `{{FRAMEWORK}}` per cell; `SPEC.md` and the `docs/` package complete the in-context input.

```
# Agent brief -- {{FRAMEWORK}} / Go1 backflip

This is your fixed working brief for one usability-benchmark rollout.
The
operator filled the placeholders below. Work only inside this
workspace.

## Your task
Implement and train the task described in `SPEC.md` (in this directory
) using
the {{FRAMEWORK}} framework. You have a 3-hour budget.

## Environment (do not reinstall)
- The {{FRAMEWORK}} framework is already installed. Run everything
with plain
  `python` / `python3` (e.g. `python train.py`) -- these are pre-wired
 to the
  correct environment for you. Do not call other interpreters or `uv`.
- A Go1 MJCF is symlinked under `./assets/` (your framework also ships
 its own
  Go1 asset/config -- either is fine; match the pinned actuator
hardware in SPEC).
- Headless only -- no display.

## Allowed information (parity condition)
- `SPEC.md` (the task) and this brief.
- The framework documentation package under `./docs/`.
- The installed framework package itself (you may read/introspect its
source).
- No web access. No other frameworks' code.
```

```
## Required deliverables
Leave two things in this directory:

1. `train.py` -- trains the policy from scratch. It MUST accept `--
seed <int>`
   and save its best checkpoint to a path that depends on the seed (
the operator
   re-runs it across several seeds to check reproducibility).
2. `export_rollout.py` -- loads a saved checkpoint and writes a
neutral rollout:
   `export_rollout.py [--ckpt <path>] --out <rollout.npz>`. `--ckpt`
must be
   OPTIONAL: if omitted, default to your latest/best checkpoint (so
the operator
   can score with just `export_rollout.py --out rollout.npz`). It must
 run ONE
   deterministic evaluation episode for a single env and record the
floating-base
   trajectory with the provided recorder:

     from rollout_contract import RolloutRecorder   # symlinked into
this workspace
     rec = RolloutRecorder(dt=<your control dt>, quat_order="wxyz")
     # ONCE, record the sim/controller setup (for the physical-
plausibility score):
     rec.set_setup(joint_names=[...12...], kp=[...], kd=[...],
effort_limit=[...],
                   sim_dt=<physics dt>, decimation=<n>, substeps=<n>,
                   action_scale=[...])
     # per control step, also pass the per-joint state (12-vectors,
same order):
     rec.record(base_pos=..., base_quat=..., base_linvel=...,
base_angvel=...,
                fell=..., dof_pos=..., dof_vel=..., dof_target=...)  #
 dof_target = PD setpoint
     rec.save(out_path)

   Deterministic eval = mean action (no exploration noise), domain
randomization
   OFF, nominal initial state, one env, full episode. quat_order is "
wxyz" for
   IsaacLab and mjlab/MuJoCo -- confirm your framework's convention; a
 wrong order
   silently breaks scoring. The set_setup + dof_* fields are used by
   plausibility_checks.py (real-robot torque/gain/timestep limits);
the base
   fields by success_checks.py.

The operator runs `export_rollout.py` from your saved checkpoint and
scores it --
you do not produce the final `rollout.npz` yourself. (You may still
call the
checker while iterating; see below.)

## How you are scored (do not optimize to this directly)
Stage-2 success is judged by the operator with:

  python success_checks.py rollout.npz --task backflip --config
success_config.json

Success = a full backward rotation, a real jump, and an upright, in-
range,
settled landing. Your training reward is yours to design; the checker
is
reward-agnostic, so "gaming" it means actually doing the backflip.
```

```
## Logging
You do not need to run the harness -- the operator wraps your commands.
 Just do
the work in this directory and leave `train.py`, `export_rollout.py`,
and source.
```

Listing 2: Task specification (`SPEC.md`) for the Go1 backflip task: the pre-registered, framework-neutral description given to every agent. It names observation, action, and reward *concepts* and the pinned actuator hardware, not any framework API.

```
# Task: Unitree Go1 backflip (detailed spec)

> Pre-registered, framework-neutral task description -- the mental
model a
> competent RL engineer would carry into implementing this. It names
the
> observation / action / reward concepts, not any framework API. Do
not edit
> per framework; the only per-framework artifact is the docs package.

## Goal
Train an RL policy that makes the Unitree Go1 quadruped perform a
single
backward somersault (full backward rotation about the body lateral/
pitch axis)
from a stand, and land upright and stable.

## Robot
Unitree Go1, 12 actuated joints -- for each leg {FL, FR, RL, RR}: hip
(abduction), thigh, calf (knee). Floating base.

- Home/stand pose: base height ~= 0.30 m (initialize around 0.32 m),
identity
  orientation; per-leg joint angles hip ~= 0.0, thigh ~= 0.8, calf ~=
-1.5 rad.
- Joint limits (rad): hip [-0.863, 0.863], thigh [-0.686, 4.501],
  calf [-2.818, -0.888].
- Control rate: 50 Hz (control dt = 0.02 s) with the physics stepping
several
  sub-steps per control step.

### Actuator hardware -- FIXED (configure your robot to match)
These model the real Go1 motors (Unitree GO-M8010-6) and must be
identical
across frameworks (do not use a framework's differing built-in
defaults -- set
them explicitly), so the result reflects API usability, not a more
generous
actuator model:
- Peak joint torque (effort limit): hip & thigh = 23.7 N*m, calf =
35.55 N*m
  (the calf is stronger via its ~1.5x linkage reduction).
- Max joint speed (no-load): hip & thigh = 30 rad/s, calf = 20 rad/s
(/1.5 linkage).
- Joint armature / friction per the Go1 MJCF defaults.
Everything else about the controller (kp, kd, action scale, reward
design,
observation design, training hyperparameters) is yours to choose.

## Action (12-D)
Target joint positions as offsets from the home pose, fed to a PD
controller
```

```
(target = home + scale*action). Suggested action scale ~= 0.25-0.5;
clip targets
to joint limits. Use moderate PD gains (position stiffness on the
order of tens,
damping of a few) -- soft enough to allow the explosive motion, stiff
enough to
track. (Torque control is acceptable if the framework favours it, but
PD on
position targets is the expected, faster-to-tune choice.)

## Observation
A proprioceptive state vector, typically:
- base linear velocity (3) and angular velocity (3), body frame;
- projected gravity (3) -- the gravity unit vector in the body frame (
encodes
  orientation, and is what the flip rotates);
- joint positions relative to home (12) and joint velocities (12);
- previous action (12);
- a phase / time-remaining scalar for the maneuver (1) -- important so
 the
  policy can sequence crouch -> launch -> tuck -> land.
No privileged/asymmetric critic obs required, though it is allowed.

## Reward (shaping, reference-free)
A backflip trains reliably in <1 h with a staged shaping reward.
Concepts:
- Rotation: reward backward pitch angular velocity / accumulated
backward pitch
  progress while airborne (this is the dominant term).
- Takeoff: reward upward base velocity and air time at launch; reward
the feet
  leaving the ground together.
- Landing: as the maneuver phase ends, reward returning to upright
  (projected-gravity z -> -1), base height back in the stand range,
and low base
  linear/angular velocity (a settled stand held to episode end).
- Regularization / safety penalties: torque, joint-velocity, action-
rate,
  joint-limit, and self-collision penalties to keep it physical.
Exact weights are the implementer's design choice -- that design
effort is part
of what the benchmark measures.

## Termination
- Early terminate on a non-foot body hitting the ground hard while
inverted, base
  leaving a sane bounding region, or NaN/divergence.
- Time-out at episode end (a short episode of ~= 2-3 s is enough; the
whole
  crouch -> launch -> flip -> land sequence completes in roughly two
seconds).

## Training
- Use the framework's native RL stack / PPO defaults; parallel GPU
envs.
- Headless. Fits the 3-hour budget; the reference converges in well
under 1 h.

## Deliverable (required for scoring)
After training, run one deterministic evaluation episode of the best
policy for a
single env and export the floating-base trajectory via
rollout_contract.RolloutRecorder to rollout.npz. Success is then
judged by
```

```
success_checks.py --task backflip (see success_config.json), NOT by
your training
reward.

Success = full backward rotation (~360 deg sagittal) AND an upright,
in-range,
settled, non-fallen landing held through the final second.
```