GPUSimBench: Towards Scalable and Reliable GPU-Accelerated Simulators in Embodied AI

Huzhenyu Zhang[1,2,*], Shenghai Yuan[3,*], *Member, IEEE*, Wenrui Yan[1], Li Ma[1], Hengjie Li[1], Jingcheng Pang[4,†], and Dmitry Yudin[2,5,*]

***Abstract*— Data-driven embodied AI is rapidly transitioning into a paradigm that scales training through massively parallel simulation, where GPU-accelerated simulators serve as the foundational data infrastructure. However, as computational throughput scales, the underlying trade-offs between parallel efficiency, physical fidelity, and execution determinism remain largely unexamined, hindering the development of reliable robot learning. In this paper, we expose the hidden limits of mainstream GPU-based robotic simulators (e.g., Isaac Lab, Genesis) by introducing GPUSimBench, which focuses on scalability, physical consistency, and computational determinism. First, GPUSimBench establishes a physical grounding evaluation with a controlled inclined-plane task, quantifying the distributional alignment between simulated dynamics and their real-world counterparts. Second, we benchmark parallel scalability by measuring throughput and memory footprints across scaling environment counts. Crucially, beyond standard performance metrics, we unveil and quantify the inherent non-determinism introduced by GPU-batched execution, characterized by significant run-to-run and inter-environment variability even under identical initial conditions. Finally, we identify four empirical regimes of stochasticity within current simulator stacks, highlighting that unbounded scaling can compromise reproducibility without explicit constraints. The developed code is publicly available at github.com/zhanghuzhenyu/gpu_sim_bench.**

I. Introduction

Enabling robots to complete tasks is a hallmark of machine intelligence. Recent advances in embodied AI take a solid step towards this direction [1]–[5], which is rapidly transitioning into a paradigm that scales training from extensive robotic data [6]–[11]. In this area, massively parallel simulation has emerged as the foundational data infrastructure. Advancements in GPU-accelerated robotic simulators, such as Isaac Lab [12], and Genesis [13], enable the simulation of even millions of environments simultaneously. By leveraging the immense parallel computing of modern GPUs, these platforms provide promising computational throughput, fundamentally shifting how the community scales robot learning and policy training [14]–[18].

However, treating these massively parallel simulators merely as infinite data generators overlooks critical underlying complexities. As computational throughput scales to new heights, the complicated trade-offs between parallel efficiency, physical fidelity, and execution determinism remain largely unexamined. In traditional sequential simulators [19], computational determinism and reliable physical contact modeling are often assumed as baselines. In contrast, the highly concurrent nature of GPU-batched execution introduces subtle numerical variations, parallel synchronization artifacts, and floating-point non-determinism. *Without a rigorous understanding of these phenomena, the development of reliable and reproducible robot learning policies is largely hindered.* Despite the critical role of simulation infrastructure, existing benchmarks primarily focus on task-level performance or parallel capability [20]–[24], such as success rates or convergence speed, rather than the fundamental properties of the simulator engine itself. *There is a notable lack of standardized evaluation for how physical consistency degrades as parallelization increases, or how the GPU-parallelization affects the simulation performance.*

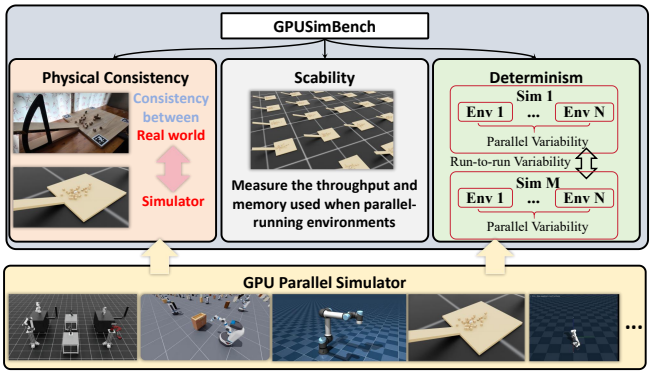

Fig. 1: An overview of key features of GPUSimBench: *physical consistency* between simulator and the real world, *scalability* for sampling efficiency, and *determinism* quantified by parallel and run-to-run variability.

To address these challenges, we introduce GPUSimBench, a comprehensive benchmark designed to expose the hidden limits of mainstream GPU-based robotic simulators. Unlike traditional task benchmarks, GPUSimBench focuses on three key features: scalability, physical consistency, and computational determinism. First, we establish a physical consistency evaluation using a controlled inclined-plane task on both the simulator and the real world, which allows us to quantify the distributional alignment between simulated dynamics and real-world physics. Second, we benchmark parallel scalability by rigorously measuring throughput and memory footprints across varying environment counts. Crucially, beyond standard performance metrics, we unveil and quantify the inherent non-determinism introduced by GPU-batched execution. We make a comprehensive evaluation of

†Corresponding author. [1]Shanghai AI Laboratory, China. [2]MIRAI, Russia. [3]Nanyang Technological University, Singapore. [4]Nanjing University, China. [5]AXXX, Moscow, Russia. Email: *zhanghuzhenyu@pjlab.org.cn / *shyuan@ntu.edu.sg / *yudin.da@miriai.org / †pangjc@lamda.nju.edu.cn

This work was supported by Shanghai AI Laboratory and the National Research Foundation, Singapore, under its Medium-Sized Centre funding scheme through the Centre for Advanced Robotics Technology Innovation.

mainstream GPU-accelerated parallel simulators, including IsaacLab [25], ManiSkill [15], Genesis [13], Madrona [26], MuJoCo Warp [27], MJX [28], and MuJoCo Playground [29]. Simulators that only support CPU-parallel execution or rely on GPUs merely for partial acceleration within the simulation pipeline, such as Gazebo [30] and PyBullet [31], are not considered in this benchmark. Our analysis characterizes significant run-to-run and inter-environment variability, even under identical initial conditions. Specifically, we identify four empirical regimes of stochasticity within current simulator stacks, highlighting how noise accumulates as parallelization increases. Based on these findings, we provide practical guidelines for simulator selection.

Our main contributions are summarized as follows.

- We expose critical but underexamined limitations of mainstream GPU-based simulators, revealing fundamental trade-offs among throughput, physical fidelity, and execution non-determinism.
- We introduce GPUSIMBENCH, a unified benchmarking suite for systematically evaluating scalability, sim-to-real physical consistency, and stochasticity in massively parallel robotic simulation.
- Through cross-simulator and sim-to-real experiments, we quantify GPU-batched non-determinism, identify four empirical stochasticity regimes, and derive guidance for simulator choice and reliable robot learning.

II. Overview of Simulation Platforms

Current GPU-accelerated parallel simulators can be categorized by their underlying programming abstractions and execution models. This classification reflects how simulation state is managed and how parallel work is scheduled onto GPU hardware. Table I summarizes key design features of the simulators considered in this paper.

A. Array-Based and XLA-Accelerated Simulators

This category leverages tensor-based programming models, such as JAX [32], where the simulation state is represented as contiguous arrays. Representative systems include **Brax** [33], **MJX**, and **MuJoCo Playground** [29]. Using JAX-XLA, these simulators compile simulation logic into optimized kernels. While this enables seamless integration with deep learning pipelines and automatic differentiation, the static-shape requirements of XLA can necessitate padding for irregular data structures, such as contact buffers, to maintain a fixed memory layout during execution.

B. High-Level Kernel Framework-Based Simulators

These simulators are built on specialized high-performance computing frameworks that allow users to write simulation logic in a Pythonic syntax while compiling to specialized GPU programs. **Genesis** [13] utilizes Taichi [34], enabling flexible deployment of physics solvers across backends. Similarly, **MuJoCoWarp** and **Taccel** [35] are constructed on NVIDIA Warp [27], a differentiable simulation framework that supports kernel compilation and CUDA Graph capture for low-overhead execution. These platforms balance Python scripting ease with native-level performance, though they require environment authors to explicitly manage data layouts for efficient GPU batching.

C. Task-Centric Simulators

This category includes platforms that wrap established physics engines with specialized management layers. **Isaac Gym** [12] pioneered large-scale GPU-based physics simulation for robot learning, and recent task-centric successors such as **Isaac Lab** [25] and **ManiSkill** [15] provide higher-level task APIs while utilizing the GPU-accelerated PhysX backend for rigid and articulated body dynamics. ManiSkill is built on SAPIEN [36] and supports a wide range of robot embodiments and tasks. These frameworks [37] keep simulation data in device memory to reduce host–device synchronization overhead during parallel execution.

D. ECS-Based Specialized Simulators

A novel approach is represented by **Madrona** [26], which employs an Entity Component System (ECS) architecture. Unlike the tensor-based approach, Madrona decouples components from systems and schedules simulation work directly on the GPU using a Megakernel.

III. Evaluation Methodology

To evaluate GPU-accelerated parallel simulators, we design experiments to measure parallel scalability, resource usage, and distribution-level physical fidelity. This section introduces the core metrics used throughout the experiments.

A. Parallel Throughput

To quantify the computational efficiency of the physics step process, we measure the aggregate simulation throughput in terms of Frames Per Second (FPS) across N_{env} parallel environments. In each benchmark, we perform an initial warmup phase that is excluded from timing, followed by a fixed number of physics steps. Let N_{step} denote the number of timed steps per environment and T_{step} denote the measured time required to execute all calls to steps in N_{env} environments. The throughput is defined as

$$\text{FPS} = (N_{\text{env}} \times N_{\text{step}})/T_{\text{step}}. \tag{1}$$

B. GPU Memory Usage

GPU memory usage is a critical indicator of the resource footprint of GPU-accelerated simulators, especially for large-scale parallel runs. Let $M_{\text{used}}(t)$ denote the instantaneous usage of GPU memory reported by the hardware monitor at time t. We record memory at several checkpoints (baseline before environment creation, post-initialization, post-warmup, peak during stepping, and final). To isolate the memory footprint introduced by simulation, we introduce the peak memory increment

$$M_{\text{usage}} = M_{\text{peak}} - M_{\text{base}}, \tag{2}$$

where M_{base} is the baseline usage before creating environments and M_{peak} is the maximum usage observed.

TABLE I: Comparison of GPU-accelerated parallel simulation frameworks.

Feature	Isaac Lab [25]	ManiSkill [15]	Madrona [26]	Genesis [13]	MuJoCoWarp [27]	MJX [28]	Playground [29]
Parallelized Simulation	✓	✓	✓	✓	✓	✓	✓
Parallelized Heterogeneous Simulation	✓	✓	✓	✓	✗	✗	✗
Implementation	Isaac Sim	SAPIEN [36]	Madrona	Genesis	MuJoCo Warp	MJX	Playground
Physics Backend	PhysX	PhysX	Custom (XPBD [38])	Taichi-based Custom	Warp [27]	MuJoCo	MuJoCo
Primary language	Python	Python	C++ (CUDA Ext.)	Python	Python	Python (JAX)	Python (JAX)

C. Distribution Heatmaps and Density Visualization

We evaluate how accurately different simulators reproduce the final spatial distribution of a cube array after collision by a rolling ball on an inclined plane. For each simulator, we run N_{env} parallel environments with identical initial conditions and record 3D positions of the cube at a fixed physical time t_s. Given a simulation time step Δt, the sampling step is

$$N_s = \text{round}\big(t_s/\Delta t\big)\,. \tag{3}$$

Let $\mathbf{p}_{e,i} \in \mathbb{R}^3$ be the position of the cube i in the environment e at t_s. We aggregate positions across all environments and project them onto the horizontal plane.

$$\mathcal{P}_{xy} = \big\{(x_{e,i}, y_{e,i}) \,\big|\, \mathbf{p}_{e,i} = (x_{e,i}, y_{e,i}, z_{e,i}),\ e = 1, .., N_{\text{env}}\big\}. \tag{4}$$

The distribution heat maps are then generated from $\mathcal{P}_{xy}$ as scatter plots or 2D density maps.

For density visualization, we discretize the xy-plane into K fixed-size hexagonal bins. Let c_b be the number of points in bin b; we use a logarithmic mapping

$$D_b = \log_{10}\big(\max(c_b, 1)\big) \tag{5}$$

for color mapping. This logarithmic density improves high-impact regions and avoids saturation in dense areas, improving comparability for large sample sets. For comparison of cross-simulators, we compute a global maximum bin count $c_{\max}$ and use a shared logarithmic color scale over $1 \le c \le c_{\max}$, ensuring consistent color interpretation across all simulators.

D. Distribution Consistency and Real-World Alignment

We quantify distributional discrepancies using an assignment-based Earth Mover's Distance (EMD) on the projected planar positions. Concretely, we ignore the z direction and compute all distribution distances in $d = 2$ on the xy plane.

1) Parallel Variability (Intra-run Consistency): To assess the stability of a simulator under identical initial conditions, we evaluate Parallel Variability across parallel environments in a single run. For the environment index e, let

$$\mathcal{X}_e = \big\{\mathbf{x}_{e,1}, \ldots, \mathbf{x}_{e,n}\big\}, \quad \mathbf{x}_{e,i} \in \mathbb{R}^2, \tag{6}$$

denote the planar point set of cube positions at sampling time t_s, where $n = m^3$ is the number of cubes per environment ($m = 3$ in our setup, hence $n = 27$).

The EMD between two environments e and e' is computed via a one-to-one optimal assignment:

$$W_1(e, e') = \min_{\pi \in S_n} \frac{1}{n} \sum_{i=1}^{n} \big\|\mathbf{x}_{e,i} - \mathbf{x}_{e',\pi(i)}\big\|_2, \tag{7}$$

where S_n is the set of all permutations of $\{1, \ldots, n\}$. We construct a symmetric consistency matrix $\mathbf{C} \in \mathbb{R}^{N_{\text{env}} \times N_{\text{env}}}$ with elements $\mathbf{C}_{e,e'} = W_1(e, e')$, and summarize it through the average pairwise EMD:

$$\bar{W}_1^{\text{parallel}} = \frac{2}{N_{\text{env}}(N_{\text{env}} - 1)} \sum_{1 \le e < e' \le N_{\text{env}}} W_1(e, e'). \tag{8}$$

This scalar $\bar{W}_1^{\text{parallel}}$ quantifies Parallel Variability: smaller values indicate higher intra-run consistency across parallel environments.

2) Real-World Alignment: A key objective is to measure how closely the simulator output matches the real-world experimental results at the *distribution* level. For a simulator run, we aggregate all parallel environments into a single planar point set $\mathcal{X}^{\text{sim}} = \bigcup_{e=1}^{N_{\text{env}}} \mathcal{X}_e$, and $\big|\mathcal{X}^{\text{sim}}\big| = N_{\text{env}}\, n$.

For the real-world apparatus, we perform K independent trials and aggregate all measured cube xy positions into $\mathcal{X}^{\text{real}} = \bigcup_{k=1}^{K} \mathcal{X}_k^{\text{real}}$, with $\big|\mathcal{X}^{\text{real}}\big| = K\, n$. Here we use $N_{\text{env}} = 16$ and $K = 16$, so $\big|\mathcal{X}^{\text{sim}}\big| = \big|\mathcal{X}^{\text{real}}\big| = N_{\text{env}}\, n$.

The physical-unit EMD between the simulated and real distributions is computed via a one-to-one assignment on the aggregated point sets:

$$W_1^{\text{phys}}\big(\mathcal{X}^{\text{sim}}, \mathcal{X}^{\text{real}}\big) = \min_{\pi \in S_{N_{\text{env}}\, n}} \frac{1}{N_{\text{env}}\, n} \sum_{i=1}^{N_{\text{env}}\, n} \Big\|\mathbf{x}_i^{\text{sim}} - \mathbf{x}_{\pi(i)}^{\text{real}}\Big\|_2, \tag{9}$$

where all positions are expressed in cm and $\mathbf{x} \in \mathbb{R}^2$.

We define the sim2real gap metric as:

$$d_{\text{EMD,phys}}^{\text{sim}} = W_1^{\text{phys}}\big(\mathcal{X}^{\text{sim}}, \mathcal{X}^{\text{real}}\big). \tag{10}$$

Smaller $d_{\text{EMD,phys}}^{\text{sim}}$ values indicate better alignment with real-world planar spatial distributions.

3) Run-to-Run Variability (Inter-run Consistency): To quantify inter-run consistency, let $\mathcal{X}^{(r)}$ be the aggregated planar distribution of run $r \in \{1, \ldots, R\}$ (with $R = 10$), where each $\mathcal{X}^{(r)}$ aggregates all N_{env} parallel environments of that run. We compute

$$\bar{W}_1^{\text{run-to-run}} = \frac{2}{R(R-1)} \sum_{1 \le r < r' \le R} W_1\big(\mathcal{X}^{(r)}, \mathcal{X}^{(r')}\big), \tag{11}$$

where $W_1(\cdot, \cdot)$ uses the same assignment-based EMD as above in cm.

IV. Experiments

All experiments are conducted on a workstation with a 13th Gen Intel(R) Core(TM) i5-13400F CPU, 32 GB RAM, and an NVIDIA GeForce RTX 5070 GPU with 12 GB memory. The evaluation involves seven physics simulators: ManiSkill (v3.0.0b22), Genesis (v0.3.11), Madrona, MujocoWarp, MJX (MuJoCo v3.4.0), MujocoPlayground (v0.1.0),

TABLE II: Physical Parameters of the Inclined Collision Experiment

Object	Physical Property	Value
Ground Plane	Material	Wood (rough)
	Size	$0.6 \times 0.6 \times 0.02$ m
	Center Coordinates (x,y,z)	$(0.3, 0.0, 0.01)$ m
	Static Friction Coefficient	0.65
	Kinetic Friction Coefficient	0.45
	Coefficient of Restitution	0.40
Slope	Material	Wood (smooth)
	Slope Angle	$20°$
	Size	$0.6 \times 0.1 \times 0.02$ m
	Center Coordinates (x,y,z)	$(-0.28191, 0.0, 0.12261)$ m
	Static Friction Coefficient	0.35
	Kinetic Friction Coefficient	0.25
	Coefficient of Restitution	0.40
Ball	Material	304 Stainless Steel
	Radius	0.0175 m
	Mass	0.17782 kg
	Initial Coordinates (x,y,z)	$(-0.22552, 0.0, 0.13135)$ m
	Static Friction Coefficient	0.35
	Kinetic Friction Coefficient	0.25
	Coefficient of Restitution	0.55
Cube	Material	Wood (smooth)
	Edge Length	0.02 m
	Mass (per cube)	0.005 kg
	Static Friction Coefficient	0.35
	Kinetic Friction Coefficient	0.25
	Coefficient of Restitution	0.40
Cube Array	Array Dimension (m×m×m)	$3 \times 3 \times 3$
	Center Spacing (per cube)	0.02 m
	Array Center Coordinates (x,y,z)	$(0.08, 0.0, 0.05)$ m

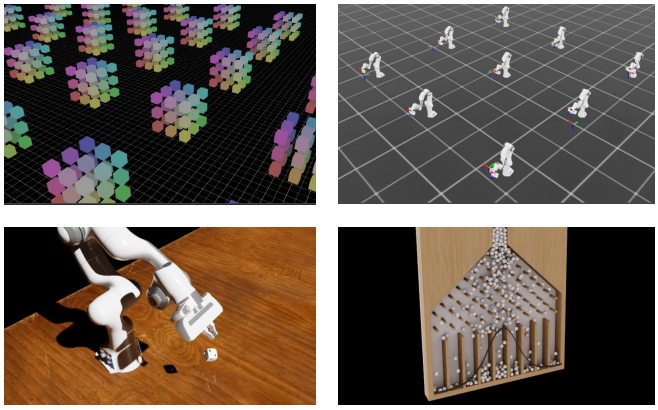

Fig. 2: Benchmark scenes used to stress parallel physics stepping: (up left) a $3 \times 3 \times 3$ cube stack released for free fall and ground impacts, and (up right) a Franka Panda manipulator driven by random actions. Additional real-to-sim distribution experiments, including dice-drop joint pose distribution and Gaussian ball-drop distribution setups, are omitted here due to space constraints and will be released on the project website.

and IsaacLab (v2.2.1), all retrieved from the main branches of their respective repositories.

A. Parallel Capability

We first evaluate the parallel capability of different simulators in two representative scenarios: a free-falling cube array and a Franka manipulator controlled by random actions, as shown in Fig. 2. For each scenario, we disable rendering and measure *physics stepping time only*. Concretely, we use a unified simulation time step $\Delta t = 0.01$ s across all simulators and run 100 warmup steps for JIT compilation or scene initialization. The GPU memory usage of FPS and GPU is measured in 1000 steps.

a) Free-falling cube array.: Each environment contains a uniformly stacked cube array with identical geometry and mass properties. The cubes are arranged in a regular grid of dimension $m \times m \times m$ (with $m = 3$ in our benchmark), a center-to-center spacing equal to the edge length, and a total array height corresponding to m cubes. The array is initially placed above a flat ground plane and released under gravity to undergo free fall and subsequent impacts with the ground. We use simple cube objects to construct the scene, minimizing the influence of file formats on the parallel-capability measurements.

b) Franka manipulator with random actions.: Each environment contains a Franka Panda manipulator mounted on a fixed base. The manipulator is initialized in the same configuration and, at each control step, we sample a random *target joint position* uniformly within the physical joint limits as the control signal. And *self-collision* is enabled for all robot links where supported, so that contact interactions are modeled consistently across simulators. Compared with the free-fall scene, this setup introduces a high-DoF articulated mechanism with joint constraints, which stresses articulated-body dynamics, constraint stabilization, and the ability to batch independent control-and-step loops on the GPU.

B. Inclined Collision

To evaluate physical fidelity and stability under contact-rich dynamics, we design an inclined collision experiment in which the final configuration of a cube array after a ball impact is treated as an observable distribution. Distribution-level outcomes are visualized in Fig. 3 using aggregated density maps and in the right panel of Fig. 4 using one-dimensional marginals

a) Rolling ball down an incline.: The experimental scene consists of a wooden ground plane, an inclined plane, a steel ball, and a cube array. Specifically, the ball rolls down from a specified starting point on the incline, collides with a cube array composed of m^3 uniformly stacked cubes (with $m = 3$), and slowly stops. We construct a physical experiment platform in the real world and measure key physical parameters, including object sizes, masses, friction coefficients, and restitution as shown in Table II. The real-world experiment is conducted indoors, with the ambient temperature maintained at approximately 15°C and the relative humidity maintained at around 15%.

The material and contact parameters reported below are obtained through identification in professional laboratory facilities and are treated as approximate values. In this benchmark, the real-world distribution is used as an empirical reference of the observed outcome, and the reported EMD values should be interpreted as distribution-level agreement with this measured reference under our best-effort parameter matching. For evaluation, we record all cube positions at a fixed time $t_s = 5.0$ s.

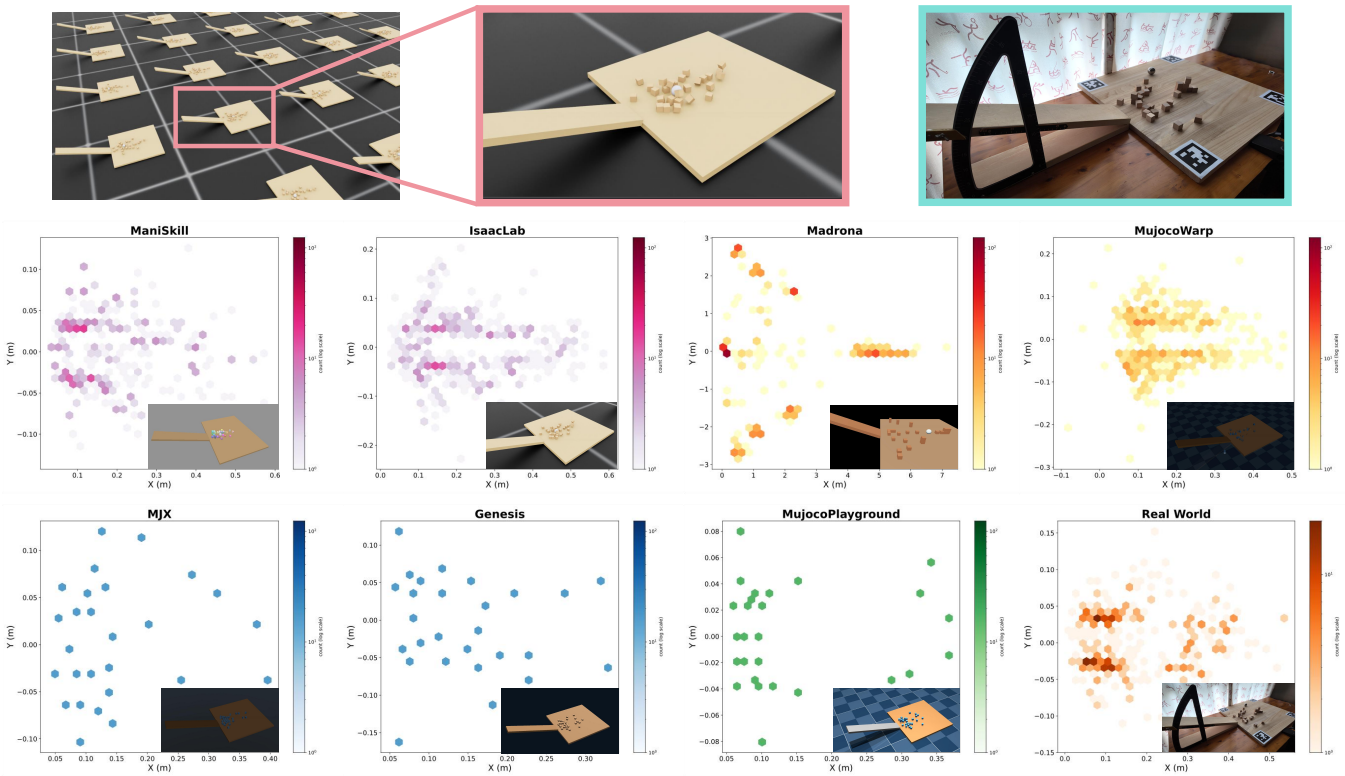

Fig. 3: Distribution-level outcomes of the inclined-collision benchmark. For each simulator and the real-world reference, we aggregate final cube positions across N_{env} parallel environments, project them to the xy plane, and visualize the resulting log-density estimate alongside a representative simulator screenshot. Border colors indicate the determinism regimes Type 1–4 defined in Table IV.

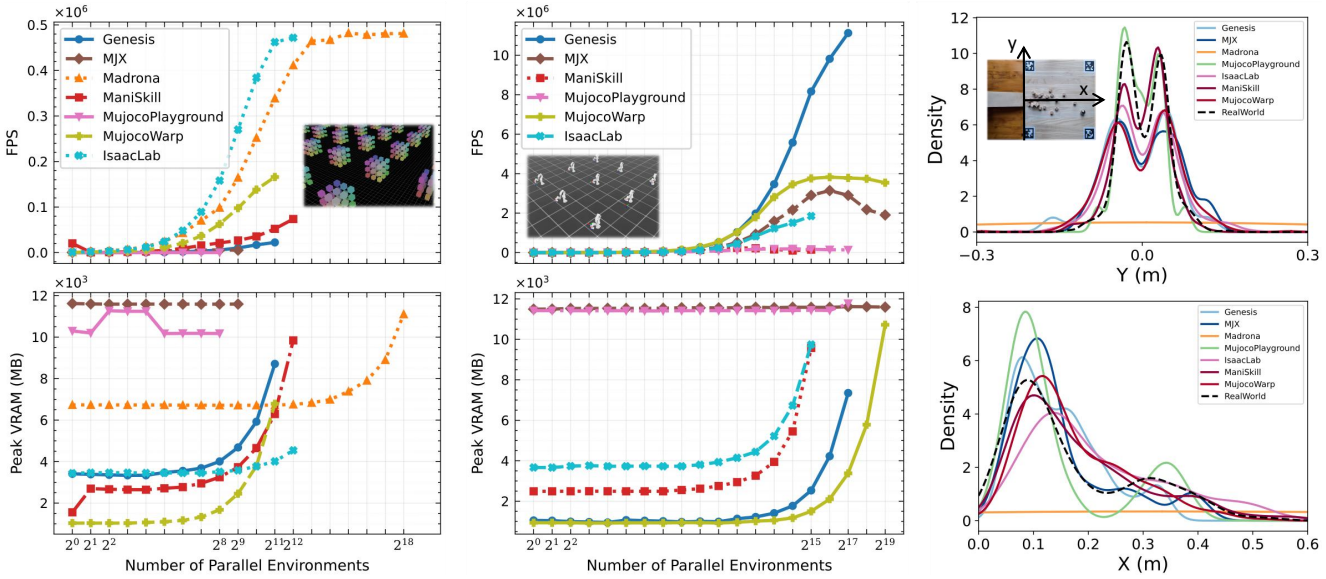

Fig. 4: Composite summary of scalability and distribution results. **Left:** free-fall benchmark throughput and GPU memory increment versus the number of parallel environments N_{env}. **Middle:** Franka random-action benchmark throughput and GPU memory increment versus N_{env}. Each simulator is evaluated up to its maximum supported parallelism $N_{\text{env}}^{\max}$. **Right:** one-dimensional marginal distributions along x and y for the inclined-collision benchmark, aggregated across parallel environments for each simulator.

V. Benchmarking Results

A. Parallel Capability

The parallel capability benchmark evaluates the simulators' throughput and memory usage across varying environment scales. Two scenarios are considered: a free-falling cube array and a Franka manipulator with random actions. For each simulator, we repeat the experiment for ten independent runs.

Throughput & GPU Memory Usage: We summarize the maximum supported parallel environment count $N_{\text{env}}^{\max}$ in each scenario, together with the peak throughput and GPU memory increment observed, in Table III. The left and the middle of Fig. 4 visualizes the full scalability curves.

TABLE III: Maximum supported parallel environment count ($N_{\text{env}}^{\max}$) in the parallel-capability sweep, together with the throughput (FPS) and GPU memory increment (GB) observed at that maximum, for both scenarios. Values are averaged over successful runs.

Simulator	Free-fall			Franka		
	$N_{\text{env}}^{\max}$	FPS ($\times 10^4$)	Mem.	$N_{\text{env}}^{\max}$	FPS ($\times 10^6$)	Mem.
Genesis	2^{11}	2.234	7.8	2^{17}	11.124	6.4
Isaac Lab	2^{12}	47.189	3.3	2^{15}	1.851	8.1
Madrona	2^{18}	48.108	9.5	*Not supported*		
ManiSkill	2^{12}	7.375	8.3	2^{15}	0.149	8.3
MJX	2^{9}	0.534	10.6	2^{19}	1.905	10.6
MuJoCo Warp	2^{11}	16.600	5.9	2^{19}	3.541	9.8
Playground	2^{8}	0.003	8.9	2^{17}	0.135	10.3

B. Physical Consistency

The second part of our evaluation focuses on parallel and physical consistency through an inclined collision experiment. The simulated distribution of cubes is compared against real-world distribution using the planar (xy) assignment-based Earth Mover's Distance (EMD), i.e., we ignore the z direction. We use $N_{\text{env}} = 16$ parallel environments per simulator run and perform $K = 16$ matched real-world trials, so both the simulated and real distributions contain $N_{\text{env}}\, n$ cube xy positions and can be compared with a one-to-one assignment.

Table IV summarizes both physical alignment to the real-world apparatus and consistency metrics in the inclined-collision experiment. In terms of fidelity, ManiSkill and MJX achieve the lowest EMD values of 2.520 ± 0.000 cm and 3.970 ± 0.000 cm, respectively. MuJoCo Playground also shows strong accuracy with an EMD of 3.380 ± 0.110 cm, and Genesis follows closely at 4.780 ± 0.000 cm. In contrast, Isaac Lab (5.400 ± 0.000 cm) and Madrona (211.60 ± 2.22 cm) exhibit larger discrepancies. The extreme value of Madrona is caused by its current implementation of the XPBD [38] solver in our setup: the cubes do not reliably decelerate in the wooden plane due to insufficient frictional effects, leading to large planar drift. MuJoCo Warp shows a moderate deviation (4.230 ± 0.740 cm). Table IV is obtained as mean $\pm$ standard deviation in ten independent runs.

VI. Discussion

A. Parallel and Run-to-Run Variability

We use two consistency metrics defined in Section III: (i) *Parallel Variability*, the mean pairwise EMD $\bar{W}_1^{\text{parallel}}$ between environments within a *single* run, averaged over ten independent runs, and (ii) *Run-to-Run Variability*, the mean pairwise EMD $\bar{W}_1^{\text{run-to-run}}$ between aggregated distributions from all pairs of independent runs under the same nominal configuration. In Table IV, the values are reported as the mean $\pm$ standard deviation for ten runs (for Parallel Variability) or for all pairs of runs (for Run-to-Run Variability). Values displayed as 0.00 ± 0.00 indicate the absence of both Parallel Variability and Run-to-Run Variability.

For discussion purposes, we group the simulators into four empirical regimes based on whether each variability metric is present or absent in Table IV. In all inclined-collision runs, we fix random seeds and disable all task-level randomization, so any observed variability reflects numerical and execution-level effects rather than intentional stochasticity.

Type 1 (Isaac Lab, ManiSkill): Parallel Variability present; Run-to-Run Variability absent. Within a single run, different parallel environments drift to measurably different outcome distributions, leading to non-zero Parallel Variability. However, when we repeat the entire experiment across independent runs under the same nominal configuration, the Run-to-Run Variability is reported as 0.00 ± 0.00 in Table IV. At our reporting precision, the aggregate outcome distribution is effectively unchanged between runs.

Type 2 (Genesis, MJX): Parallel Variability absent; Run-to-Run Variability absent. Both metrics are reported as 0.00 ± 0.00 in Table IV. In our benchmark, this means that the results are indistinguishable between parallel environments and between independent runs at the reported precision, making this regime the most straightforward choice for fair comparisons.

Type 3 (Madrona, MuJoCo Warp): Parallel Variability present; Run-to-Run Variability present. Both intra-run (between parallel environments) and inter-run (between runs) variability are measurable in Table IV. In other words, repeating the same nominal setup can change the aggregate outcome distribution, and environments within a run also diverge from one another.

Type 4 (MuJoCo Playground): Parallel Variability absent; Run-to-Run Variability present. Parallel environments within a run are reported as 0.00 ± 0.00, but repeating the experiment across independent runs yields measurable Run-to-Run Variability.

B. Mechanisms Behind the Stochasticity

In this benchmark, we fix random seeds and disable task-level randomization. Consequently, the observed variability in Table IV isolates the *numerical and execution-level artifacts* inherent to batched GPU simulation. We suggest that these variability metrics arise from a tightly coupled interplay between hardware-level execution models (SIMT dynamic scheduling), algorithmic solver structures, and the inherently non-smooth nature of rigid-body contact dynamics. We interpret these mechanisms as follows:

a) Parallel Variability: Parallel Variability is quantified by $\bar{W}_1^{\text{parallel}}$, with $\bar{W}_1^{\text{parallel}} > 0$ indicating divergence between parallel environments within *a single run* under identical initialization. The non-zero $\bar{W}_1^{\text{parallel}}$ is typically introduced by GPU-parallel computations that rely on atomics or parallel reductions. Because floating-point accumulation depends on operation order, different thread schedules can create small numerical noise across environments; in contact-rich rollouts, this noise accumulates and appears as a distribution shift. For instance, the Parallel Variability in Isaac Lab and ManiSkill arises from PhysX's GPU execution-order differences in floating-point computations, where small round-off perturbations accumulate over contact-rich rollouts.

TABLE IV: Inclined-collision benchmark summary: physical alignment to the real-world apparatus ($d^{\text{sim}}_{\text{EMD,phys}}$), intra-run consistency ($\bar{W}_1^{\text{parallel}}$), and inter-run consistency ($\bar{W}_1^{\text{run-to-run}}$). Checkmarks indicate whether the corresponding variability is present (nonzero after rounding) and are used to define Type 1–4. Values are mean $\pm$ standard deviation over 10 runs (for $d^{\text{sim}}_{\text{EMD,phys}}$ and $\bar{W}_1^{\text{parallel}}$) or over all run pairs (for $\bar{W}_1^{\text{run-to-run}}$).

Type	Simulator	$d^{\text{sim}}_{\text{EMD,phys}}$ (cm)	$\bar{W}_1^{\text{parallel}}$ (cm)	Parallel Var.	$\bar{W}_1^{\text{run-to-run}}$ (cm)	Run-to-Run Var.
1	Isaac Lab	5.400 ± 0.000	4.21 ± 0.00	✓	0.00 ± 0.00	✗
	ManiSkill	2.520 ± 0.000	4.76 ± 0.00	✓	0.00 ± 0.00	✗
2	Genesis	4.780 ± 0.000	0.00 ± 0.00	✗	0.00 ± 0.00	✗
	MJX	3.970 ± 0.000	0.00 ± 0.00	✗	0.00 ± 0.00	✗
3	Madrona	211.60 ± 2.22	32.53 ± 1.86	✓	11.47 ± 1.40	✓
	MuJoCo Warp	4.230 ± 0.740	4.55 ± 1.39	✓	2.15 ± 0.82	✓
4	MuJoCo Playground	3.380 ± 0.110	0.00 ± 0.00	✗	1.39 ± 0.47	✓

TABLE V: Guidelines for simulator choice.

Keywords (task)	Recommended
Overall performance & Balanced trade-off	**Isaac Lab**
Reproducibility & Fair comparisons & Repeated runs	**MJX**, **Genesis**
Sim-to-real & Contact-rich & Repeated-run tuning/reporting	**MJX**
Sim-to-real & Contact-rich & Single-run evaluation	**ManiSkill**, **MuJoCo Playground**
Large parallelism & Articulated robot	**Genesis**
Large parallelism & Simple agent	**Madrona**
Memory-limited & Modest parallelism	**MuJoCo Warp**

b) Run-to-Run Variability: Run-to-Run Variability is quantified by $\bar{W}_1^{\text{run-to-run}}$, with $\bar{W}_1^{\text{run-to-run}} > 0$ indicating differences between *multiple independent runs* under the same nominal configuration. In our setting, a primary source is the algorithmic path taken by iterative numerical solvers: small floating-point perturbations can change contact ordering, activation of constraints, projection decisions, and early-termination conditions, leading to different sequences of intermediate iterates even with identical nominal settings. In long-horizon contact-rich rollouts, these solver-path differences can compound into observable distribution shifts.

C. Guidelines for Simulator Choice

Based on scalability and real-world-matched inclined-collision results, Fig. 5 compares simulators across six axes: throughput, maximum parallelism, GPU memory footprint, physical alignment (EMD), Parallel Variability, and Run-to-run Variability. All axes are min–max normalized (with a $\log_2$ transform on maximum parallelism before normalization). For metrics where lower is better (GPU memory footprint, EMD, Parallel Variability, Run-to-run Variability), we invert the direction so that larger radial values always indicate better performance.

We provide the guidelines below based on the variability regimes and the physical-alignment ranking in Table IV. Use Fig. 5 to compare shortlisted candidates across these criteria, rather than relying on any single metric.

a) Sim-to-real transfer and contact-rich manipulation.: Start by shortlisting simulators with the strongest nominal physical alignment in Table IV. If your workflow involves contact-parameter tuning, design comparisons, or reporting averaged sim-to-real outcomes across repeated runs, additionally prioritize low Run-to-Run Variability in Table IV to reduce confounding from solver path drift; in our benchmark, MJX offers the most favorable overall trade-off between

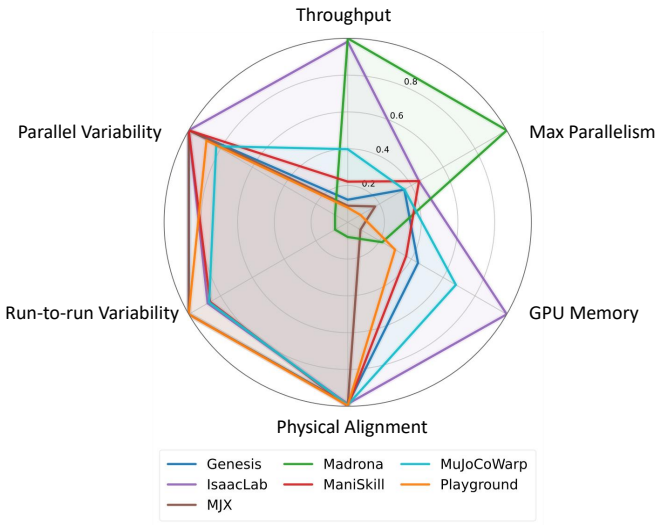

Fig. 5: An overall statistics of the key features for different simulators. Larger values indicate better performance.

alignment and run-to-run stability. If you evaluate within a single run and mainly need plausible contact behavior without extensive tuning, ManiSkill and MuJoCo Playground are also competitive choices. Domain randomization can further improve robustness, but it is best applied around a well-aligned nominal model rather than used to compensate for systematic simulator bias.

b) Scaling-first training with very large parallelism.: When sample throughput is primary constraint, choose simulators that maximize parallelism for your target agent class. For multi-joint articulated robots such as Franka, Genesis achieves the highest throughput in our Franka scalability experiment in Fig. 4, making it a strong default for scaling training with articulated agents. For tasks that do not rely on articulated robots or custom agent pipelines, Madrona is a compelling option for pushing maximum parallelism, as its ECS architecture is designed for batched GPU execution.

c) Hardware-limited setups with modest parallelism.: If GPU memory is the primary bottleneck and only a small number of parallel environments is needed, MuJoCo Warp is a practical choice. In our scalability experiment, it has the lowest GPU memory footprint at low environment counts while maintaining moderate throughput, as shown in Fig. 4.

d) Balanced default choice.: When your task does not impose a dominant constraint and prefer a well-balanced

simulator, Isaac Lab is a strong default because it offers a competitive balance across throughput, scalability, memory footprint, physical alignment, and reproducibility.

VII. Conclusion and Limitation

This paper benchmarks mainstream GPU-accelerated parallel robotic simulators under unified conditions, focusing on the practical trade-offs between throughput, GPU memory footprint, and physical consistency. Using two scalability tasks (free-falling cube arrays and a randomly actuated Franka arm) and a real-world-matched inclined-collision experiment, we quantify both performance scaling and distribution-level physical fidelity. Across simulators, we observe distinct regimes of determinism characterized by Parallel Variability and Run-to-Run Variability, showing that GPU-parallel design choices affect not only speed but also reproducibility and outcome distributions. We release this benchmark and evaluation protocol to support more principled simulator selection and future work on scalable, physically reliable GPU simulation.

This work has several limitations. First, GPUSimBench currently covers only a limited set of tasks and metrics, and the conclusions should be interpreted within this benchmark scope. Specifically, we evaluate two scalability scenes and one contact-rich real-world-matched experiment, all on a single hardware and software stack. As a result, the reported performance and variability may change across different GPUs, drivers, and simulator versions. In addition, the current benchmark does not yet cover deformable-body dynamics, fluid simulation, or the effects of perception and sensor rendering. Extending the benchmark to these settings is an important direction for future work.

References

[1] A. O'Neill, A. Rehman *et al.*, "Open x-embodiment: Robotic learning datasets and RT-X models," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024.

[2] B. Zitkovich, T. Yu *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proc. Conf. Robot Learn. (CoRL)*, 2023.

[3] A. Brohan, N. Brown *et al.*, "RT-1: Robotics transformer for real-world control at scale," in *Proc. Robot. Sci. Syst. (RSS)*, 2023.

[4] J.-C. Pang, N. Tang *et al.*, "Learning view-invariant world models for visual robotic manipulation," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2025.

[5] J.-C. Pang, S.-H. Yang *et al.*, "Object-oriented option framework for robotics manipulation in clutter," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2023.

[6] T. Yu, T. Xiao *et al.*, "Scaling robot learning with semantically imagined experience," in *RSS*, 2023.

[7] F. Lin, Y. Hu *et al.*, "Data scaling laws in imitation learning for robotic manipulation," in *ICLR*, 2025.

[8] Y. Fang, Y. Yang *et al.*, "Rebot: Scaling robot learning with real-to-sim-to-real robotic video synthesis," in *IROS*, 2025.

[9] Q. Li and S. Yuan, "Jacquard V2: Refining datasets using the human-in-the-loop data correction method," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024.

[10] T.-M. Nguyen, S. Yuan *et al.*, "MCD: Diverse large-scale multi-campus dataset for robot perception," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2024.

[11] S. Wang, S. Li *et al.*, "UAVScenes: A multi-modal dataset for UAVs," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2025.

[12] V. Makoviychuk, L. Wawrzyniak *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," in *NeurIPS*, 2021.

[13] G. Authors, "Genesis: A generative and universal physics engine for robotics and beyond," December 2024. [Online]. Available: https://github.com/Genesis-Embodied-AI/Genesis

[14] S. H. Jeon, H. J. Lee *et al.*, "Residual MPC: Blending reinforcement learning with GPU-parallelized model predictive control," *arXiv preprint arXiv:2510.12717*, 2025.

[15] S. Tao, F. Xiang *et al.*, "Maniskill3: GPU parallelized robotics simulation and rendering for generalizable embodied AI," in *Proc. Robot. Sci. Syst. (RSS)*, 2025.

[16] N. Rudin, D. Hoeller *et al.*, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *CoRL*, 2021.

[17] H. Wang, M. Sit *et al.*, "GEAR: A gpu-centric experience replay system for large reinforcement learning models," in *ICML*, 2023.

[18] A. Pitkevich, "Faster training for robotic manipulation in GPU parallelized robotics simulation," *Int. J. Softw. Sci. Comput. Intell.*, 2025.

[19] M. Cao, T.-M. Nguyen *et al.*, "Cooperative aerial robot inspection challenge: A benchmark for heterogeneous multi-uncrewed-aerial-vehicle planning and lessons learned," *IEEE Robot. Autom. Mag.*, 2025.

[20] H. Geng, F. Wang *et al.*, "Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning," in *Proc. Robot. Sci. Syst. (RSS)*, 2025.

[21] M. Sedlacek, P. Yefanov *et al.*, "Realm: A real-to-sim validated benchmark for generalization in robotic manipulation," *arXiv preprint arXiv:2512.19562*, 2025.

[22] Y. Liu, Q. Wang *et al.*, "Performance comparison of typical physics engines using robot models with multiple joints," *IEEE Robot. Autom. Lett.*, 2023.

[23] T. Yoon, J. Lee, and J. Park, "A comparative study on physics engines for robot simulation with mechanical interaction," *Sensors*, 2023.

[24] A. A. Shahid, Y. Narang *et al.*, "Benchmarking population-based reinforcement learning across robotic tasks with gpu-accelerated simulation," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, 2025.

[25] M. Mittal, P. Roth *et al.*, "Isaac lab: A GPU-accelerated simulation framework for multi-modal robot learning," *arXiv preprint arXiv:2511.04831*, 2025.

[26] B. Shacklett, L. G. Rosenzweig *et al.*, "An extensible, data-oriented architecture for high-performance, many-world simulation," *ACM Trans. Graph.*, 2023.

[27] M. Macklin, "Warp: A high-performance python framework for gpu simulation and graphics," https://github.com/nvidia/warp, 2022, nVIDIA GPU Technology Conference (GTC).

[28] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IROS*, 2012.

[29] K. Zakka, B. Tabanpour *et al.*, "Mujoco playground," *arXiv preprint arXiv:2502.08844*, 2025.

[30] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2004.

[31] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[32] J. Bradbury, R. Frostig *et al.*, "JAX: Composable transformations of Python+NumPy programs," 2018. [Online]. Available: https://github.com/jax-ml/jax

[33] C. D. Freeman, E. Frey *et al.*, "Brax – a differentiable physics engine for large scale rigid body simulation," in *NeurIPS*, 2021.

[34] Y. Hu, T.-M. Li *et al.*, "Taichi: a language for high-performance computation on spatially sparse data structures," *ACM Trans. Graph.*, 2019.

[35] Y. Li, W. Du *et al.*, "Taccel: Scaling up vision-based tactile robotics via high-performance GPU simulation," in *NeurIPS*, 2025.

[36] F. Xiang, Y. Qin *et al.*, "SAPIEN: A simulated part-based interactive environment," in *Proc. CVPR*, 2020.

[37] W. Li, G. Chen *et al.*, "Cleanupbench: Embodied sweeping and grasping benchmark," *arXiv preprint arXiv:2508.05543*, 2025.

[38] M. Macklin, M. M"uller, and N. Chentanez, "Xpbd: Position-based simulation of compliant constrained dynamics," in *MIG*, 2016.