ORBISIM: WORLD MODELS AS DIFFERENTIABLE PHYSICS ENGINES FOR EMBODIED INTELLIGENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

We present OrbiSim, a novel robotic simulation paradigm that redefines world models as a fully differentiable physics engine for embodied intelligence. Unlike prior world models that focus on unconstrained imagination in latent or visual domains, OrbiSim establishes a unified, physically-grounded pathway that bridges structured scene assets, neural dynamics, and downstream reinforcement learning. By enabling end-to-end differentiability throughout the entire simulation loop—spanning from explicit state transitions to visual observation generation—OrbiSim supports tasks traditionally intractable for classical simulators, such as differentiable contact modeling, gradient-based policy optimization under sparse rewards, and intuitive physical inference. Empirical results demonstrate that OrbiSim significantly outperforms state-of-the-art world models in both predictive fidelity and control performance. Furthermore, its consistent responsiveness to asset configurations and physical parameters suggests its potential as a differentiable tool for enhancing robot simulation and policy training.

1 INTRODUCTION

The advancement of embodied intelligence is intrinsically tied to the fidelity of simulation software (Todorov et al., 2012; Zhu et al., 2020; Mittal et al., 2025), with classical physics engines such as MuJoCo (Todorov et al., 2012), PhysX (NVIDIA Corporation, 2020), and Bullet (Coumans, 2015) serving as the core execution backends of these platforms. Despite their success, these engines are fundamentally constrained by their non-differentiable or only partially differentiable execution pipelines, rigid modeling assumptions, and the difficulty of integrating perception, dynamics, and control within a unified optimization framework.

In parallel, recent years have seen rapid advances in world models that learn predictive representations of environment dynamics directly from data (Bruce et al., 2024; Gao et al., 2025; Quevedo et al., 2025). These models have demonstrated impressive capabilities in visual perception, video prediction, and counterfactual reasoning for planning and reinforcement learning (RL). Nevertheless, most existing world models act more as visual predictors than as functional simulators, lacking the structural integrity required to replace traditional engines in a closed-loop training pipeline. They typically do not expose explicit physical states, do not accept structured scene assets or physical parameters as inputs, and do not provide a fully differentiable execution pathway that connects environment configuration to downstream control objectives.

To bridge this gap, we introduce *OrbiSim*, a novel paradigm that redefines world models as a fully differentiable physics engine for robotics. Unlike prior models that operate on unconstrained latent spaces, OrbiSim treats scene geometry and physical parameters as first-class inputs within a unified, physically-grounded pathway. As illustrated in Figure 1, the core strengths of OrbiSim are threefold: (1) *General-purpose world representation.* OrbiSim adopts an asset-conditioned representation interface that supports heterogeneous object types through appropriate state and geometry encodings, rather than being limited to a task-specific design; (2) *Decoupled dynamics and vision.* By decoupling the neural architecture into interlinked dynamics and rendering modules, OrbiSim simultaneously predicts precise physical states and high-fidelity visual observations and enables seamless integration with existing simulation platforms; and (3) *End-to-end differentiability.* The differentiable pipeline facilitates Real-to-Sim system identification over scene parameters and gradient-based policy optimization for downstream control.

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.

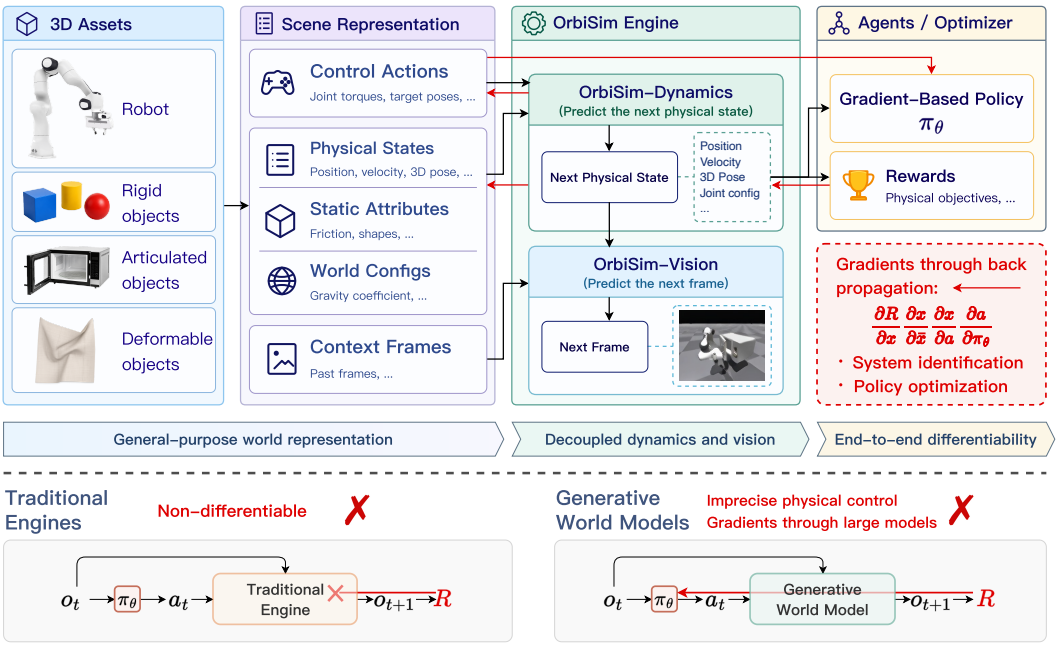


Figure 1: **Overview of OrbiSim.** Unlike prior generative world models, OrbiSim is designed as a *differentiable physics engine* that seamlessly integrates with modern robotic simulation software, enabling asset control and joint state-pixel generation. OrbiSim provides analytical gradients for direct policy optimization, which is a critical capability that distinguishes it from traditional simulators.

We evaluate OrbiSim across a diverse suite of physics-rich benchmarks. The results show that OrbiSim achieves competitive physical consistency compared to classical physics solvers while substantially outperforming state-of-the-art world models in video prediction fidelity and long-horizon rollout stability. Beyond standard benchmarking, we further show that OrbiSim supports real-to-simulation through visual inference of physical states and hidden properties, extends naturally to articulated and deformable objects under a unified asset-conditioned framework, and enables analytical policy gradients for sparse-reward robotic manipulation.

2 Related Work

Traditional Physics Engines for Robotics. Simulators such as MuJoCo Todorov et al. (2012), PyBullet Coumans & Bai (2016–2021), Gazebo Koenig & Howard (2004), robosuite Zhu et al. (2020), and Isaac Gym/Lab Liang et al. (2018); Makoviychuk et al. (2021); Mittal et al. (2025) are the standard for robotic learning due to their high physical fidelity and scalability. However, their reliance on non-differentiable numerical solvers introduces discontinuities that prevent gradient propagation through environment transitions. Differentiable simulators, including Brax Freeman et al. (2021), Warp Macklin (2022), DiffTaichi Hu et al. (2019), and GradSim Murthy et al. (2020), seek to address this limitation by redesigning simulation pipelines for automatic differentiation. While these methods enable gradient-based optimization, they often trade off physical accuracy, relying on simplified or approximate dynamics. They struggle to faithfully capture complex contact interactions, limiting their applicability in high-fidelity robotic settings.

Generative World Models. Recent generative world models Agarwal et al. (2025); Gao et al. (2025); Huang et al. (2025); Rigter et al. (2024) excel at synthesizing visually coherent futures but often lack explicit physical attributes (*e.g.*, mass, friction) necessary for precise control. While state-of-the-art embodied world models such as DreamDojo Gao et al. (2026), Lingbot-World Team et al. (2026), MOTUS Bi et al. (2025), and LAWM Tharwat et al. (2025) leverage world modeling to improve VLA policies, they do not function as full physical simulators. Similarly, navigation-centric models He et al. (2025); Öcal et al. (2024); Lu et al. (2024); Yu et al. (2025) focus on infinite scene generation rather than object-centric interactivity. Fundamentally, these paradigms entangle visual synthesis with temporal transitions, encoding physics implicitly within massive parameter spaces.

This entanglement hampers reliable gradient propagation, making them ill-suited for the differentiable robot simulation.

Latent World Models. Latent world models have transformed model-based RL by enabling compact, controllable representations of environment dynamics Ha & Schmidhuber (2018). Building on this foundation, methods such as PlaNet Hafner et al. (2019), Dreamer Hafner et al. (2025), and subsequent extensions Pan et al. (2024); Li et al. (2025a); Sun et al. (2024); Burchi & Timofte (2025); Cohen et al. (2024) achieve efficient policy optimization by performing imagination and rollout entirely in latent space. These advances have been further adopted in robotics for manipulation and offline evaluation (*e.g.*, IRASim Zhu et al. (2025), PIN-WM Li et al. (2025b), and WorldGym Quevedo et al. (2025)), often outperforming model-free baselines. Despite their empirical success, latent world models remain poorly aligned with simulation-oriented settings. They provide limited support for customizable assets, such as variable object geometries or compositional environments. Moreover, their reliance on abstract latent representations deprioritizes high-fidelity visual rendering, restricting their applicability in scenarios that require photorealistic simulation or rendering-based evaluation.

3 ORBISIM

We aim to learn a differentiable world model that functions as an execution engine for embodied manipulation tasks. Unlike monolithic world models that entangle physics and pixels, OrbiSim adopts a decoupled neural architecture comprising two specialized modules: *OrbiSim-Dynamics* and *OrbiSim-Vision*. As illustrated in Figure 1, this decomposition mirrors the separation of physics and rendering in classical simulation software, and it provides flexible, state-visual dual interfaces for downstream robot learning, supporting both MDP and POMDP formulations.

We first introduce a unified world representation for the robot and scene objects, which accommodates multiple object types and their corresponding geometry and physics descriptors (Sec. 3.1). At each step, OrbiSim-Dynamics autoregressively predicts physical states from historical trajectories, actions, and scene descriptors (Sec. 3.2). OrbiSim-Vision renders the next observation by grounding latent diffusion in the predicted states and scene context (Sec. 3.3). This modularity decouples physical evolution from visual complexity, while maintaining a fully differentiable pipeline that enables real-to-sim system identification (Sec. 3.4) and downstream gradient-based policy learning (Sec. 3.5).

3.1 UNIFIED ASSET AND WORLD REPRESENTATION

We model embodied manipulation as a physically grounded simulation process in which a robot interacts with N objects over discrete time steps. Each task instance is defined by a scene configuration specifying object geometry, physical properties, and global environment parameters. As a drop-in replacement for a classical physics engine, OrbiSim represents the system through three components:

- *Physical state (x_t):* The time-dependent physical state x_t is defined as the union of the robot's physical state x_t^p and the states of N interacting objects $\{x_t^{(i)}\}_{i=1}^N$. Each state vector contains the quantities required for dynamics prediction, such as pose, velocities, joint configurations, or other time-varying state variables depending on the asset type.
- *Visual observation (o_t):* Given x_t, the world-model engine produces a high-dimensional visual observation in the form of an RGB image $o_t \in \mathbb{R}^{H\times W\times 3}$.
- *Static scene descriptors ($\bar{x}$):* We use $\bar{x}$ to represent time-invariant properties that characterize the neural simulator's physical bounds. We categorize these into:
  - Asset attributes ($\bar{x}^{(i)}$), which are intrinsic properties of each object i, such as geometry, mass, inertia, surface friction, or other asset-specific descriptors.
  - World configurations ($\bar{x}^w$), which capture global environment parameters shared across all objects, such as gravity or contact properties.

A key advantage of this representation is that OrbiSim is designed as a general-purpose, asset-conditioned framework rather than a task-specific predictor. For simple objects, physical states and static descriptors can be represented directly using low-dimensional attributes. For objects with more complex geometry, such as *articulated* or *deformable* assets, their shape descriptions (*e.g.*, meshes or

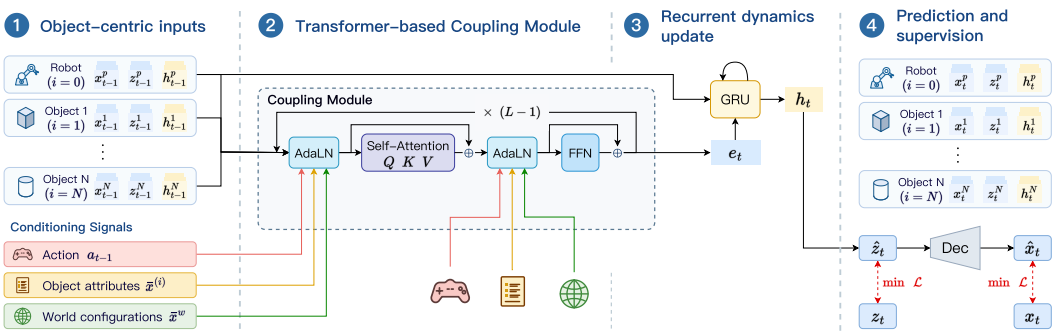


Figure 2: **OrbiSim-Dynamics**. An object-centric recurrent dynamics core that predicts next-step physical states. The model utilizes a Transformer-based coupling module to capture multi-entity interactions, with dynamics modulated by control actions and physical attributes via AdaLN.

point clouds) can instead be encoded by the corresponding geometric encoder into fixed-shape object-level or part-level tokens, while the overall representation interface remains unchanged. Additional implementation details for complex objects are provided in Appendix B.3.

3.2 OrbiSim-Dynamics: Object-Centric Physics Core

OrbiSim-Dynamics models the evolution of physical states as a deterministic, history-aware recurrent process. As shown in Figure 2, given the past states $x_{0:t-1}$, the previous action a_{t-1}, and static scene descriptors $\bar{x}$, the dynamics core predicts the subsequent state:

$$\hat{x}_t = f_\phi^{\text{dyn}}(x_{0:t-1}, a_{t-1}, \bar{x}). \tag{1}$$

To capture the complex multi-object interactions, OrbiSim-Dynamics employs an *object-centric paradigm*. Each entity (the robot and individual objects) is treated as a discrete token with its own latent and recurrent memory. Static object attributes, world configurations, and control actions are incorporated as conditioning signals via adaptive normalization, providing structured modulation of the dynamics. We employ a recurrent state space model structured as follows:

$$\begin{cases} z_t = f_\phi^{\text{enc}}(h_t, x_t), \quad h_t = f_\phi^{\text{rec}}(h_{t-1}, e_t), \quad \hat{z}_t = f_\phi^{\text{tra}}(h_t), \quad \hat{x}_t = f_\phi^{\text{dec}}(\hat{z}_t), \\ \textbf{Coupling module:} \quad e_t = f_\phi^{\text{cp}}(z_{t-1}, a_{t-1}, \bar{x}). \end{cases} \tag{2}$$

The execution begins by mapping the physical state x_{t-1} and corresponding recurrent hidden states h_{t-1} into a sequence of latent tokens $z_{t-1} = \{z_{t-1}^p, z_{t-1}^{(1)}, \ldots, z_{t-1}^{(N)}\}$, where the first token is for the robot. These tokens are processed by a *Transformer-based Coupling Module* utilizing global self-attention to model inter-entity dependencies, such as collisions and articulated constraints. We inject physical constants and control signals through Adaptive Layer Normalization (AdaLN) to modulate the dynamics flow, where the token feature u represents physical states and the condition c encodes actions and static scene descriptors:

$$\text{AdaLN}(u, c) = \gamma(c) \odot \text{LN}(u) + \beta(c), \tag{3}$$

where $\gamma(\cdot)$ and $\beta(\cdot)$ are learnable functions of the condition. Specifically, the robot token is modulated by the action a_{t-1} and world configurations $\bar{x}^w$, while object tokens are modulated by their respective attributes $\bar{x}^{(i)}$ (*e.g.,* mass, friction) and $\bar{x}^w$. This structured modulation provides the model with the necessary inductive bias to generalize across varying physical environments.

The coupling module outputs physics-aware embeddings $e_t = \{e_t^p, e_t^{(1)}, \ldots, e_t^{(N)}\}$, which are then combined with h_{t-1} to update h_t. From the updated recurrent states, a transition module produces the predicted latent states: $\hat{z}_t$, which are finally decoded into system state predictions: $\hat{x}_t$.

OrbiSim-Dynamics is trained with three loss terms, including the latent alignment loss ($\mathcal{L}_{\text{tra}}$), the encoding consistency loss ($\mathcal{L}_{\text{enc}}$), and the state reconstruction loss ($\mathcal{L}_{\text{dec}}$):

$$\mathcal{L}_{\text{tra}} = \|\hat{z}_t - \text{sg}(z_t)\|^2, \quad \mathcal{L}_{\text{enc}} = \|z_t - \text{sg}(\hat{z}_t)\|^2, \quad \mathcal{L}_{\text{dec}} = \mathcal{L}_{\text{state}}(x_t, \hat{x}_t), \tag{4}$$

where $\text{sg}(\cdot)$ denotes gradient stopping. $\mathcal{L}_{\text{state}}$ may employ different loss forms for different physical state components (see Appendix B.1). For long-horizon stability, the total objective $\mathcal{L}_{\text{dyn}}$ is optimized over T-step autoregressive rollouts over an episode $\{\mathcal{D}_t = (x_t, z_t, \hat{x}_t, \hat{z}_t)\}_{t=1}^T$.

3.3 ORBISIM-VISION: STATE-GUIDED DIFFUSION

OrbiSim-Vision transforms predicted physical states into high-fidelity visual observations $\hat{o}_t$ through a conditional image generation module. As shown in Figure 7, We formulate this as a *Latent Diffusion Process*, which is grounded in the scene context $\bar{x}$ and the physical state $\hat{x}_t$ produced by OrbiSim-Dynamics:

$$\hat{o}_t \sim p_\phi^{\text{vis}}(o_t \mid o_{t-K:t-1}, \hat{x}_t, \bar{x}), \tag{5}$$

where the denoising process is carried out in the latent space and explicitly conditioned on the predicted physical state. We represent the visual latent at time t and diffusion step (s) as $y_{t,(s)}$, spanning from clean ($s = 0$) to noisiest ($s = S$) states. The preceding K observations are VAE-encoded into clean latents $y_{t-K:t-1,(0)}$, which serve as the context for the next-frame generation. We assemble a sequence $y_{t-K:t}$ by appending an initialization latent to this context, which is set to $y_{t,(0)}$ during training and a zero placeholder during inference.

Once the forward diffusion process reaches the noisy state $y_{t-K:t,(S)}$, the target frame is generated through a denoising process conditioned on the clean context latents. First, we achieve *visual anchoring* by concatenating a perturbed version of the most recent latent $y_{t-1,(0)}$ channel-wise to the denoising sequence $y_{t-K:t,(S)}$. Second, *physical grounding* is incorporated by encoding the predicted physical state $\hat{x}_t$ and static scene descriptors $\bar{x}$ into two complementary representations:

- *Spatial condition maps:* These provide spatially aligned geometric cues, such as heatmaps or depth-like priors, which are concatenated directly to the U-Net feature maps.
- *Object-level tokens:* These facilitate cross-attention modulation within the U-Net, enabling the model to attend to specific object states during denoising.

The U-Net iteratively refines the sequence for S steps, and the final clean latent $\hat{y}_{t,(0)}$ is decoded into pixel space. After the denoising process, we take the final clean latent at the last time step as the predicted latent representation $\hat{y}_{t,(0)}$. Finally, a VAE decoder maps $\hat{y}_{t,(0)}$ back to pixels to produce $\hat{o}_t$, completing a fully differentiable rendering process conditioned on physical states.

We train the conditional denoising network D_ϕ as follows, which enforces physical consistency by grounding the denoising process in the predicted states:

$$\mathcal{L}_{\text{vis}} = \mathbb{E}_{\sigma,\epsilon}\Big[w(\sigma)\,\big\|\big(D_\phi(y_{t-K:t,(0)} + \sigma\epsilon;\, c_t)\big)_t - y_{t,(0)}\big\|_2^2\Big], \tag{6}$$

where c_t denotes the combined visual and physical conditioning, and $w(\sigma)$ is a noise-dependent weighting function following standard latent diffusion training. During training, we sample a noise scale σ and Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$, and apply noise to the latent sequence.

3.4 GRADIENT-BASED SYSTEM IDENTIFICATION

The integration of OrbiSim-Dynamics and OrbiSim-Vision provides a fully differentiable execution engine, enabling gradient-based system identification (SysID) over simulation rollouts. Accurate simulation requires scene parameters such as material stiffness, damping, and friction, which are often difficult to measure in real environments. Conventional simulators therefore rely on manual calibration or derivative-free search to match simulated and observed trajectories. In contrast, OrbiSim exposes these quantities as optimizable inputs to its differentiable execution graph. Let η denote the unknown physical parameters in the static scene descriptors $\bar{x}$. Given the initial condition and action sequence associated with an observed trajectory, OrbiSim generates state and observation rollouts, denoted by $\hat{x}_{1:T}(\eta)$ and $\hat{o}_{1:T}(\eta)$, respectively.

We identify η by minimizing a trajectory discrepancy. When physical states are available, we use a state-space discrepancy; when only images are observed, we use the corresponding visual discrepancy. The parameter estimate is updated as

$$\eta^{k+1} = \Pi_{\mathcal{E}}\left[\eta^k - \alpha_\eta \nabla_\eta \mathcal{L}_{\text{SysID}}(\eta^k)\right], \tag{7}$$

where $\mathcal{L}_{\text{SysID}}$ is either the state-space or visual discrepancy and $\Pi_{\mathcal{E}}$ enforces valid parameter ranges. State-based identification provides a privileged reference when states are measurable, whereas vision-based identification uses predicted-observation errors to calibrate parameters that are difficult to measure directly.

3.5 Gradient-Based Control Optimization

The differentiable rollout of OrbiSim provides action-level sensitivities for optimizing control objectives. Let $\mathcal{J} = R(x_T)$ be a terminal reward and let the state evolve according to $x_{t+1} = f_\phi^{\text{dyn}}(x_t, a_t, \eta)$. The gradient of the reward with respect to each action is

$$\nabla_{a_t}\mathcal{J} = \frac{\partial R}{\partial x_T}\left(\prod_{k=t+1}^{T-1}\frac{\partial x_{k+1}}{\partial x_k}\right)\frac{\partial x_{t+1}}{\partial a_t}, \qquad t = 0, \ldots, T-1. \tag{8}$$

These action gradients can be propagated through a differentiable policy $a_t = \pi_\theta(x_t)$ to update its parameters,

$$\theta^{k+1} = \theta^k + \alpha_\theta \sum_{t=0}^{T-1} \nabla_{a_t}\mathcal{J}\,\frac{da_t}{d\theta}, \tag{9}$$

or applied directly to an action sequence $A = (a_0, \ldots, a_{T-1})$,

$$A^{k+1} = \Pi_{\mathcal{A}}\left[A^k + \alpha_A\left(\nabla_{a_0}\mathcal{J}, \ldots, \nabla_{a_{T-1}}\mathcal{J}\right)\right]. \tag{10}$$

Here, $\Pi_{\mathcal{A}}$ enforces the valid action constraints. This formulation allows the same differentiable task objective to support either reusable policy learning or task-specific action refinement, depending on the chosen control parameterization.

4 Experiments

We evaluate OrbiSim as both a generative world model and a differentiable execution engine, focusing on (1) generative fidelity and physical consistency under varying configurations, and (2) the benefits of differentiable gradient pathways for downstream policy learning and SysID.

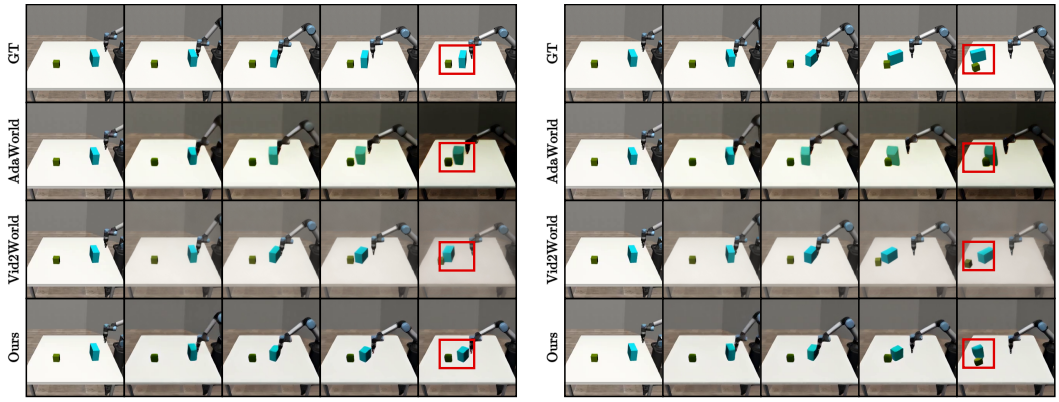
4.1 Experimental Configurations and Baselines

We evaluate OrbiSim on five physics-rich environments built upon classical simulation platforms: • **Robosuite `Push`:** A UR5e manipulator pushes one cube into another under randomized object and physical configurations (Zhu et al., 2020). • **Isaac Lab `Stack`:** A Franka arm sequentially stacks three cubes, testing long-horizon stability and multi-object interaction (Mittal et al., 2025). • **AdaManip `Articulated`:** A robot manipulates articulated objects with diverse shapes and joint mechanisms in the AdaManip environment (Wang et al., 2025; Makoviychuk et al., 2021). • **Physion `Drape`:** A cloth interacts with rigid objects of varying geometries in the Physion Drape scenario (Bear et al., 2021). • **MuJoCo `Cable Lift`:** Two robot arms coordinate to move designated points on a deformable cable toward target locations, with cable stiffness as a key physical factor (Todorov et al., 2012). More dataset details are provided in Appendix A.

We organize our evaluation into three groups. First, for world-model generation quality (Sec. 4.2), we compare OrbiSim with Vid2World (Huang et al., 2025) and AdaWorld (Gao et al., 2025). Second, for downstream policy performance (Sec. 4.3), we compare with representative model-free and model-based RL methods, as well as behavior cloning; detailed experimental settings are provided in

Table 1: **Video-level world modeling performance on the robosuite `Push` task.** We report PSNR and LPIPS at different rollout horizons (10 / 100 steps), together with the overall FVD score. *TrajErr* measures the discrepancy between inferred physical states from generated videos and the corresponding true trajectories. All modes perform autoregressive rollouts from shared initial states.

Method	PSNR10 ↑	PSNR100 ↑	LPIPS10 ↓	LPIPS100 ↓	FVD ↓	TrajErr ↓
Vid2World	22.2014	17.8856	0.1312	0.2551	1750.1	0.6754
AdaWorld	26.6647	12.8346	0.1183	0.3482	1305.8	1.8597
Orbisim w/o Decoupling	**27.9346**	<u>19.9510</u>	0.1188	0.1799	689.1	0.8134
Orbisim w/o Random Sampling	26.6890	19.1119	**0.1076**	0.1669	<u>531.2</u>	0.5742
Orbisim w/o Object-Centric	25.9373	19.7581	0.1123	<u>0.1463</u>	**524.5**	<u>0.4687</u>
Orbisim (Final)	<u>26.7105</u>	**19.9819**	<u>0.1078</u>	**0.1428**	533.9	**0.4468**



(a) High friction (b) Low friction

Figure 3: **Autoregressive simulation under varying physical parameters.** All methods are evaluated with the same initial observations and action sequences. OrbiSim better simulates system dynamics under diverse object-table friction settings.

Appendix C.11. Finally, to demonstrate the advantage of differentiability, we evaluate the MuJoCo `Cable Lift` task in both simulation and the real world. We compare against $\pi_{0.5}$ and SAC in simulation, and deploy $\pi_{0.5}$ as the baseline on an AgileX dual-arm platform.

4.2 Evaluating Generative and Physical Fidelity

As shown in Table 1, OrbiSim (Final) consistently achieves state-of-the-art performance across all metrics and horizons. We compare against **AdaWorld** (Gao et al., 2025) and **Vid2World** (Huang et al., 2025); while these baselines produce visually plausible frames, they struggle with compounding physical drift. In contrast, OrbiSim maintains superior temporal coherence (FVD) and lower trajectory error, demonstrating a more robust alignment between physical dynamics and visual synthesis. Qualitative visualizations in Figure 3. A detailed ablation study of the decoupled architecture, random context sampling, and object-centric dynamics is provided in Appendix C.2.

4.3 Differentiability Evaluation on Policy Optimization

We evaluate OrbiSim as a differentiable simulator for downstream policy learning on the robosuite `Push` task, where the robot pushes one cube to collide with and displace another toward a target region. Rather than using the learned world model solely for prediction, we execute the policy through OrbiSim-Dynamics and analytically backpropagate gradients from the task objective through the predicted physical-state trajectory to the policy network. Detailed reward definitions, success criteria, baseline implementations, and training settings are provided in Appendix C.11.

While the sparse terminal objective makes credit assignment particularly challenging, OrbiSim provides direct task-level optimization signals through its differentiable dynamics. As shown in Figure 4, policies optimized with OrbiSim converge faster and achieve stronger performance than model-free RL baselines, behavior cloning, and the model-based DreamerV3 baseline. These results demonstrate the benefit of treating a world model as a differentiable simulator for policy learning under sparse supervision.

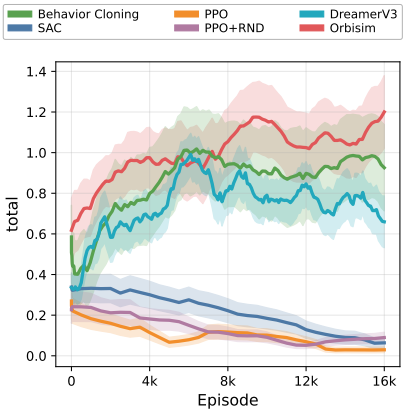


Figure 4: **Training curves on the robosuite `Push` task.** The x-axis denotes training episodes and the y-axis denotes normalized total episode rewards defined in Appendix C.11 and Eq. equation 12.

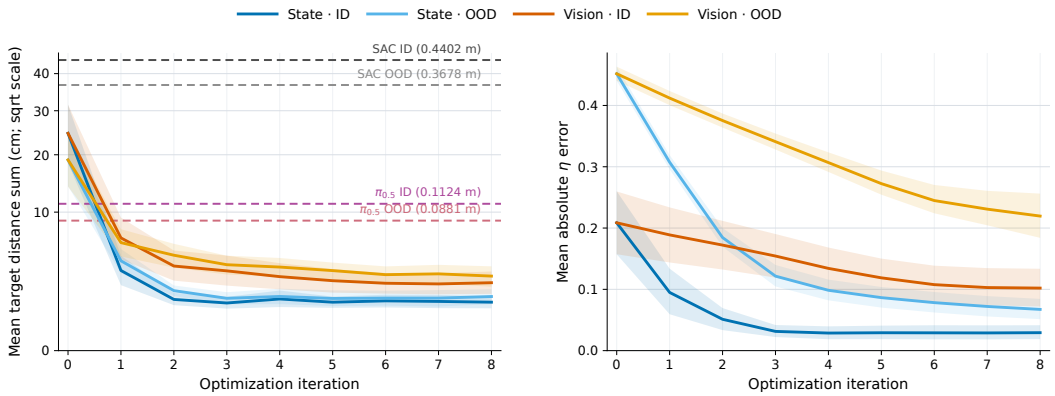


Figure 5: **Simulation results on the `Cable Lift` task.** Terminal keypoint-to-target distance (left) and physical-parameter error (right) over eight alternating rounds. Dashed lines mark the converged SAC and fine-tuned $\pi_{0.5}$ baselines. State/Vision denote state- and image-based SysID on the ID/OOD splits; shading indicates variation across cases.

4.4 Policy Learning via Differentiable Physical-Property Prediction

4.4.1 Task Setup

We consider a MuJoCo `Cable Lift` task in which two robot arms grasp the endpoints of a cable and move within a shared vertical plane. The objective is to align two marked internal points on the cable with their corresponding targets by coordinating the two end effectors. Each episode contains a 32-step action sequence, with each four-dimensional action controlling the planar motion of both end effectors. The cable motion depends strongly on physical properties such as stiffness and damping, which are treated as unknown physical parameters η and estimated from observed trajectories. Together, these settings test whether gradients through dynamics and rendering can unify physical-property identification and control optimization.

4.4.2 Simulation Results

We evaluate on 40 fixed simulation cases, comprising 20 in-distribution and 20 out-of-distribution settings, and use the same initial configurations, targets, and hidden cable properties for all methods. We compare our alternating optimization with SAC and a fine-tuned $\pi_{0.5}$. Task performance is measured by the terminal sum of the two keypoint-to-target distances and the success rate.

Figure 5 shows the optimization dynamics over eight alternating rounds. The state-based variant converges rapidly in both task performance and physical-parameter identification, providing a privileged-observation reference. The vision-based variant consistently reduces both errors across the ID and OOD settings, and its terminal distance falls below the converged SAC and $\pi_{0.5}$ baselines within the first few rounds. This demonstrates that gradients propagated from image-trajectory discrepancies remain informative for identifying cable properties and improving downstream control.

4.4.3 Real-Robot Evaluation

We further deploy the task on an AgileX dual-arm platform using two cables with different material properties. To make the pipeline broadly applicable without relying on task-specific cable-state measurement, our method identifies cable-specific physical parameters from RGB trajectories, with the recorded robot motion used as conditioning information. We then fix the identified physical parameters, optimize the action sequence in OrbiSim, and execute the resulting motion on the robot. We compare this vision-based optimization pipeline with a deployed $\pi_{0.5}$ policy. For each method, we conduct four trials with each cable, for a total of eight real-robot trajectories.

Although both methods are affected by the sim-to-real gap, our approach maintains strong performance across the two cable conditions. Across eight trials, the action sequences optimized with OrbiSim achieve a mean final keypoint-to-target distance of 17.5 ± 6.1 pixels, compared with

62.2 ± 36.3 pixels for $\pi_{0.5}$; full statistics are reported in Table 11 of Appendix C.13. Qualitative results are shown in Figure 6. These results show that differentiable visual system identification can adapt the simulator to previously unseen physical properties and provide useful gradients for real-world control using broadly available visual observations.

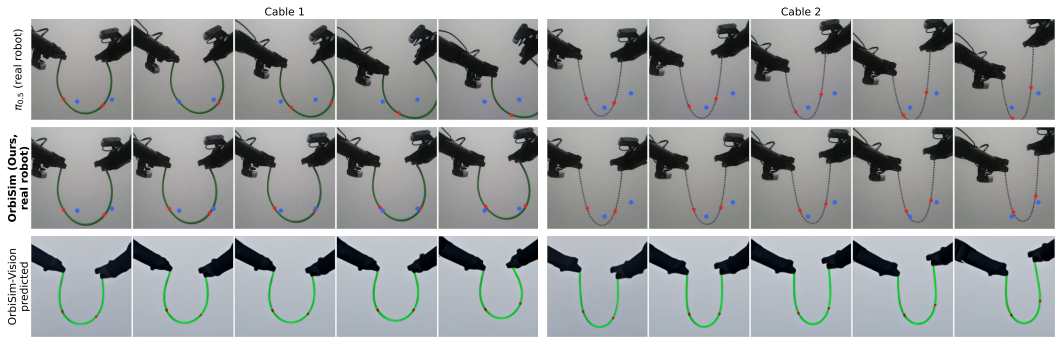


Figure 6: **Real-world evaluation results on the `Cable Lift` task.** We compare the $\pi_{0.5}$ policy with action sequences optimized using OrbiSim on two different cables, with video frames predicted by OrbiSim-Vision shown in the third row. The red dots indicate cable keypoints, and the blue dots indicate their targets. OrbiSim-Vision predicts keypoint locations and cable shapes that closely match the real-world rollouts, while the optimized actions yield smaller final keypoint-to-target distances than $\pi_{0.5}$.

4.5 Additional Evaluation

The content of this section remains to be updated.

In Appendix C, we provide comprehensive supplementary analyses across four primary axes:

- *Generalization analyses:* We evaluate heterogeneous articulated and deformable assets (C.1), the ablation analysis of the decoupled architecture, random context sampling, object-centric dynamics(C.2), and out-of-distribution object counts, shapes, and physical parameters (C.3).
- *Component analyses:* We investigate the isolated predictive performance of OrbiSim-Dynamics (C.4), assess various inference workflows (C.5), and validate the Real-to-Sim performance (C.6).
- *Scalability/robustness*: We evaluate OrbiSim's scalability to multi-object/multi-task settings (C.7) and its robustness under noisy training data that simulates real-world acquisition pipelines (C.8).
- *Extended policy and control studies:* We offer further insights into policy and control optimization, including imitation learning baselines (C.9), comparisons against specialized differentiable simulators such as NVIDIA Newton (C.10), expanded RL experiments (C.11), supplementary simulation and real-world results on the `Cable Lift` task (C.13), and computational efficiency and resource usage analyses (C.12).

5 Conclusions and Limitations

In this paper, we proposed OrbiSim, a novel world model paradigm that rethinks simulation from a learning-centric perspective by treating the world model itself as a fully differentiable physics engine. Through extensive experiments on robotic simulation and control tasks, we demonstrated that OrbiSim outperforms existing world models in future prediction accuracy and control performance.

Despite these encouraging results, OrbiSim still has important limitations. Although it shows promising robustness within related simulator families and already supports heterogeneous assets, its generalization to broader robot morphologies, scene sources, and real-world domains remains underexplored, and it is still far from a complete, general-purpose industrial simulator. Nonetheless, its fully differentiable formulation and scalability in data-rich settings make OrbiSim a promising step toward practical deployment.

AI USE DISCLOSURE

Generative AI tools and large language models were utilized exclusively for language polishing, grammatical refinement, and schematic figure drafting during manuscript preparation. AI was not employed to conceptualize research questions, design the core methodology, conduct physical or simulated experiments, or interpret scientific results. All algorithmic formulations, experimental analyses, and manuscript contents were conceived and verified by the authors, who assume full intellectual responsibility for this work.

REFERENCES

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.

Daniel M. Bear, Elias Wang, Damian Mrowca, Felix J. Binder, H Tung, R. T. Pramod, Cameron Holdaway, Sirui Tao, Kevin A. Smith, Li Fei-Fei, Nancy G. Kanwisher, Joshua B. Tenenbaum, Daniel Yamins, and Judith E. Fan. Physion: Evaluating physical prediction from vision in humans and machines. *ArXiv*, abs/2106.08261, 2021.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, et al. Motus: A unified latent action world model. *arXiv preprint arXiv:2512.13030*, 2025.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *ICML*, 2024.

Maxime Burchi and Radu Timofte. Learning transformer-based world models with contrastive predictive coding. *arXiv preprint arXiv:2503.04416*, 2025.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Lior Cohen, Kaixin Wang, Bingyi Kang, and Shie Mannor. Improving token-based world models with parallel observation prediction. *arXiv preprint arXiv:2402.05643*, 2024.

Erwin Coumans. Bullet physics simulation. In *SIGGRAPH*, 2015.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

Shenyuan Gao, Siyuan Zhou, Yilun Du, Jun Zhang, and Chuang Gan. Adaworld: Learning adaptable world models with latent actions. In *ICML*, 2025.

Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, et al. Dreamdojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026.

David Ha and Jürgen Schmidhuber. World models. In *NeurIPS*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, pp. 2555–2565, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pp. 1–7, 2025.

Xianglong He, Chunli Peng, Zexiang Liu, Boyang Wang, Yifan Zhang, Qi Cui, Fei Kang, Biao Jiang, Mengyin An, Yangyang Ren, et al. Matrix-game 2.0: An open-source, real-time, and streaming interactive world model. *arXiv preprint arXiv:2508.13009*, 2025.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

Siqiao Huang, Jialong Wu, Qixing Zhou, Shangchen Miao, and Mingsheng Long. Vid2world: Crafting video diffusion models to interactive world models. *arXiv preprint arXiv: 2505.14357*, 2025.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, 2004.

Jiajian Li, Qi Wang, Yunbo Wang, Xin Jin, Yang Li, Wenjun Zeng, and Xiaokang Yang. Open-world reinforcement learning over long short-term imagination. In *ICLR*, 2025a.

Wenxuan Li, Hang Zhao, Zhiyuan Yu, Yu Du, Qin Zou, Ruizhen Hu, and Kai Xu. Pin-wm: Learning physics-informed world models for non-prehensile manipulation. *arXiv preprint arXiv:2504.16693*, 2025b.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. *arXiv preprint arXiv:1810.05762*, 2018.

Yifan Lu, Xuanchi Ren, Jiawei Yang, Tianchang Shen, Zhangjie Wu, Jun Gao, Yue Wang, Siheng Chen, Mike Chen, Sanja Fidler, et al. Infinicube: Unbounded and controllable dynamic 3d driving scene generation with world-guided video models. *arXiv preprint arXiv:2412.03934*, 2024.

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

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. *arXiv preprint arXiv:2108.10470*, 2021.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

J Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. gradsim: Differentiable simulation for system identification and visuomotor control. In *ICLR*, 2020.

NVIDIA Corporation. Physx physics sdk, 2020. URL https://developer.nvidia.com/physx-sdk.

Başak Melis Öcal, Maxim Tatarchenko, Sezer Karaoğlu, and Theo Gevers. Sceneteller: Language-to-3d scene generation. In *ECCV*, pp. 362–378, 2024.

Minting Pan, Xiangming Zhu, Yitao Zheng, Yunbo Wang, and Xiaokang Yang. Model-based reinforcement learning with isolated imaginations. *TPAMI*, 46(5):2788–2803, 2024.

Julian Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. WorldGym: World Model as An Environment for Policy Evaluation. *arXiv preprint arXiv:2506.00613*, 2025.

Marc Rigter, Tarun Gupta, Agrin Hilmkil, and Chao Ma. Avid: Adapting video diffusion models to world models. *arXiv preprint arXiv:2410.12822*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ruixiang Sun, Hongyu Zang, Xin Li, and Riashat Islam. Learning latent dynamic robust representations for world models. *arXiv preprint arXiv:2405.06263*, 2024.

Robbyant Team, Zelin Gao, Qiuyu Wang, Yanhong Zeng, Jiapeng Zhu, Ka Leong Cheng, Yixuan Li, Hanlin Wang, Yinghao Xu, Shuailei Ma, et al. Advancing open-source world models. *arXiv preprint arXiv:2601.20540*, 2026.

Bahey Tharwat, Yara Nasser, Ali Abouzeid, and Ian Reid. Latent action pretraining through world modeling. *arXiv preprint arXiv:2509.18428*, 2025.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033, 2012.

Yuanfei Wang, Xiaojie Zhang, Ruihai Wu, Yu Li, Yan Shen, Mingdong Wu, Zhaofeng He, Yizhou Wang, and Hao Dong. Adamanip: Adaptive articulated object manipulation environments and policy learning. *ArXiv*, abs/2502.11124, 2025.

Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler, faster, stronger. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4840–4851, 2023.

Xiaoyang Wu, Daniel DeTone, Duncan P. Frost, Tianwei Shen, Christopher Xie, Nan Yang, Jakob Julian Engel, Richard A. Newcombe, Hengshuang Zhao, and Julian Straub. Sonata: Self-supervised learning of reliable point representations. *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22193–22204, 2025.

Hong-Xing Yu, Haoyi Duan, Charles Herrmann, William T. Freeman, and Jiajun Wu. Wonderworld: Interactive 3d scene generation from a single image. In *CVPR*, 2025.

Fangqi Zhu, Hongtao Wu, Song Guo, Yuxiao Liu, Chilam Cheang, and Tao Kong. Irasim: A fine-grained world model for robot manipulation. In *ICCV*, pp. 9834–9844, 2025.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

APPENDIX

A DATA COLLECTION AND GENERATION

In this appendix, we provide detailed descriptions of the data collection and generation procedures used in our experiments. These details complement the high-level experimental setups presented in the main paper and are included here for completeness and reproducibility.

Data Representation. Across all environments, datasets are organized as episodic trajectories consisting of structured state observations, static environment configurations, and, when applicable, control actions. For robot manipulation tasks, actions are represented as continuous end-effector commands. Each action is a 7-dimensional vector containing translational and rotational commands together with a gripper control signal. The Physion `Drape` task, however, involves passive deformable dynamics of a falling cloth and does not include actions. State observations adopt an object-centric representation that factorizes the world state into robot, rigid-object, articulated-object, or deformable-object components, depending on the task. Static configurations encode asset descriptors and physical attributes, including geometry, joint structure, friction, density, and global scene parameters. This unified episodic format allows OrbiSim to handle both action-conditioned manipulation and passive physical prediction within the same data pipeline.

A.1 ROBOSUITE `PUSH` TASK

Based on the robosuite simulation engine, we construct a trajectory dataset for a cube pushing and hitting task. In each simulation trial, several physical parameters were randomized, including cube friction, density, size, and visual appearance. The pushing velocity of the manipulator was also varied, as it significantly affects contact dynamics and task outcomes.

The task involves two cubes placed on a table and a UR5e manipulator. During an episode, the manipulator pushes the first cube, which then collides with the second cube. A rollout is considered successful if the first cube remains on the table and the second cube reaches a designated region near the target edge without falling off the other edges.

To ensure broad coverage of the task domain, the dataset was collected using a fixed multi-stage policy that generates trajectories from three distinct phases: (a) random manipulator movements, (b) pushing a cube toward a random direction, and (c) pushing a cube to hit another cube. Specifically, the manipulator first moves to a position behind the first cube, then executes a push toward a sampled direction, and finally remains stationary.

During the simulation, we recorded the full physical states of both cubes and the manipulator, the executed control actions, as well as RGB images captured from a side-view camera. In total, the

dataset contains 120 trajectories from phase (a), 640 trajectories from phase (b), and 1,840 trajectories from phase (c). All trajectories feature randomized physical properties and visual appearances to promote diversity and robustness.

A.2 ROBOSUITE TASKS WITH MULTI-OBJECTS

To introduce scalability and variety to the robosuite `Push` task, we subsequently construct two more tasks in robosuite.

The first task, `Push_cubes`, is an extension of the robosuite `Push` task, which also involves pushing and hitting cubes. The task is separated into four subtasks, namely, (a) randomly pushing (random), (b) pushing stacked cubes (stacked), (c) pushing toward an object (towards_object), and (d) pushing to hit an object (to_hit). The first two subtasks have randomized 3 to 4 objects, extending the number of objects. After finishing the task, the robot arm is set to randomly move with smoothed trajectories in order to extend the robot arm position coverage.

The second task is robosuite `Pick_and_place`, which requires the gripper to randomly pick cubes and place them. This task is separated into three subtasks: (a) stack cubes, (b) unstack initially stacked cubes, and (c) pick and place scattered cubes. The cubes' positions, robot arm picking order, moving keypoint trajectories, and cubes dropping heights are randomized during rollout.

The above two tasks contain most of the interaction types of robot arms in robosuite, including pushing, hitting, falling, gripper picking, etc. Each subtask of the two tasks has 420 episodes collected.

A.3 ISAAC LAB STACK TASK

We additionally construct a trajectory dataset based on the Isaac Lab cube stacking task, where a Franka manipulator is required to sequentially stack multiple cubes in a physics-based simulation environment (`Isaac-Stack-Cube-Franka-IK-Rel`).

The dataset consists of episodic trajectories collected from interactions in the simulation, with an average episode length of approximately 230 timesteps. Each trajectory records the full sequence of states and actions throughout an episode. The final dataset includes approximately 1,000 successful trajectories and 1,200 failure trajectories, providing balanced coverage of diverse task execution outcomes.

The dataset is derived from a small set of human expert demonstrations. Starting from these demonstrations, we follow the standard Isaac Lab Mimic pipeline to automatically generate additional state-based trajectories, substantially increasing the scale and diversity of the dataset. This process enables the construction of a large collection of trajectories while relying on only a limited number of manually recorded demonstrations.

After data generation, the outputs produced by the official Isaac Lab scripts are reorganized and redirected into a unified episodic format that conforms to the dataset structure. This reorganization step preserves the original trajectory content while making the data compatible with our world model learning and downstream reinforcement learning pipelines.

For downstream reinforcement learning, we formulate a sparse episodic control task on top of the stacking environment. The reward is defined based on the final stable height of the stacked objects, measured after all object velocities have decayed to zero. This reward is provided once per episode and directly reflects task completion quality.

A.4 ADAMANIP ARTICULATED TASK

We construct an articulated-object manipulation dataset based on AdaManip (Wang et al., 2025). The dataset covers nine categories of articulated objects, with a total of 210 asset instances. Each asset is represented as a structured articulated object, consisting of geometry files together with an articulation specification that defines movable links, joints, and task-relevant operable parts. The assets span diverse mechanisms, including rotating and sliding caps, lockable handles, push-or-rotate controls, and switch-contact interactions. Table 2 summarizes the asset categories used in our experiments.

Table 2: **Articulated-object assets used in the AdaManip task.** We use nine categories with different articulated mechanisms and operable parts.

Category	# Instances	Mechanism / Operable Parts
Bottle	42	cap; rotate-and-slide interaction
Pen	26	cap; rotate-and-slide interaction
Coffee Machine	17	portafilter; rotate-and-slide interaction
Window	18	handle; lock and random rotation direction
Pressure Cooker	18	handle / lid; rotate-and-slide interaction
Lamp	24	button / knob; push-or-rotate ambiguity
Door	31	handle; lock and random rotation direction
Safe	18	door / knob; lock and switch-contact interaction
Microwave	16	door / button; lock and switch-contact interaction

For data collection, we follow the policy and data-generation scripts provided by AdaManip. For each asset instance in every object category, we collect 20 episodes, and each episode contains 301 simulation steps. This results in 4,200 articulated-object manipulation episodes in total. Each episode records the executed robot actions, structured robot and object states, articulated joint states, static asset descriptors, and rendered visual observations, and is converted into the same episodic format used by the other tasks.

During trajectory generation, the camera pose and the initial robot state are kept fixed across episodes to ensure a consistent observation and control setup. To introduce controlled variation, we randomize the initial pose of the manipulated articulated object: its position is perturbed within ± 5 cm, and its orientation angle is perturbed within $\pm 10^{\circ}$. This procedure preserves the underlying articulated mechanism of each asset while providing diverse initial configurations for learning object-centric articulated dynamics.

A.5 PHYSION DRAPE TASK

We adopted Physion's point cloud dataset for arbitrarily shaped, deformable object dynamics. The Drape scene in Physion includes both irregularly shaped objects and a deformable cloth. At the start, a cloth is placed flat in the air, while several objects are placed on the floor. The objects include balls, cuboids, cones, cylinders, lamps, chairs, and so on. Then, the cloth in the air falls down, draping over the objects on the floor. During the scene, the cloth moves and deforms according to the objects' shapes, which requires OrbiSim-Dynamics to learn the deformation dynamics. The Physion dataset does not cover physical parameters or actions.

B IMPLEMENTATION DETAILS

B.1 ORBISIM-DYNAMICS

We train the dynamics module using a two-stage procedure to balance one-step prediction accuracy and long-horizon rollout stability. In the first stage, the model is optimized with teacher forcing, where ground-truth physical states are provided as inputs at every timestep. This stage focuses on learning accurate single-step state transitions while avoiding early error accumulation. In the second stage, we fine-tune the model under autoregressive rollouts by gradually replacing ground-truth inputs with the model's own predictions. The transition from teacher-forced training to fully autoregressive execution is controlled by an autoregressive ratio that increases over training epochs following a cosine annealing schedule, improving robustness to compounding errors.

The dynamics model predicts explicit physical states for the robot and all manipulated objects. The robot state consists of position, linear velocity, and orientation represented as a flattened rotation matrix. Each object state includes position, linear velocity, and orientation represented as a unit quaternion.

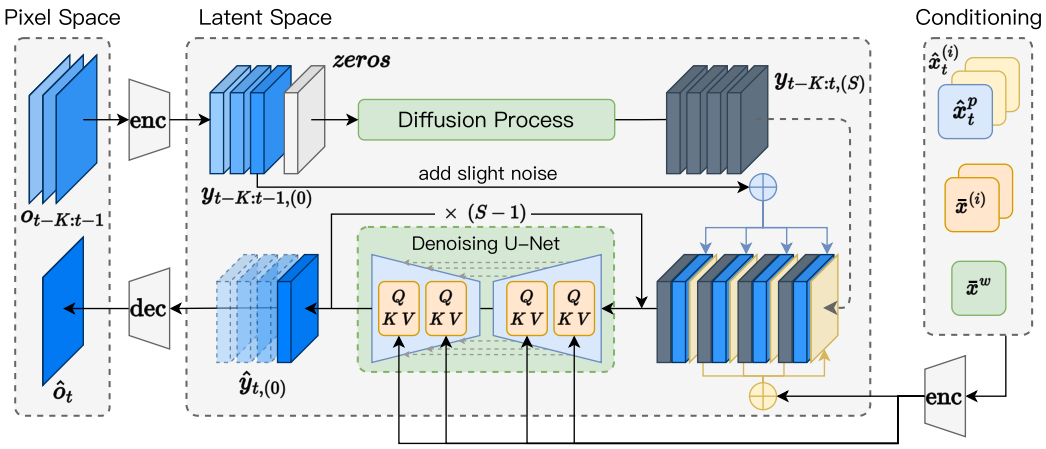


Figure 7: **OrbiSim-Vision**. The denoising process is grounded in predicted physical states through spatial condition maps and object tokens, with context frames providing supplementary visual details.

We supervise state predictions using a composite state loss, where robot and object losses are computed separately:

$$\mathcal{L}_{\text{state}} = \mathcal{L}_{\text{robot}} + \mathcal{L}_{\text{object}}. \tag{11}$$

For translational position and velocity, we apply a Smooth-ℓ_1 loss. For rotational components, we use geometry-aware distance metrics: The robot orientation loss is defined as the geodesic distance between predicted and ground-truth rotation matrices on $\mathrm{SO}(3)$, while object orientations are supervised using a normalized quaternion distance. Object losses are first averaged over objects and then over the batch.

During evaluation, we report a trajectory error (TrajErr) using the same state loss formulation. Specifically, an inference module maps generated video sequences to explicit physical state trajectories, which are then compared against ground-truth trajectories using $\mathcal{L}_{\text{state}}$. This metric directly measures long-horizon physical consistency of model rollouts and is aligned with the training objective of the dynamics module.

Model parameters are optimized using AdamW with a cosine-decayed learning rate schedule across all training stages.

B.2 OrbiSim-Vision

OrbiSim-Vision is trained as a latent diffusion model that generates visual observations conditioned on predicted physical states and scene descriptors. In this section, we focus on training objectives and data sampling strategies that improve robustness under different rollout conditions.

We adopt the standard diffusion denoising objective in the latent space. Each target frame is first encoded into a latent representation using a VAE encoder, and a forward diffusion process is applied to obtain noisy latents. The denoising network is trained to recover the clean latent conditioned on the predicted physical state and available context frame latents.

To improve robustness to varying rollout settings, we randomize the number of context frames during training. Specifically, for each training sample, we uniformly sample the context length $K \in \{0, 1, \ldots, 6\}$ and condition the model on the corresponding past frames. Importantly, we include the case $K = 0$, which explicitly trains the model to reconstruct images solely from the given physical state without relying on visual context. This encourages stable state-to-image rendering when context frames are absent or unreliable.

In addition to varying the context length, we randomize how context frames are selected. With a certain probability, we use temporally consecutive context frames and predict the next consecutive frame. Otherwise, we randomly sample $(K+1)$ frames from a full episode at different timestamps, sort them by temporal order, and use the first K frames as context to predict the $(K+1)$-th frame.

By increasing the temporal gaps between context frames, this stochastic sampling strategy reduces over-reliance on short-term visual continuity and encourages the model to generate observations that are consistent with the provided physical state.

B.3 Extension to Complex Objects

We extend OrbiSim to more complicated object types by adapting the construction of static scene descriptors and physical states while preserving the same object-centric dynamics formulation. For articulated objects, the key challenge is to encode structured multi-part geometry together with articulation priors. For deformable objects, the main difficulty is that object geometry evolves over time and therefore cannot be treated purely as a static asset attribute.

Articulated objects. For articulated objects, we represent each asset as a collection of semantic part-level tokens rather than encoding the entire object as a single rigid entity. Concretely, we first use the URDF specification to determine the part decomposition of the asset, the correspondence between each part and its associated mesh, and the articulation metadata, including joint type, axis, limits, and parent-child relations. For each part, we then recover its triangle mesh, transform it into the corresponding part-local visual frame according to the URDF scale and visual origin, and uniformly sample surface point clouds with normals. The sampled point cloud is normalized and passed through a pretrained Sonata (Wu et al., 2025)/PTV3 (Wu et al., 2023) encoder, and the resulting point features are deterministically pooled into a fixed-dimensional latent representation. The final part-level latent, together with the associated geometric and joint metadata, is stored as the asset attribute of that part. In this way, an articulated object is converted into a structured set of part-level descriptors that capture both geometry and articulation priors, and can be directly incorporated into the static scene descriptor of OrbiSim-Dynamics.

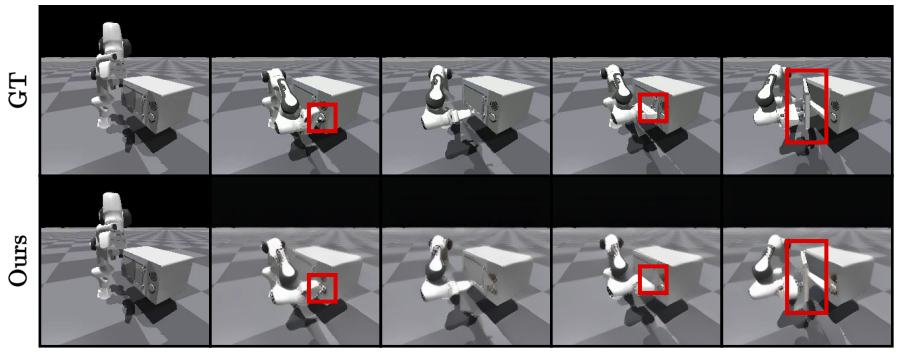
Deformable objects. For deformable objects, we adopt a different treatment because the object shape is inherently time-dependent. In the Drape (Bear et al., 2021) environment, the static objects on the floor remain fixed throughout the episode and, for simplicity, are treated as one large static object, while the cloth is modeled as a separate deformable object. We extract point-cloud-based embeddings for both the static scene object and the deformable cloth using Point Transformer V3 (Wu et al., 2023), yielding object-level features whose dimensionality depends only on the number of objects rather than the number of sampled points. This matches the object-centric encoder design of OrbiSim-Dynamics and allows deformable scenes to be represented within the same structured input interface. Unlike rigid or articulated assets, however, the shape of the cloth changes over time and therefore cannot be encoded only as a static attribute. Instead, the corresponding shape features are treated as part of the physical state and updated along the temporal rollout. This design enables OrbiSim to naturally extend from rigid-body manipulation to deformable dynamics while maintaining a unified object-centric representation.

C More Experimental Results

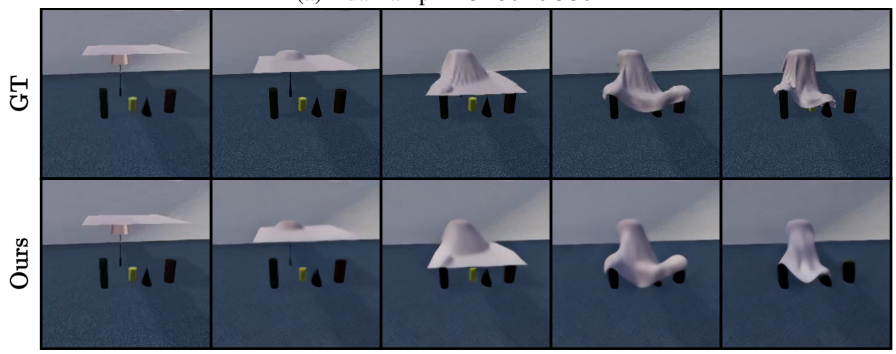
C.1 Generalization to Heterogeneous Assets

As discussed in Sec. 3.1, OrbiSim is designed as a general-purpose, asset-conditioned framework that can accommodate heterogeneous object types through appropriate state and asset encodings. We therefore evaluate this extensibility on two additional complex-object settings: articulated-object manipulation from AdaManip (Wang et al., 2025) and deformable cloth dynamics from the Physion Drape scenario (Bear et al., 2021). For articulated objects, object assets are represented by structured 3D asset files, such as URDF and mesh files, together with joint states. For deformable objects, cloth geometry is represented using point-based states. We encode these geometry-conditioned descriptors with a geometric encoder, such as Point Transformer V3 (Wu et al., 2023), and feed the resulting asset tokens into the same object-centric dynamics and state-guided vision modules. More implementation details are provided in Appendix B.3.

Figure 8 shows qualitative autoregressive rollouts on both settings. OrbiSim accurately follows joint-constrained part motion in articulated-object manipulation and captures the overall deformation pattern of cloth interacting with rigid objects. These results suggest that the asset-conditioned design



(a) AdaManip `Articulated`



(b) Physion `Drape`

Figure 8: **Autoregressive simulation on complex objects.** (a) Articulated-object manipulation from AdaManip. (b) Deformable cloth dynamics from the Physion Drape scenario. OrbiSim captures both joint-constrained articulated motion and geometry-conditioned cloth deformation under the same asset-conditioned simulation framework.

enables OrbiSim to accommodate substantially richer object geometries and physical properties without redesigning the simulator architecture.

C.2 Ablation Studies

We assess the contribution of core design choices through three variants: • *OrbiSim w/o Decoupling* removes the separation between dynamics and vision, training a single monolithic model to jointly predict states and observations. • *OrbiSim w/o Random Sampling* disables random sparsification of visual context during training, using only contiguous context frames. • *OrbiSim w/o Object-Centric* replaces object-centric dynamics modeling with a global state representation that concatenates the robot state, all object states, object attributes, and world configurations at each time step.

As shown in Table 1, removing the decoupled dynamics–vision architecture improves single-frame reconstruction (higher PSNR) but severely degrades temporal consistency and trajectory accuracy, leading to pronounced long-horizon distortions, confirming the necessity of separating dynamics modeling from visual rendering. Disabling random visual context sampling slightly degrades most metrics and increases trajectory error, indicating that context sparsification improves robustness to autoregressive noise. Replacing object-centric modeling with a global state representation yields consistent but moderate performance drops, with visualizations further revealing impaired modeling of object interactions, highlighting the importance of object-level structure for multi-body dynamics.

C.3 Robustness to Out-of-Distribution Scenarios

We further evaluate the generalization ability of OrbiSim under out-of-distribution (OOD) configurations. We construct three types of OOD scenarios: • *Object-count OOD*, where the model is trained on two-object scenes and evaluated on unseen three-object scenes; • *Shape OOD*, where point-cloud-based object geometries in the test split do not appear during training; and • *Physical-parameter*

Table 3: **Out-of-distribution world modeling performance.** We report PSNR and LPIPS at different rollout horizons (10 / 100 steps), together with the overall FVD score. OOD settings include unseen object counts, unseen shape distributions, and extreme physical parameters.

Method	PSNR10 ↑	PSNR100 ↑	LPIPS10 ↓	LPIPS100 ↓	FVD ↓
AdaWorld (ID)	26.6647	12.8346	0.1183	0.3482	1305.8
OrbiSim (ID)	26.7105	19.9819	**0.1078**	**0.1428**	**533.9**
AdaWorld (OOD)	26.3468	13.2215	0.1228	0.3836	1288.5
OrbiSim (OOD)	**27.1867**	**20.1109**	0.1188	0.1847	597.3

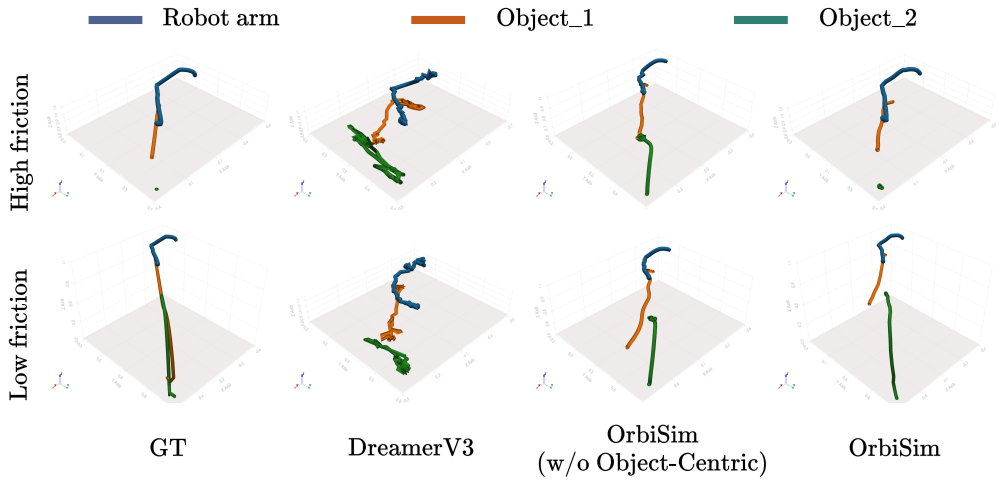


Figure 9: **Physical state trajectories by OrbiSim-Dynamics.** The autoregressive rollouts demonstrate high sensitivity to physical parameters, accurately distinguishing between low- and high-friction regimes while maintaining smoothness over long horizons.

OOD, where we evaluate the model under extreme friction settings that are significantly outside the training distribution in the push-hit task.

Table 3 summarizes the video-level performance under in-distribution and OOD settings. Even under unseen object counts, shapes, and physical conditions, OrbiSim maintains strong generation quality and substantially outperforms AdaWorld, especially over long horizons. These results indicate that the asset-conditioned and object-centric design of OrbiSim provides robust generalization beyond the training distribution, rather than merely memorizing seen scene configurations.

C.4 OrbiSim-Dynamics Evaluation

We evaluate the state-space generative performance of OrbiSim-Dynamics by benchmarking it against DreamerV3-Torch on the robosuite offline dataset. As illustrated in Figure 9, our dynamics core effectively models complex state transitions while demonstrating a high degree of sensitivity to underlying physical configurations.

Specifically, the results show that OrbiSim-Dynamics accurately distinguishes between low- and high-friction regimes, adjusting object trajectories accordingly while maintaining predictive smoothness over long-horizon rollouts. This responsiveness to varying object-table friction coefficients confirms the model's capacity to ground its learned dynamics in explicit physical parameters, a critical requirement for robust system identification and transferable control.

C.5 Workflow Analysis

To disentangle the effects of grounding, autoregressive feedback, and visual context, we evaluate four distinct execution workflows: • *GT-first* initializes rollouts with ground-truth states and observations,

Table 4: **Analysis of OrbiSim rollout variants.** (i) **GT-first:** Autoregressive rollout given true $\{o_0, x_0, \bar{x}\}$; (ii) **GT-state:** Uses true state sequences throughout to isolate rendering performance; (iii) **Recon-first:** Autoregressive rollout only from true initial state; (iv) Recon-all: Evaluates per-frame state-to-pixel fidelity by excluding all temporal visual context during generation, *i.e.,* $p_\phi^{\text{vis}}(o_t \mid \hat{x}_t, \bar{x})$. **TrajErr**: Measures the discrepancy between *inferred physical states from generated videos* (see Sec. 3.4) and the ground truth.

Method	PSNR10 ↑	PSNR100 ↑	LPIPS10 ↓	LPIPS100 ↓	FVD ↓	TrajErr ↓
OrbiSim (GT-first)	<u>26.7105</u>	21.8269	**0.1078**	<u>0.1428</u>	**533.9**	0.4468
OrbiSim (GT-state)	**26.7253**	<u>20.2680</u>	<u>0.1087</u>	**0.1388**	<u>537.3</u>	<u>0.4322</u>
OrbiSim (Recon-first)	24.6366	19.3114	0.1236	0.1509	553.6	0.4558
OrbiSim (Recon-all)	26.4910	**23.3776**	0.1279	0.1628	922.1	**0.2927**

Table 5: **Preliminary evaluation of the Real-to-Sim modules. StaInf** predicts visible scene states and object attributes from a single frame, while **PhyInf** infers hidden physical parameters from a 64-frame video window and is evaluated through rollout consistency under frozen OrbiSim-Dynamics.

	Object Metrics					Robot Metrics		
Module	Position↓	Rotation↓	Size↓	Color(MAE)↓	Velocity↓	Position↓	Velocity↓	Rotation↓
StaInf	27.09mm	7.37°	5.2%	0.0131	–	22.44mm	–	4.08
PhyInf	22.40mm	4.28°	–	–	28.69mm/s	12.70mm	28.63mm/s	1.02°

after which OrbiSim autoregressively predicts both future states and images. • *Recon-first* initializes rollouts from a ground-truth state only, generating the first observation from the state without visual context and then rolling out autoregressively. • *GT-state* replaces autoregressive state predictions with ground-truth state sequences, evaluating visual rendering under perfectly grounded dynamics. • *Recon-all* predicts each observation independently from the corresponding physical state, without using any temporal visual context.

As shown in Table 4, a compelling trade-off is observed in the **Recon-all** variant. It attains substantially higher PSNR due to its bias toward per-frame reconstruction accuracy; however, its degradation in perceptual similarity (LPIPS) and temporal coherence (FVD) indicates a lack of long-horizon visual consistency. In contrast, **GT-first** and **GT-state** strike a superior balance, demonstrating that OrbiSim effectively utilizes temporal context to stabilize long-horizon rollouts.

C.6 Real-to-Sim Experiments

We evaluate the proposed Real-to-Sim pipeline by examining the two inference modules introduced in Sec. 3.4: the state inference module (StaInf) and the physics inference module (PhyInf). StaInf predicts visible scene states and object attributes from a single RGB frame, while PhyInf infers hidden physical parameters from a 64-frame video window through the frozen OrbiSim-Dynamics model.

Table 5 shows that StaInf recovers visible scene quantities with high accuracy, including object position, rotation, size, and color, as well as robot pose. These results indicate that the frozen OrbiSim-Vision representation preserves sufficient geometric and appearance information for decoding structured physical states directly from visual observations.

We further evaluate PhyInf through rollout consistency, *i.e.,* whether the inferred hidden parameters are sufficient to reproduce the observed motion under the frozen OrbiSim-Dynamics model. As shown in Table 5, the resulting rollouts achieve low errors in both object and robot states, including positions, rotations, and velocities. This suggests that the inferred hidden physical parameters are informative enough to support simulation-consistent trajectory prediction and system identification from videos.

Table 6: **Results on multi-object, multi-task datasets.** Single represents OrbiSim and AdaWorld trained on single-task robosuite `Push` dataset with up to 2 objects. Multi represents OrbiSim and AdaWorld trained on the extended `Push_cubes` and `Pick_and_place` dataset with up to 4 objects. For data with more than 100 frames, the results end with "100" are from all frames.

Method	PSNR10 ↑	PSNR100 ↑	LPIPS10 ↓	LPIPS100 ↓	FVD ↓
OrbiSim (single)	26.7105	19.9819	0.1078	0.1428	533.9
AdaWorld (single)	26.6647	12.8346	0.1183	0.3482	1305.8
OrbiSim (multi)	24.8676	18.4515	0.1210	0.1959	863.1
AdaWorld (multi)	24.4685	14.3235	0.1332	0.2661	1292

C.7 Multi Objects / Multi Tasks

We conduct a scaling study focusing on predictive fidelity and memory overhead in multi-task multi-object scenarios.

For predictive fidelity, we evaluate OrbiSim and AdaWorld baseline in the extended multi-object robosuite dataset, namely `Push_cubes` and `Pick_and_place`. Dataset configuration and collection details are discussed in Appendix A.2. These task scenes contain more than 2 objects with a wide range of task settings. These tasks include not only cube pushing, hitting, and friction, but also objects falling, stacking, and robot gripper picking. Thus, these two categories are primitive and fundamental tasks for most robot arm tasks, requiring OrbiSim to learn 3-dimensional object positioning, interaction, and gripping.

The main results of multi-object and multi-task world prediction are in Table 6. Although it experiences performance degradation in multi-object, multi-task scenarios, OrbiSim still achieves higher visual quality and consistency scores.

Additionally, we measured the training memory overhead as the number of objects increased from 2 to 8. The results are shown in Figure 10. As shown in the results, VRAM usage grows approximately linearly. The VRAM usage of OrbiSim-Dynamics is nearly proportional to the number of objects, while OrbiSim-Vision's VRAM growth is tiny compared to its original overhead. This demonstrates that the Transformer-based coupling remains computationally manageable for medium-scale scenes.

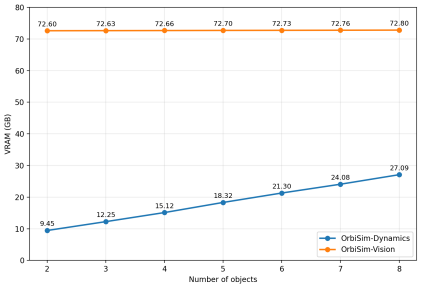


Figure 10: **Memory overhead results from object number scaling.**

C.8 Robustness

We conduct experiments on noisy states to quantify OrbiSim's robustness. We set two levels of Gaussian noise on physical and visual states:

- Moderate Noise: Small-scale perturbations to physical states.
- Strong Noise: Significant noise levels designed to simulate high-error visual state estimation.

Table 7: **Results on noisy states rollout.**

Method	PSNR100 ↑	LPIPS100 ↓	FVD ↓
OrbiSim	19.9819	0.1428	533.9
OrbiSim with moderate noise	20.2508	0.1309	587.14
OrbiSim with strong noise	20.1911	0.1337	606.98

As shown in Table 7, OrbiSim exhibits substantial resilience to noise on both physical and visual states. Even under high levels of visual noise, OrbiSim remains highly robust, exhibiting low degradation in visual generation. This demonstrates that OrbiSim's framework can tolerate imperfection in physical

Table 8: **Imitation learning performance on the Isaac Lab `Stack` task.** The imitation learning baseline is implemented using the official Isaac Lab demonstration-based example without reinforcement learning. Results highlight the sensitivity of pure imitation learning to distributional shifts in long-horizon manipulation tasks.

Method	Input	Success (%)
Isaac Lab-IL (Vision)	RGB + State	78.53
Isaac Lab-IL (Noise)	Noise + State	0.00
Isaac Lab-IL (State)	State	78.00

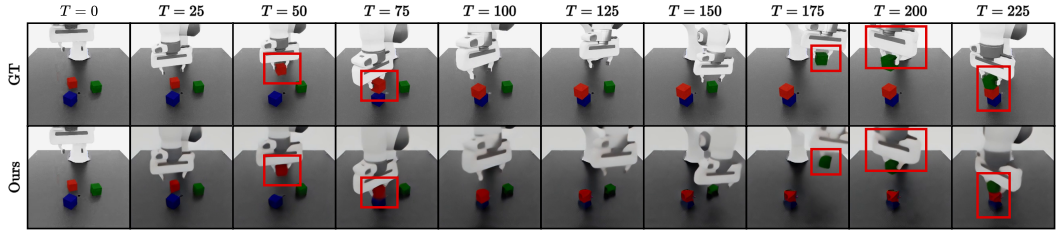


Figure 11: **Autoregressive simulation on the Isaac Lab `Stack` task over a 225-step horizon.** OrbiSim accurately captures intricate manipulator rotations and complex multi-object collisions, maintaining high physical fidelity and stability throughout the sequence.

and visual states, which is promising for potential Real-to-Sim scenarios even where ground-truth states are unavailable.

C.9 Imitation Learning Experiments on the Isaac Lab Stack Task

We further conduct imitation learning experiments on the Isaac Lab `Stack` task, which is substantially more challenging than the pushing task due to its longer horizons, multi-stage object interactions, and stricter requirements on precise state coordination. The stacking task requires not only reaching intermediate configurations, but also maintaining stability under contact-rich dynamics, making it particularly sensitive to compounding errors. Qualitative visualizations of Isaac Lab `Stack` task are in Figure 11, showing the robustness of OrbiSim in 225-step long-term simulations.

As shown in Table 8, the imitation learning baseline (**Isaac Lab-IL**) can achieve reasonable performance when evaluated under in-distribution conditions that closely match the demonstration data. However, its performance degrades catastrophically under out-of-distribution (OOD) settings, such as variations in object configurations or interaction dynamics. This failure mode highlights a fundamental limitation of pure imitation learning: without an explicit mechanism for modeling environment dynamics or long-horizon credit assignment, small deviations from the demonstration manifold can rapidly accumulate and lead to irreversible task failure.

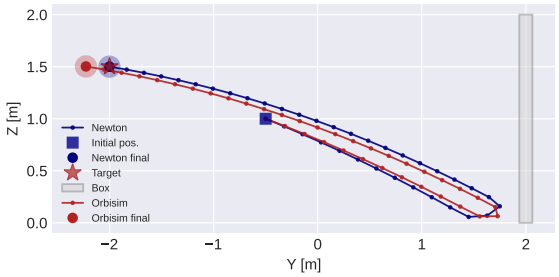


Figure 12: **Rollout trajectory in Newton simulator.** Both trajectories initialized with velocity optimized in Newton and OrbiSim-Dynamcics are shown. OrbiSim-Dynamics' result shows a tolerable difference.

These results motivate the use of reinforcement learning, particularly when augmented with learned world models, to enable robust long-horizon decision making and recovery under distributional shifts.

Table 9: **Success rate on the robosuite `Push` task.** We evaluate OrbiSim, model-based baseline (DreamerV3), Behavior Cloning (BC) and model-free baselines (SAC, PPO, PPO+RND) in the robosuite environment. Success is achieved if the second cube moves more than 5cm.

Method	**OrbiSim**	DreamerV3	BC	PPO+RND	PPO	SAC
Success Rate(%)	**42.71**	25.00	19.79	2.08	1.04	0.00

C.10 ORBISIM-DYNAMICS COMPARED TO NEWTON SIMULATOR

To prove the effectiveness of analytical gradients of OrbiSim, we conducted experiments comparing OrbiSim to differentiable physics simulators. We adopted NVIDIA Newton as a baseline, with its diffsim tasks simulated in Warp differentiable simulator. The `diffsim_ball` task is chosen. At the start, a ball is in the air with an initial velocity. The task requires optimization over initial velocity, making the ball bounce on the floor and a static wall, and finally reach a target position. No robot arms or actions are included in this task. In newton simulator, the optimization is achieved by the raw gradient descent method, utilizing the gradient of the squared target position error through the simulator.

For optimization in OrbiSim, we first collect a dataset by rollout in Newton with randomly-sampled initial velocity. A total of 500 episodes are collected for OrbiSim-Dynamics training. After training, we optimized initial velocity through analytical gradients from OrbiSim-Dynamics with the same method as Newton.

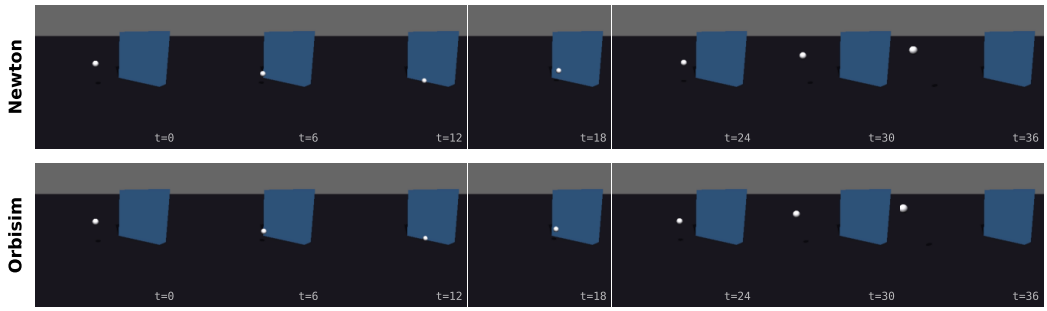


Figure 13: **Visual rollout in Newton simulator.** The two rows represent rollouts with initial velocity optimized with Newton warp and OrbiSim-Dynamics, respectively.

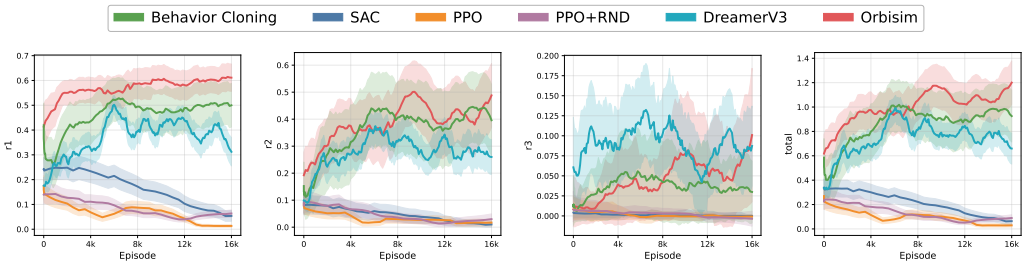
With a learning rate of 0.8, gradient descent in Newton simulator reaches zero loss within 60 steps. We conduct gradient descent with the same learning rate in OrbiSim-Dynamics, and the loss value is reduced to less than 1e-4 within 800 steps.

The visual results are shown in Figure 13 and Figure 12. The initial velocity optimized from newton is $[0.0, 10.639, -4.282]$, while the velocity optimized from OrbiSim-Dynamics is $[-0.07, 11.218, -4.238]$. A tolerable difference is shown in the velocity and rollout trajectories.

OrbiSim shows a slight error in dynamics estimation accuracy compared to simulators. However, differentiable simulators such as Newton DiffSim only support limited scenarios, while OrbiSim supports a wider variety of object types, scene configurations, and physics settings.

C.11 MORE DOWNSTREAM REINFORCEMENT LEARNING EXPERIMENTS

In the robosuite environment, we define three normalized terminal rewards based on distance reduction: r_1 encourages approaching the first cube, r_2 promotes contact between the two cubes, and r_3 measures task progress by pushing the second cube toward the table boundary. Each reward is computed as the relative decrease from its initial distance. For r_1 and r_2, the minimum distance over the episode is used to capture transient interactions, while r_3 uses the final distance to reflect task completion. Specifically, for an episode of T steps, let $d_{\mathrm{r,\,c1},t}$ denote the distance between the robot



(a) r_1 (b) r_2 (c) r_3 (d) Total reward

Figure 14: **Training curves on the robosuite `Push` task.** The x-axis denotes training episodes and the y-axis denotes normalized episode rewards defined in Appendix C.11 and Eq. equation 12, namely r_1, r_2, r_3, and the total reward. See reward definition in the text.

arm (r) and the first cube (c1) at timestep t, and the rewards are defined as:

$$r_1 = 1 - \frac{\min_{t=1}^{T} d_{\text{r, c1},t}}{d_{\text{r, c1},0}}, \quad r_2 = 1 - \frac{\min_{t=1}^{T} d_{\text{c1, c2},t}}{d_{\text{c1, c2},0}}, \quad r_3 = 1 - \frac{x_{\text{table_border}} - x_{\text{c2},T}}{x_{\text{table_border}} - x_{\text{c2},0}}. \tag{12}$$

During training, these three rewards are provided only at episode termination, resulting in sparse learning signals that are challenging for conventional policy optimization methods. We also use these three rewards to evaluate task completion performance across different methods. Success is defined in robosuite `Push` task if the second cube has moved more than 5cm during rollout.

The complete training curves are shown in Figure 14. The success rate comparison is provided in Table 9. OrbiSim performs higher task completion rewards and success rate than all baselines.

To qualitatively evaluate policy performance on the downstream task, we visualize representative rollouts produced by policies trained using model-free RL, model-based RL and Behavior Cloning (BC). For model-free RL, we include Soft Actor-Critic (SAC) (Haarnoja et al., 2018), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and PPO with Random Network Distillation (PPO+RND) (Burda et al., 2018). For model-based RL, we adopt DreamerV3 (Hafner et al., 2025) as a baseline.

As shown in Fig. 15, policies trained with model-free RL fail to learn effective behaviors for the downstream task, resulting in disordered motions and unsuccessful task completion. DreamerV3 policies perform more effectively but still fail in stability and generalization in unseen distributions. Behavior Cloning (BC) policies, while able to reproduce demonstrations to some extent, exhibit unstable performance and frequently deviate from desired behaviors due to the accumulation of prediction errors over long horizons. In contrast, our method consistently produces coherent and goal-directed behaviors, enabling successful task execution across different scenarios. These visual comparisons demonstrate that our approach provides more reliable policy performance on the downstream task compared with both reinforcement learning and imitation learning baselines.

C.12 Computational Efficiency and Resource Usage

We provide additional computational statistics for representative components and baselines used in our experiments. Table 10 reports the training throughput and GPU memory usage measured during downstream policy learning. All measurements are collected under the same simulator and training environment settings.

The results show that OrbiSim achieves substantially higher training throughput than latent world-model approaches such as DreamerV3 while maintaining moderate memory usage. Pure model-free baselines such as SAC achieve higher raw environment throughput, but fail to obtain competitive downstream control performance in our tasks. These observations support the practical benefit of structured end-to-end differentiability for efficient embodied policy optimization.

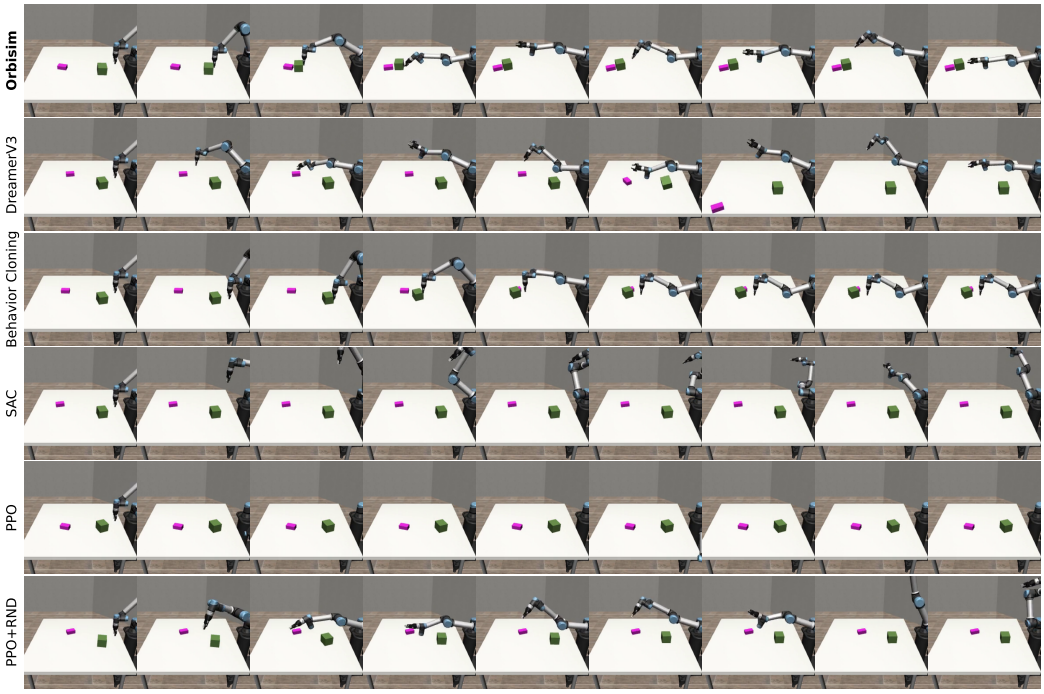


Figure 15: **Reinforcement learning results on robosuite `Push` task.** We show video rollouts in robosuite simulator of OrbiSim model-based policies trained with OrbiSim dynamics, DreamerV3 model-based policies trained with DreamerV3 prediction, behavior cloning trained with expert policies, and model-free baselines trained with robosuite environment.

Table 10: **Training throughput and GPU memory usage of representative methods.**

Method	Steps/s	Device	GPU Memory
OrbiSim dynamics	29.41	RTX 3080	372 MB
Behavior Cloning	47.71	RTX 3080	27.14 MB
DreamerV3	4.54	RTX 4090D	931 MB
SAC	66.05	RTX 3080	33.79 MB

C.13 Simulation and Real-World Results on MuJoCo `Cable Lift` Task

Quantitative results for the real-world deployment of `Cable Lift` task controllers are shown in Table 11. The action sequences optimized with OrbiSim achieve a substantially smaller final keypoint-to-target distance than $\pi_{0.5}$.

Table 11: **Distance mean and standard deviation on the real-world cable task.**

Method	$\pi_{0.5}$	**OrbiSim**
Distance mean (px)	62.2	**17.5**
Distance std (px)	36.3	**6.1**