MANISKILL3: GPU PARALLELIZED SIMULATION AND RENDERING FOR GENERALIZABLE EMBODIED AI

Stone Tao[1,3], **Fanbo Xiang**[1,3], **Arth Shukla**[1,3], **Yuzhe Qin**[1], **Xander Hinrichsen**[1], **Xiaodi Yuan**[1,3]
Chen Bao[2], **Xinsong Lin**[1], **Yulin Liu** [1,3], **Tse-kai Chan**[1], **Yuan Gao**[1], **Xuanlin Li**[1], **Tongzhou Mu**[1]
Nan Xiao[1], **Arnav Gurha**[1], **Viswesh Nagaswamy Rajesh**[1], **Yong Woo Choi**[1], **Yen-Ru Chen**[1]
Zhiao Huang[1,3], **Roberto Calandra**[4], **Rui Chen**[5], **Shan Luo**[6], **Hao Su**[1,3]
[1]University of California San Diego, [2]Carnegie Mellon University, [3]Hillbot, [4]TU Dresden
[5]Tsinghua University, [6]King's College London.

ABSTRACT

Simulation has enabled unprecedented compute-scalable approaches to robotics. However, many existing simulators typically support a narrow range of tasks and lack features critical for scaling generalizable robotics and sim2real. We introduce ManiSkill3, a state-of-the-art state-visual GPU parallelized robotics simulator with contact-rich physics targeting generalizable manipulation. ManiSkill3 supports GPU parallelization of many aspects including simulation+rendering, heterogeneous simulation, pointclouds, and more. GPU simulation+rendering uses 2-4x less GPU memory compared to other platforms and achieves up to 30,000+ FPS in benchmarked environments due to minimal overhead, simulation on the GPU, and the use of the SAPIEN parallel rendering system, enabling visual RL to solve tasks in minutes instead of hours. We further provide the most comprehensive range of tasks spanning 12 distinct domains including but not limited to mobile manipulation, drawing, humanoids, and dextrous manipulation in realistic scenes designed by artists or real-world digital twins. In addition, millions of demonstration frames are provided from motion planning, RL, and teleoperation. ManiSkill3 also provides a comprehensive set of baselines that span popular RL and learning-from-demonstrations algorithms.

1 INTRODUCTION

One of the grand challenges of robotics is robust and generalized manipulation. However, unlike vision and language research, there are still no good datasets for robotic manipulation that can be trained on. One approach has been to create human-scalable real-world teleoperation tools (Fu et al., 2024; Cheng et al., 2024b) to then perform imitation learning. Another is to set up real-world reinforcement learning to fine-tune offline trained policies (Feng et al., 2023). However, real-world imitation learning approaches require enormous amounts of data that are infeasible to collect efficiently at low costs only to achieve relatively low success rates that are otherwise impractical for real-world deployment (Zhao et al., 2024). Real-world reinforcement learning approaches are promising, but require extensive setups to generate real-world rewards and environment resets.

GPU parallelized simulations such as Isaac (Makoviychuk et al., 2021) and Mujoco's MJX (Todorov et al., 2012) have made massive advancements in solving some robotics problems such as robot locomotion by training in large-scale GPU parallelized simulations with reinforcement learning (RL) (Rudin et al., 2021). GPU parallelized simulation makes data incredibly cheap to generate. However, when it comes to manipulation, success is often limited to narrower ranges of manipulation tasks and typically requires strong state estimation (Handa et al., 2023) to replace visual inputs like RGB or pointcloud. Existing GPU simulators have limitations that hinder the generalization and scalability of previous work. These simulators lack support for heterogeneous simulation, where each parallel environment contains different scenes. Additionally, they often don't support fast parallel rendering capabilities. As a result, algorithms like reinforcement learning (RL) that operate on visual input train too slowly to be practical. We propose ManiSkill3 to address past limitations, building upon past work in ManiSkill 1 and 2 (Mu et al., 2021; Gu et al., 2023). The core contributions of ManiSkill3 that set it apart from existing simulators are as follows:

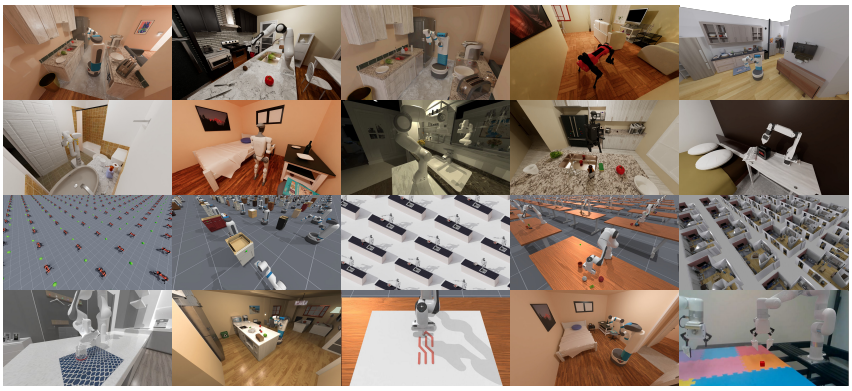


Figure 1: Multiple distinct task categories are displayed, ranging from room-scale tasks to humanoid interactions and drawing tasks. Majority of tasks are GPU-parallelized, simulating+rendering at state-of-the-art speeds and GPU memory efficiency. Scenes are from ReplicaCAD and AI2-THOR.

1) State-of-the-art GPU Parallelized Simulation and Rendering: RL algorithms like PPO (Schulman et al., 2017) can now solve visual tasks faster than before due to fast parallel rendering and low overhead in the system design of ManiSkill3, leading to highly efficient use of the GPU. Depending on task the simulation + rendering FPS can reach up to 30,000+ with 2-4x better GPU memory usage. RGB policies can be trained and deployed zero-shot for simple tasks much faster as well.

2) Most comprehensive range of environments with 12 different categories of environments and 20+ different robots provided out of the box: ManiSkill3 out of the box provides a diverse set of different types of environments including but not limited to mobile manipulation, room-scale scenes, drawing, and humanoid/bi-manual manipulation. Furthermore we support several sim2real and real2sim setups for manipulation. A core focus of ManiSkill3 is building many template/examples that users can then build on top of themselves for their own use-cases.

3) Heterogeneous Simulation for Generalizable Learning: ManiSkill3 makes it possible to simulate and render completely different objects, articulations, even entire room-scale scenes in each parallel environment. This is done thanks to a data-oriented system design and easy-to-use API to manage GPU memory of objects/articulations even if they may have different degrees of freedom.

4) Simple Unified API to Easily Manage and Build GPU Simulated Tasks: ManiSkill3 distinguishes itself from other GPU-parallelized robotics simulators by offering a user-friendly API for creating diverse tasks. Improvements include object-oriented APIs and the elimination of complex tensor indexing. The platform provides feature-rich tooling to streamline various operations, such as visual domain randomization, trajectory replay, controller action conversion, and more.

5) Scalable Dataset Generation Pipeline from Few Demonstrations: For tasks in ManiSkill3 where reward design is difficult, we provide a pipeline that leverages demonstration efficient, wall-time fast, online imitation learning algorithms, to learn a generalized neural network policy from a few teleoperated/hardcoded demonstrations. The generalized task-specific neural network policy is then used to rollout many more demonstrations to form larger datasets.

2 Related Work

Robotics Simulation Frameworks: Isaac Lab (Mittal et al., 2023) and Brax (Freeman et al., 2021) are open-source GPU-parallelized robotics frameworks. Similar to ManiSkill3, Isaac Lab and Brax (which uses the Mujoco MJX (Todorov et al., 2012) backend) provide ready-to-use environments for reinforcement/imitation learning and APIs for custom environments. In contrast, CPU-based frameworks like Robocasa (Nasiriany et al., 2024), Habitat (Szot et al., 2021), AI2THOR (Kolve et al., 2017), OmniGibson (Li et al., 2022), and RLBench (James et al., 2020) run significantly slower, often limiting research to imitation learning and motion planning. Isaac Lab relies on Isaac Sim for GPU-based simulation and rendering while ManiSkill3 uses the open-source SAPIEN (Xiang et al., 2020) for the same features. We note that the provided parallel rendering features are concurrent to Isaac Lab's work.

Feature	ManiSkill3	Isaac Lab	RoboCasa	OmniGibson	Habitat	AI2THOR
Parallelized Simulation	✓	✓	✗	✗	✗	✗
Parallelized Rendering	✓	✓	✗	✗	✗	✗
Parallelized Heterogeneous Sim	✓	✗	✗	✗	✗	✗
Large Scale Demonstrations	✓	✗	✓	✗	✗	✗
Realistic Object Physics	✓	✓	✓	✓	✗	✗
Photorealistic Rendering	✓	✓	✓	✓	✗	✗
Visual RL Baselines	✓	✓	✗	✗	✓	✓
Vision-based sim2real setups	✓	✗	✗	✗	✗	✗
Trajectory replay/conversion	✓	✗	✗	✗	✗	✗
Task Categories	12	5	2	2	1	1

Table 1: Comparison of major features across different open-source robotics frameworks/tools.

Robotics Datasets: Amongst existing datasets there are typically two kinds, real-world and simulated datasets. Open-X (Collaboration et al., 2023) and DROID (Khazatsky et al., 2024) are two of the largest real-world datasets. However, both Open-X and DROID require immense amounts of human labor to collect data and are inherently difficult to scale up to the sizes of typical vision/language datasets. Among simulated datasets, frameworks like AI2-THOR (Kolve et al., 2017), and OmniGibson (Li et al., 2022) have complex room-scale scenes but do not readily provide demonstrations or ways to generate large-scale demonstrations for use in robot learning. Robocasa has a myriad of tasks and realistic room-scale scenes, but further leverages MimicGen (Mandlekar et al., 2023) to scale human teleoperated demonstrations by generating new demonstrations. ManiSkill3 sources large-scale demonstrations through a few methods. For easier tasks, motion planning and rewards for RL are used to generate demonstrations. For more complex tasks without easily defined motion planning solutions or rewards, ManiSkill3 relies on online learning from demonstrations algorithms to train on a few demos and then rollout converged policies to generate more demos.

3 Core Features of ManiSkill3

ManiSkill3 is the most feature-rich GPU simulation framework compared to popular alternatives as shown in Table 1. For the largest features, we detail them in the subsections below.

3.1 Diverse Tasks Supported Out Of The Box

The design of ManiSkill3 enables support for many different kinds of task categories via a flexible task-building API. Of the existing popular robotics simulators ManiSkill3 supports the most categories of different tasks. Concretely we categorize the 12 distinct categories as follows: Table top manipulation, mobile manipulation, room-scale scenes for manipulation, quadruped/humanoid locomotion, humanoid/bi-manual manipulation, multi-agent robotics, drawing/cleaning, dextrous manipulation, vision-tactile manipulation, classic control, digital twins, and soft body manipulation environments. The majority of these tasks are GPU parallelized and can be rendered fast in parallel as well, with examples of the tasks shown in Fig. 1. Each of these task categories have various optimizations done to run more accurately and/or faster. Other simulators typically support a smaller subset of the type of tasks ManiSkill3 supports easily. Additional details on the exact optimizations/implementations and available robots are detailed in Appendix A.

3.2 GPU Parallelized Simulation and Rendering

ManiSkill3 distinguishes itself from its predecessors and other robotics simulators by offering robust support for GPU-parallelized simulation and rendering. ManiSkill3 is the first general benchmark to enable fast RL from visual inputs on complex robot manipulation tasks, with Isaac Lab recently adding a similar feature. Tasks such as picking up a cube or controlling a quadruped to reach a goal from pixel inputs are now solved on the order of minutes instead of hours. RL training results/speed are detailed in Section 4.2. The performance results shown in Figure 4 are the results after simulating + rendering RGB, depth, and segmentation data simultaneously for various tasks. In terms of

speed and GPU memory use, Figure 2 shows ManiSkill3 outperforms Isaac Lab, particularly when it comes to rendering common real-world camera resolutions which can be important for sim2real and real2sim transfer. In particular, with 128 parallel environments for the benchmarked task, ManiSkill3 uses just 3.5GB of GPU memory whereas Isaac Lab uses 14.1GB. The memory efficiency of the ManiSkill3 platform allows for more room for e.g. RL replay buffers or larger neural network models such as large vision language action models. Training and inference can be kept extremely optimized on a single GPU as a result without needing to store any data on the CPU. From experimentation with visual RL, we find that GPU memory efficiency becomes much more important as the FPS gains from more parallel environments become marginal. GPU memory efficiency is especially important for off-policy algorithms like TD-MPC2 (Hansen et al., 2024) and SAC (Haarnoja et al., 2018) that typically maintain replay buffer sizes on the order of $10^5 \sim 10^6$ frames. For example storing RGB data from one 128x128 camera would require at least 9GB of GPU memory for a replay buffer of size $200,000$, which can easily lead to out of memory issues. For more in-depth performance benchmarking results and comparisons of rendered outputs, see Appendix E.

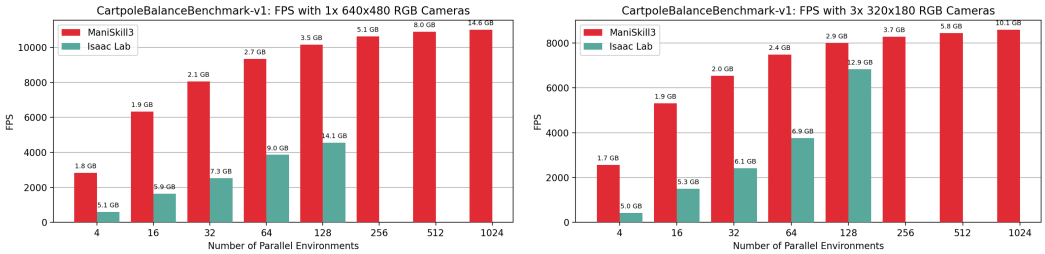


Figure 2: GPU Simulation+Rendering of RGB speeds of the Cartpole environment with different camera setups ManiSkill3 and Isaac Lab. Annotated numbers indicate GPU memory usage, with no data points beyond 128 environments for Isaac Lab due to running out of GPU memory. Note that this rendering setting mimics that of real world datasets collected in Open-X and Droid. Speed is dependent on a few factors, primarily the number of objects, geometry complexity of each object, as well as simulation/rendering configurations which can be tuned for speed or accuracy. As a result, it is possible the numbers/trends here may not hold for every environment.

We acknowledge that this comparison is not strictly apples-to-apples due to differences in rendering techniques. Isaac Lab employs ray-tracing for parallel rendering, while the ManiSkill3 results are generated using SAPIEN's rasterization renderer (see Figure 3 for a visual comparison), although ManiSkill3 also supports a ray-tracing mode without parallelization. Ray-tracing generally offers greater flexibility in balancing rendering speed and quality through the adjustment of parameters such as samples per pixel. Note that the Isaac Lab data presented here uses the fastest rendering settings, but can be tuned to improve rendering quality for e.g. sim2real. Despite the use of different rendering techniques, we believe this provides a meaningful basis for comparison.

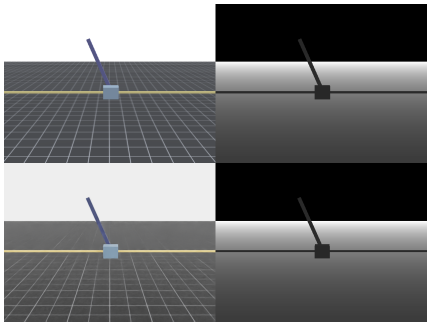


Figure 3: Comparison of ManiSkill3 (Top row) and Isaac Lab (Bottom row) parallel rendering 640x480 RGB and depth images of the Cartpole task.

GPU parallelized simulation and rendering enable an entirely new regime of running efficient domain randomizations. For example you can quickly render over a 1000 different cameras, each with different extrinsics/intrinsics, mounted/fixed, as well as randomize object textures in each of the parallel environments. A subset of 4 out of 1024 environments renders are shown in Figure 5 with different settings. This type of visual diversity in simulation enables much faster training of more visually robust policies and is critical for sim2real applications. Furthermore, ManiSkill3 supports parallelized rendering of voxel/pointcloud formats necessary for 3D robot learning approaches (Ze et al., 2024; Huang et al., 2023; Shridhar et al., 2022).

Finally, ManiSkill3 enables extremely fast visual digital twins. For example, ManiSkill3 implements 4 of the environments in SIMPLER (Li et al., 2024) which are evaluation digital twins

that enable the evaluation of generalist robotic policies trained on real-world data like Octo (Octo Model Team et al., 2024) and Robotics Transformers (RT) (Brohan et al., 2022). ManiSkill3 digital twins can evaluate models like Octo at 60x to 100x the speed of the real world without human supervision, approximately 10x faster than the original digital twin implementations in SIMPLER. The speed increase is due to fast and efficient parallel rendering of large camera resolutions (640x480) and flexible GPU parallelized controllers to match most real-world robot arms/manipulators. More details on sim2real/real2sim support are covered in Section 3.5.

3.3 HETEROGENEOUS GPU SIMULATION

ManiSkill3 is so far the only simulation framework that completely supports heterogeneous GPU simulation. This is the feature of being able to simulate different object geometries, different numbers of objects, and different articulations with different DOFs across parallel environments. For example, in the OpenCabinetDrawer task, for each parallel environment, we build a different cabinet (all with different DOFs) and sample a random drawer link that needs to be opened to succeed. In the Pick Clutter YCB task, we sample a different number of YCB objects in each parallel environment and sample one random object out of the clutter as the goal object to pick. ManiSkill3 easily supports this kind of simulation and further supports rendering these different scenes in parallel all at once with an example 3rd view rendering illustrated in Figure 6. Heterogeneous GPU simulation enables more generalizable manipulation as algorithms like PPO can simultaneously train on every single object from the YCB dataset (Çalli et al., 2015) or the PartNetMobility dataset of cabinets (Mo et al., 2019).

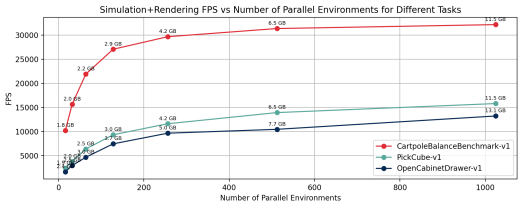


Figure 4: GPU Simulation+Rendering speeds of various tasks with a single 128x128 resolution camera with a simulation frequency of 120 and control frequency of 60, meaning the camera renders ever 2 sim steps. RGB, depth, and segmentation data are all simultaneously being rendered.

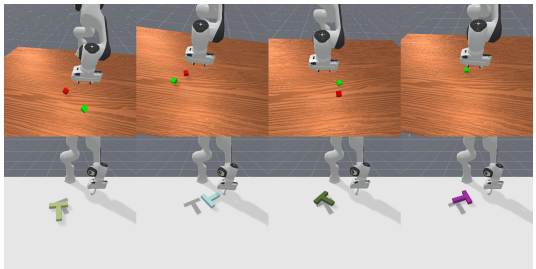


Figure 5: Parallel rendering outputs of 1024 parallel environments for the StackCube and PushT tasks with a subset of 4 them visualized here. Original renders are size 128x128, images shown are up-scaled for clarity. Top-row shows camera pose randomization and bottom row shows texture randomization.

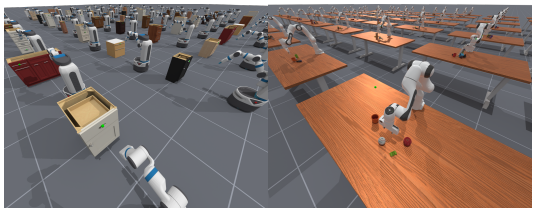


Figure 6: Tasks in ManiSkill3 showing heterogeneous GPU simulation with different DoF articulations and/or different numbers of objects being simulated in each parallel environment.

3.4 TELEOPERATION

We provide a Virtual Reality (VR) Teleoperation system that is seamlessly integrated into ManiSkill3 with immersive visual feedback and supports wired connections. ManiSkill3 receives real-time hand pose data from the teleoperator that is translated into corresponding robot actions. At the same time, the system streams 4K stereo video via Air Light VR (ALVR) to the VR device at 60 Hz, ensuring a smooth and immersive "all-scene view" experience. This allows the user to explore the entire environment freely, making it possible to teleoperate long-horizon tasks as well as more precise tasks by simply looking closer in the simulation. A crucial control feature is "where your hand is, where the end-effector is", which tightly aligns the operator's hand movement with the robot's end-effector. Prior work provide ways to stream video feedback over the internet via a hardware agnostic web app Cheng et al. (2024a); Ding et al. (2024) but tradeoff some ease-of-use for better accessibility (such as long-range remote teleoperation). Our integrated system enables wider viewports and reduce the need for teleoperators to compensate for hand-robot misalignment and visual disparities, visualized in 7. Our VR teleop system is also compatible with real-world teleop-

eration. For real-world operations, we use one or more depth cameras to generate a point cloud of the scene. The point cloud is then rendered and streamed as stereo RGB video to the operator in real time, allowing for effective and immersive control. Example code for configuring this system is provided for teleoperating robot arms or floating robots with multiple grippers/fingers. See Appendix F for more details on this teleoperation system.

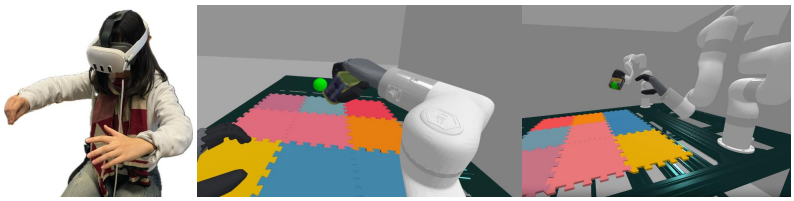


Figure 7: Visualization of VR teleoperation system. Left: A teleoperator using hand poses captured by the Meta Quest 3 headset to control robot motion in real-time. Middle: A 360-degree scene displayed in the VR device, providing immersive sensory feedback. Right: Trajectory replay.

3.5 SIM2REAL AND REAL2SIM FOR ROBOT MANIPULATION

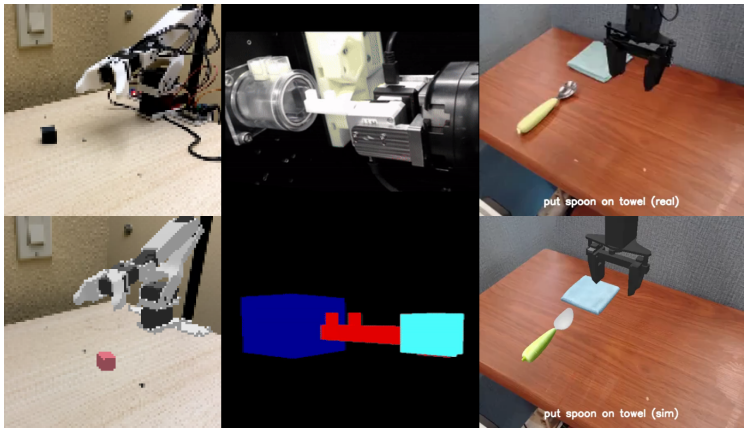


Figure 8: Three different kinds of digital twins in ManiSkill3. Top row shows the real-world setup and bottom row shows the digital twin. Left: Domain randomized digital twin of a cube picking task. Middle: Digital twin of the vision-tactile simulation of a key insertion task. Right: Real2sim digital twin of a spoon placing task.

Towards the goal of robust real-world robotics beyond simulation, we verify sim2real and real2sim are both possible using ManiSkill3 via digital twins on some tasks. Figure 8 showcases several digital twins supported with real world counterparts. For the cube picking task, we train with RL on simulation images and directly deploy to the real world, achieving a real world success rate of 91.6% averaged across 3 RL training runs. See Section 4.3 for more details on rigid body manipulation sim2real. For the vision-tactile peg insertion task, we simulate the tactile sensor made of silicone as a softbody and refer readers to the results showcased in the original work by Chen et al. (2024) for those environments, which achieved a 95.08% success rate in the real world. Finally, for the real2sim digital twins we evaluate Octo and RT-1X on the ManiSkill3 GPU parallelized version of 4 tasks in SIMPLER (Li et al., 2024). We achieve a correlation between real-world success rates and simulation success rates of 0.9284 and a Mean Maximum Rank Violation value 0.0147, close to the original values reported in SIMPLER. See Appendix A.11 for more details on real2sim evaluations.

3.6 SIMPLE UNIFIED API FOR BUILDING GPU PARALLELIZED ROBOTICS TASKS

To enable the flexibility of the ManiSkill3 system to support many different distinct task categories, we provide a clean/simple API for task-building. The API enables users to easily build and customize their own robotics tasks without having to worry about complex GPU memory management details or desigining robot controllers. We describe two features provided by ManiSkill3 below:

3.6.1 Object-Oriented API for Articulations, Links, Joints, and Actors

Figure 9: Code comparison for computing a grasp position on a cabinet handle and the joint angle of the cabinet drawer in 3 different GPU simulation frameworks.

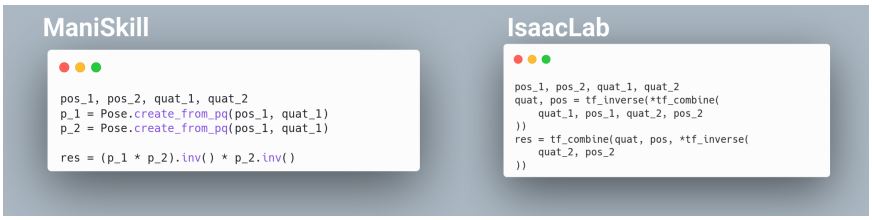


Figure 10: Code comparison for manipulating batched poses

ManiSkill3 is the only framework with a complete object-oriented API around the high-level articulations/actors down to individual links/joints and meshes. In contrast, IsaacGymEnvs requires users to instantiates relevant GPU buffers for holding articulation state such as root pose and joint angles. Isaac Lab improves on this with a partially object-oriented articulation API that allows one to create an articulation object (e.g., for a cabinet). However, one still has to often play around with index values to get the relevant articulation data they need. We use a cabinet opening task as a case study. In a cabinet drawer opening task, to write good reward functions you need to access the drawer link's handle mesh's pose, as well as the joint angle between the drawer and the cabinet. A visual comparison of the 3 APIs (simplified from the actual code) is shown in Figure 9.

Furthermore, pose information in ManiSkill3 is object-oriented and stored as batched Pose objects, enabling an easy to read, method chaining pattern of programming for working with poses. For the sake of an example, suppose that we have 2 poses P_1, P_2 and want to compute $(P_1P_2)^{-1}P_1^{-1}$, ManiSkill3 provides a simpler method chainable API compared to Isaac Lab, as shown in Figure 10.

3.6.2 Robots and Controllers

ManiSkill3 supports both URDF and Mujoco MJCF definition formats natively and builds articulated robots based on the URDF/MJCF directly. For each robot, ManiSkill3 further provides a number of pre-built configurable controller options for both GPU parallelized joint position control and inverse-kinematic (IK) control, modified from ManiSkill2 for GPU simulation. ManiSkill3 builds upon the PyTorch Kinematics package (Zhong et al., 2024) to support inverse-kinematic based controllers parallelized on the GPU, typically used to control robot arm end-effectors. These options are

easily configured at runtime with either preset configurations or user-supplied controller configurations. Currently, there are 20+ different robots supported out of the box in ManiSkill3, a subset of which are visualized in Figure 14. Finally, ManiSkill3 comes with extensive tutorials and examples of how to tune and optimize robots for fast simulation, which has often proven a stumbling block for those new to robot simulation importing complex robots for the first time.

3.7 Demonstration Datasets

We leverage a variety of approaches to collect/generate demonstration datasets infinitely at scale. For easy tasks, we provide motion planning scripts to generate demonstration data. Some tasks with easy-to-define rewards have dense reward functions defined and converged RL policies are used to generate demonstration data. For more difficult tasks, we collect demonstration data (typically about 10 emos) via teleoperation tools. Then, we use RFCL (Tao et al., 2024) or RLPD (Ball et al., 2023) to run fast online imitation learning and generate data from converged policies.

We further adapt the trajectory replay tool from ManiSkill2 to work with both CPU and GPU simulated demonstration data. The replay tool enables users to change the observations stored (e.g., state or rgbd, and allow modifying the rendering shaders used) as well as modifying the rewards stored (dense or sparse). In tasks involving the Franka robot arms, we provide code to convert actions from one controller type to another (e.g. joint position control to delta end effector pose control). Importantly we support collecting data in CPU/GPU simulation and replaying them in CPU/GPU simulation with different numbers of parallel environments via explicit control over randomization/RNG seeding at the per-parallel-environment level. This enables flexibility in trajectory replay as data collected on one machine with more GPU memory can be replayed on other machines with less GPU memory that cannot use as many parallel environments.

4 Baselines and Results

4.1 Reinforcement Learning / Learning From Demonstrations (LfD)

We provide two categories of RL baselines and two categories of LfD baselines as follows:

Wall-time Efficient Reinforcement Learning: We include a torch based vectorized implementation of model-free RL algorithms PPO and SAC (Haarnoja et al., 2018), as well as the state-of-the-art model-based RL algorithm TD-MPC2 (Hansen et al., 2024). Configurations for baselines are tuned to minimize training wall time with no regard to sample efficiency. Code for PPO and SAC is implemented based on CleanRL (Huang et al., 2022) and leverages the torch compile and cudagraphs acceleration features introduced by LeanRL.

Sample Efficient Reinforcement Learning: All of the RL baselines in the wall-time efficient setting besides PPO are included here with configurations tuned towards more gradient updates and fewer environment steps to maximize sample efficiency.

We share results on training speed in Section 4.2. More in-depth details on RL setups and more training results are shared in Appendix C.

Offline Imitation Learning: We currently provide Behavior Cloning, Diffusion Policy (Chi et al., 2023), Action Chunking Transformer (Zhao et al., 2024), and PerACT (Shridhar et al., 2022) as baselines. We also support evaluating (but not training) several vision-language action (VLA) models, namely Octo (Octo Model Team et al., 2024), RT-X (Collaboration et al., 2023), and RDT-1B (Liu et al., 2024). We leave to future work to support training VLA models on simulation data as RDT-1B has done with the previous ManiSkill2 datasets.

Online Imitation Learning: Online imitation learning generally refers to algorithms that learn from demonstrations in addition to collecting online environment transitions. We currently provide the two state-of-the-art baselines: Reinforcement Learning from Prior Data (RLPD) (Ball et al., 2023), and Reverse Forward Curriculum Learning (RFCL) (Tao et al., 2024).

Mandlekar et al. (2021) show that imitation learning algorithm performance heavily depends on how the demonstrations were collected, particularly on how "multimodal" the data is. For example, some behavior cloning algorithms perform poorly when trained on motion planning or human teleoper-

ated data, but perform well if trained on data generated by a neural network. With this caveat in mind, we explicitly track in all our LfD baselines the number of demonstrations used, what type of demonstrations are used (RL generated, motion planning, or human), and where the demonstrations are sourced from exactly (a longer description e.g. neural net trained via TD-MPC2, teleoperation via Meta-Quest VR). Imitation learning results on some environments are shared in Appendix D.

4.2 RL Training Speed

We run experiments using PPO (Schulman et al., 2017) on the ManiSkill3 GPU simulation and the ManiSkill2 CPU simulation. ManiSkill2 was previously the fastest robotics simulation+rendering framework until ManiSkill3. The experiments were run on an RTX-4090 GPU on the PickCube task, where a Franka robot arm must grasp a randomly initialized cube and hold it still at a random goal location. For the vision-based task no ground truth data like cube pose is provided and the RL policy must solve from proprioceptive information and one 128x128 RGB image rendered by the environment's 3rd-person camera. RL hyperparameters are tuned to achieve the fastest training time in both settings. Results in Figure 11 show that state and vision based training are massively accelerated with GPU simulation and rendering. PickCube with delta joint position control from state-based observations in GPU simulation reaches near 100% success rate after about 1 minute of training, a 15x speed up relative to ManiSkill2. From RGB observations with parallel rendering PickCube is solved after about 10 minutes of training, a 8x speed up relative to ManiSkill2. For results and details of RL on more environments see Appendix C.

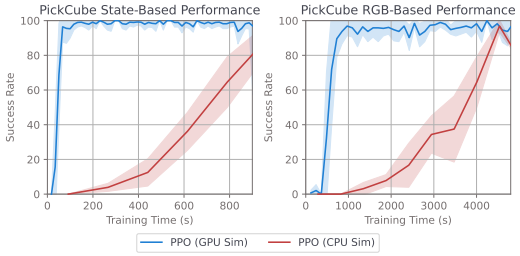


Figure 11: Wall-clock training time of PPO on GPU/CPU simulation showing the average success rate over time across 5 seeds. Shaded areas correspond to the 95% confidence interval.

4.3 Vision-Based Sim2Real Manipulation

To showcase the use-case of heterogeneous simulation and parallel rendering, we provide a simple reproducible setup for end-to-end training a vision-based manipulation policy that deploys successfully zero-shot. The setup uses the low-cost $300 Koch robot arm and a phone camera for third-person RGB observations, leveraging the open-source Hugging Face LeRobot library (Cadene et al., 2024) for the hardware and setup to enhance reproducibility and accessibility. This setup is not limited to the Koch arm; other more expensive robotic arms can be used and likely perform better due to more precise hardware.

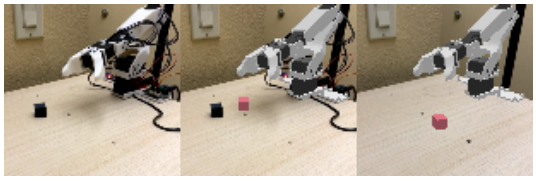


Figure 12: Green screening visualized with Koch-v1.1 robot. Left: Real RGB observations. Right: Sim RGB observations overlaid on real background. Middle: Sim RGB observations overlaid on real RGB observations.

During simulation training and real-world evaluation, observations are restricted to RGB inputs and robot joint positions; no demonstrations or privileged state information such as cube pose is used, and the robot is controlled via continuous joint level input. We note however that future work combining simulation training with teleoperated demonstrations could improve performance and training time. To match the real background within simulation and reduce the visual sim2real gap we green screen the real-world background. We skip green-screening over the dynamic objects like the cube and robot arm via the environment's segmentation map, see Figure 12 for an example. To enhance generalization and accommodate variations in real-world setups, we domain randomize a number of elements of the simulation. Task agnostic domain randomizations include camera pose, lighting direction, and the robot pose. Specific to the cube picking task we further domain randomize the cube size, color, and friction.

In this demonstration we picked a random table in a house and set up the green-screening and roughly aligned the simulation camera pose with the real-world camera pose. Then we train on the cube picking environment with PPO using a simple Nature CNN backbone for image feature

processing for 15 million samples using 256 parallel environments, taking approximately 1 hour on a RTX 4090 GPU. The trained policy is then zero-shot deployed on the real robot, using the same controller the policy was trained on in simulation, namely a delta target joint position controller. Evaluating the final checkpoint across 3 training runs in the real-world 8 times each yielded a $22/24 = 91.6\%$ success rate. Real world evaluations test on cubes of varying sizes, colors, and start poses, demonstrating our vision-based sim2real setup is capable of learning robust manipulation policies. Evaluation videos can be found in the supplemental and can be reproduced in new environments following a tutorial we will open source.

We further evaluated each intermediate checkpoint from the 3 RL training runs and plot the simulation and real-world success rates in Figure 13. The figure shows a good correlation between real-world and simulated success rates, indicating that some sim2real digital twins built with ManiSkill can be fairly accurate and reflect the real-world. Interestingly, we observe that despite not modelling shadows or the exposed wiring of the robot arm accurately, a successful sim2real policy is still capable of being trained. See Appendix B for more details on the exact setup, domain randomizations, controller implementation, and more.

5 LIMITATIONS

While parallel rendering can enable some new and more accessible approaches to sim2real and visual RL, it has its limits. For complex environments with many geometries (e.g. room-scale scenes) we can GPU parallel render and simulate them but at a lower number of environments on one GPU.

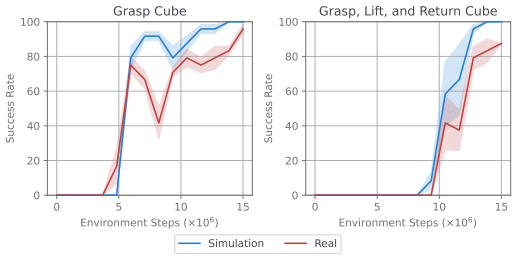


Figure 13: Pick-cube sim and real success rates on the grasp cube subtask as well as the full success consisting of grasping, lifting, and return the cube to a goal position. 14 training checkpoints for each of the 3 seeds are evaluated on both sim and real 8 times, using a variety of cube colors and sizes. Success rates are averaged across trials and shaded areas represent 95% confidence intervals.

While it is now possible to do some more generalizable zero-shot RGB based sim2real with a relatively simple setup, some reward engineering was required to encourage safer and more robust grasp behaviors of RL trained policy. Moreover, this is far from solving sim2real completely, but more of opening a new avenue of accessible approaches based on fast visual RL in simulation on one GPU. The sim2real demo is also limited to using just static cameras and it can be interesting future work to explore how to use methods like Gaussian Splatting for novel-view rendering to support mounted cameras. One could also explore algorithnmic changes and/or domain randomizations that can help remove the need for green screening.

Not all types of environments in ManiSkill are readily "GPU parallelized" in the sense that there are a batch of parallel environments. Most GPU parallelized environments are rigid-body based. The soft body environments are not batched as they use a significant portion of the GPU to simulate just a single environment fast. The vision-tactile simulator tuned for sim2real transfer uses a different set of algorithms for physics simulation compared to the majority of rigid-body only environments.

6 CONCLUSION

ManiSkill3 introduces a state-of-the-art framework/benchmark for generalizable robotics simulation and rendering. ManiSkill3 uses less GPU memory and depending on scenario can run faster while also supporting heterogeneous GPU simulation. Additionally we support the most diverse range of robotics tasks compared to alternative simulators. Importantly, we reliably support both sim2real and real2sim environments in manipulation tasks with real-world reproducible results. Moreover, we provide a immersive VR teleoperation system option for users to collect data for their own research. Furthermore, ManiSkill3 provides an easy-to-use object-oriented API for building all kinds of GPU simulated tasks, democratizing access to scalable robot learning. Finally, demonstrations and RL/IL baselines with clearly defined metrics are open sourced for users to use. We believe that our comprehensive approach to building the open-osurce ManiSkill3 will encourage the research community to tackle manipulation challenges more extensively through compute-scalable simulation.

REFERENCES

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL https://pytorch.org/assets/pytorch2-2.pdf.

Philip J. Ball, Laura M. Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 1577–1594. PMLR, 2023. URL https://proceedings.mlr.press/v202/ball23a.html.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong T. Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: robotics transformer for real-world control at scale. *CoRR*, abs/2212.06817, 2022. doi: 10.48550/ARXIV.2212.06817. URL https://doi.org/10.48550/arXiv.2212.06817.

Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. https://github.com/huggingface/lerobot, 2024.

Berk Çalli, Aaron Walsman, Arjun Singh, Siddhartha S. Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols. *CoRR*, abs/1502.03143, 2015. URL http://arxiv.org/abs/1502.03143.

Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio, 2015–2021.

Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.

Linghao Chen, Yuzhe Qin, Xiaowei Zhou, and Hao Su. Easyhec: Accurate and automatic hand-eye calibration via differentiable rendering and space exploration. *IEEE Robotics and Automation Letters*, 2023.

Weihang Chen, Jing Xu, Fanbo Xiang, Xiaodi Yuan, Hao Su, and Rui Chen. General-purpose sim2real protocol for learning contact-rich manipulation with marker-based visuotactile sensors. *IEEE Transactions on Robotics*, 40:1509–1526, 2024. doi: 10.1109/TRO.2024.3352969.

Yuanpei Chen, Yaodong Yang, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen Marcus McAleer, Hao Dong, and Song-Chun Zhu. Towards human-level bimanual dexterous manipulation with reinforcement learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=D29JbExncTP.

Xuxin Cheng, Jialong Li, Shiqi Yang, Ge Yang, and Xiaolong Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024a.

Xuxin Cheng, Jialong Li, Shiqi Yang, Ge Yang, and Xiaolong Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024b.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu (eds.), *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://doi.org/10.15607/RSS.2023.XIX.026.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Mart'in-Mart'in, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864, 2023.

Runyu Ding, Yuzhe Qin, Jiyue Zhu, Chengzhe Jia, Shiqi Yang, Ruihan Yang, Xiaojuan Qi, and Xiaolong Wang. Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning. *arXiv preprint arXiv:2407.03162*, 2024.

Yunhai Feng, Nicklas Hansen, Ziyan Xiong, Chandramouli Rajagopalan, and Xiaolong Wang. Finetuning offline world models in the real world. In *Proceedings of the 7th Conference on Robot Learning (CoRL)*, 2023.

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, 2021. URL `http://github.com/google/brax`.

Zipeng Fu, Qingqing Zhao, Qi Wu, Gordon Wetzstein, and Chelsea Finn. Humanplus: Humanoid shadowing and imitation from humans. *CoRR*, abs/2406.10454, 2024. doi: 10.48550/ARXIV.2406.10454. URL `https://doi.org/10.48550/arXiv.2406.10454`.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, and Yashraj S. Narang. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pp. 5977–5984. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10160216. URL `https://doi.org/10.1109/ICRA48891.2023.10160216`.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

Yunfan Jiang, Chen Wang, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Transic: Sim-to-real policy transfer by learning from online correction. In *Conference on Robot Learning*, 2024.

Mukul Khanna, Yongsen Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Shacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X. Chang, and Manolis Savva. Habitat Synthetic Scenes Dataset (HSSD-200): An Analysis of 3D Scene Scale and Realism Tradeoffs for Object-Goal Navigation. *arXiv preprint*, 2023.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin

Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1k: A benchmark for embodied AI with 1,000 everyday activities and realistic simulation. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=_8DoIe8G3t.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In Aleksandra Faust, David Hsu, and Gerhard Neumann (eds.), *Conference on Robot Learning, 8-11 November 2021, London, UK*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1678–1690. PMLR, 2021. URL https://proceedings.mlr.press/v164/mandlekar22a.html.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *7th Annual Conference on Robot Learning*, 2023.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object

understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Yuzhe Qin, Wei Yang, Binghao Huang, Karl Van Wyk, Hao Su, Xiaolong Wang, Yu-Wei Chao, and Dieter Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. *arXiv preprint arXiv:2307.04577*, 2023.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann (eds.), *Conference on Robot Learning, 8-11 November 2021, London, UK*, volume 164 of *Proceedings of Machine Learning Research*, pp. 91–100. PMLR, 2021. URL https://proceedings.mlr.press/v164/rudin22a.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Stone Tao, Arth Shukla, Tse-kai Chan, and Hao Su. Reverse forward curriculum learning for extreme sample and demo efficiency. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=w4rODxXsmM.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043, 2021.

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

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy P. Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Softw. Impacts*, 6:100022, 2020. doi: 10.1016/J.SIMPA.2020.100022. URL https://doi.org/10.1016/j.simpa.2020.100022.

Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

Tony Z. Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Seyed Kamyar Seyed Ghasemipour, Chelsea Finn, and Ayzaan Wahid. ALOHA unleashed: A simple recipe for robot dexterity. In *8th Annual Conference on Robot Learning*, 2024. URL `https://openreview.net/forum?id=gvdXE7ikHI`.

Sheng Zhong, Thomas Power, Ashwin Gupta, and Peter Mitrano. PyTorch Kinematics, February 2024.

Appendix Table of Contents

- Environments and Robots A
- Vision Based Sim2Real B
- Reinforcement Learning Baselines C
- Imitation Learning Baselines D
- Simulation and Rendering Benchmarking E
- VR Teleoperation F

A ENVIRONMENTS AND ROBOTS LIST

Figure 14: A sample of 16 of the robots supported in ManiSkill3 on both CPU and GPU simulation.

This section covers key implementation and optimization details of the general supported task categories. For videos of soft body and vision-tactile tasks we refer readers to the ManiSkill2 paper. A sample of 16 of the supported robots, which include mobile manipulators, floating grippers, quadrupeds, etc. is displayed in Figure 14. Videos of some tasks and how they are solved are attached in the supplemental.

We emphasize here that while ManiSkill3 may not have the most distinct number of tasks compared to some benchmarks, the core contribution is supporting a diverse array of possible tasks with open-sourced code that users can reference and use to easily build more tasks.

A.1 TABLE TOP MANIPULATION

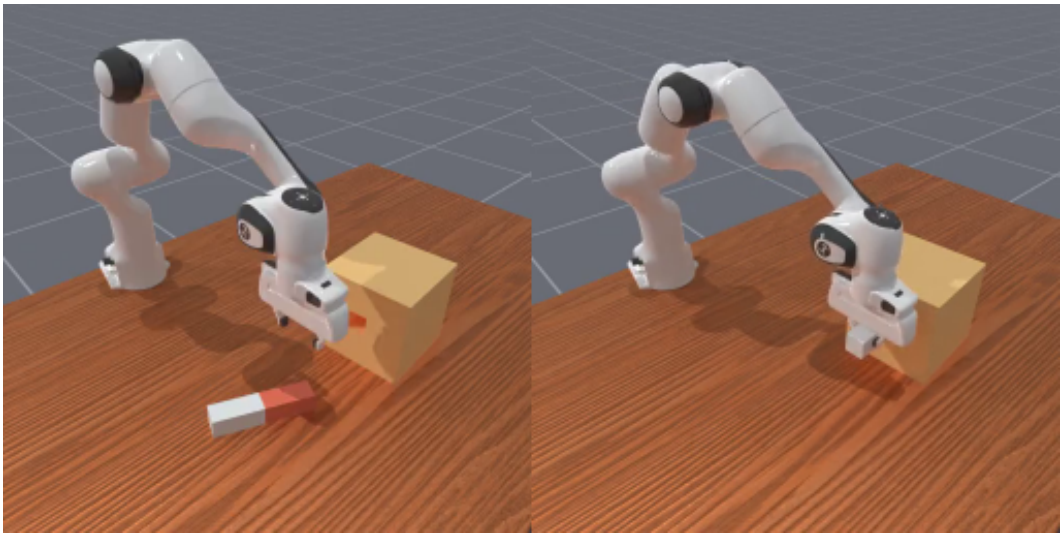


Figure 15: Example table-top manipulation task showing the start state and the solved state. The robot arm must grasp the white-orange peg and insert it into the box.

Table-top manipulation is primarily related to controlling one or more robot arms to manipulate an object on a table. Robots like Franka Emika Panda and Universal Robots 5 fall under this category. Typical tasks may include picking up objects, inserting a peg, assembling a structure, pushing objects, etc. An example task is shown in Figure 15.

Implementation Details: All robot arms are modified to have certain impossible self-collisions disabled and some have their collision meshes modified for faster simulation.

A.2 MOBILE MANIPULATION

Mobile manipulation here refers to tasks in which a robot arm has a mobile base. Robots like Fetch and Stretch fall into this category. Typical tasks may include placing objects on surfaces, opening cabinet doors/drawers, picking up objects off the ground, etc. Example task is shown in Figure 16.

Implementation Details: The default robot supported is Fetch. The mobile base in particular is not simulated by driving the wheels, and is modeled similarly to AI2-THOR (Kolve et al., 2017) and Habitat (Szot et al., 2021) with one joint controlling forward/backward movement and another controlling rotation of the base. The Fetch robot definition and collision meshes have further been tuned to be simpler for faster simulation. Several impossible self-collisions between some links have been explicitly ignored to speed up simulation.

A.3 ROOM SCALE ENVIRONMENTS

ManiSkill3 provides out-of-the-box code to build the ReplicaCAD environment from Habitat (Szot et al., 2021), all AI2-THOR environments (Kolve et al., 2017) using assets compiled by the authors of the Habitat Synthetic Scenes Dataset (Khanna et al., 2023), and the RoboCasa scenes dataset (Nasiriany et al., 2024). Photo-realism is also possible by turning on the ray-tracing shader options when creating the environment.

AI2-THOR (Kolve et al., 2017) and Habitat (Szot et al., 2021) have long-horizon mobile manipulation tasks but rely on slow CPU simulation / slow rendering systems, and do not support contact-rich

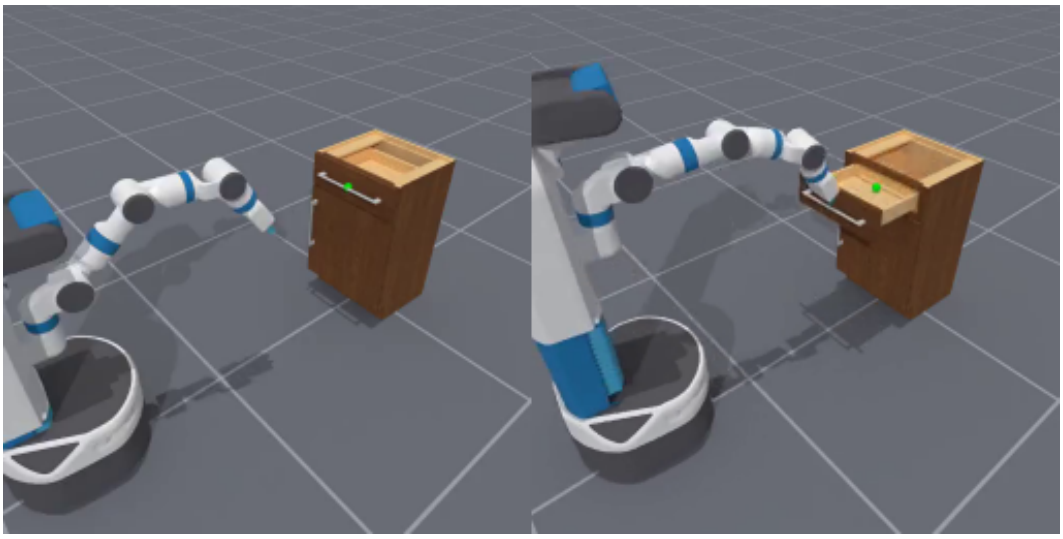


Figure 16: Example mobile manipulation task showing the start state and the solved state. The mobile robot arm must grasp the handle indicated by the green sphere and pull it open.

physics for manipulation (only magical grasp) and do not look photorealistic. Robocasa (Nasiriany et al., 2024) has contact-rich long-horizon mobile manipulation tasks in photorealistic room-scale environments. However, Robocasa simulates and renders these scenes at around 25FPS as it does not use GPU parallelized simulation and rendering. In contrast ManiSkill3 can simulate the complex ReplicaCAD environment up towards 2000+ FPS with rendering.

Implementation Details: We further make several modifications to ReplicaCAD to make it completely interactive as some of the collision meshes for articulations were modelled incorrectly and thus did not support low-level grasping. Via CoACD (Wei et al., 2022) we run convex decomposition on objects in AI2-THOR scenes to generate simulatable non-convex collision meshes so those objects can e.g. be grasped and moved around correctly. Via manual annotation by ManiSkill3 authors, certain categories of objects in AI2-THOR are made to be kinematic so they cannot be moved around (e.g. tables, TVs, clocks, paintings) with the rest allowed to be dynamic to be fully simulated (e.g. apples, baseball bat, cups) to optimize simulation speed.

A.4 Locomotion

Locomotion here refers to controlling robot joints to move a robot from one location to another. Quadrupeds such as AnyMAL-C and humanoids such as Unitree-H1 fall into this category. Example task is shown in Figure 17.

Implementation Details: Similar to Isaac Lab, quadruped robots in locomotion tasks are modified such that the large majority of collision meshes are removed, leaving behind just the feet, ankles, and the body visualized in Figure 18. Moreover following Isaac Lab, joint limits are significantly constrained such that random actions do not easily cause the robot to fall over.

A.5 Humanoid / Bi-manual Manipulation

Tasks here refer to the use of a humanoid embodiment such as the Unitree H1 robot or bi-manual robot embodiments for manipulation tasks. Example task shown in 19.

This type of task can be found in benchmarks like RoboCasa and BiGym, although we note that RoboCasa and BiGym do not reliably support working RL setups and are not GPU parallelized. Isaac Lab currently does not have these types of tasks out of the box.

Implementation Details: For some tasks where the robot has legs, to simplify the task the legs are fixed in place so that robot learning methods can focus on manipulation only and train faster. Tasks

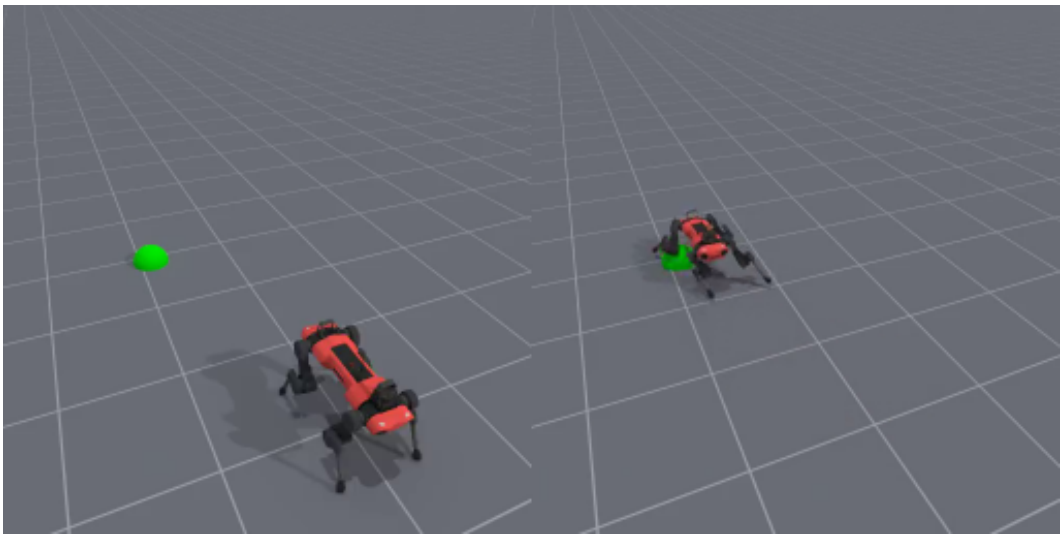


Figure 17: Example humanoid manipulation task showing the start state and the solved state. The humanoid must use both arms to grasp the box and transport it to the other table.

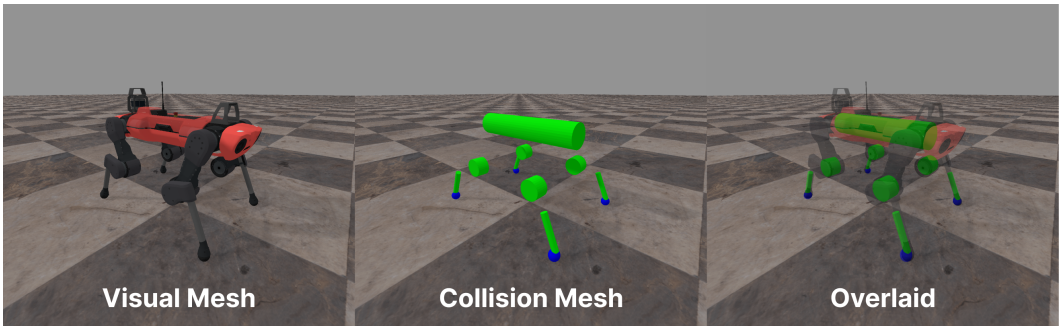


Figure 18: Comparison of the visual and collision mesh of one of the robot quadruped models, AnyMAL-C.

still have the option to swap a version of the robot where all joints are controlled although they are much harder.

A.6 Multi-Agent Robots

Multi-Agent robots refer to support for controlling multiple different robots in the same simulation to perform a task. Setups such as multiple quadrupeds or robot arms fall into this category. A common task is the handover of objects. Example task shown in Figure 20.

This type of task can be found in Robosuite. Isaac Lab supports this type of task, but not out of the box. Past environments based on older versions of Isaac have example tasks with multiple robot arms/hands running on GPU simulation (Chen et al., 2022).

Implementation Details: By default the action space is a dictionary action space in multi-agent environments with a dictionary key for each controllable agent, which follows the standard PettingZoo API Terry et al. (2021). PettingZoo is currently the most popular interface for multi-agent RL environments. For users who do not wish to do multi-agent RL they can flatten the action space into a single vector if necessary via a environment wrapper provided by ManiSkill3.

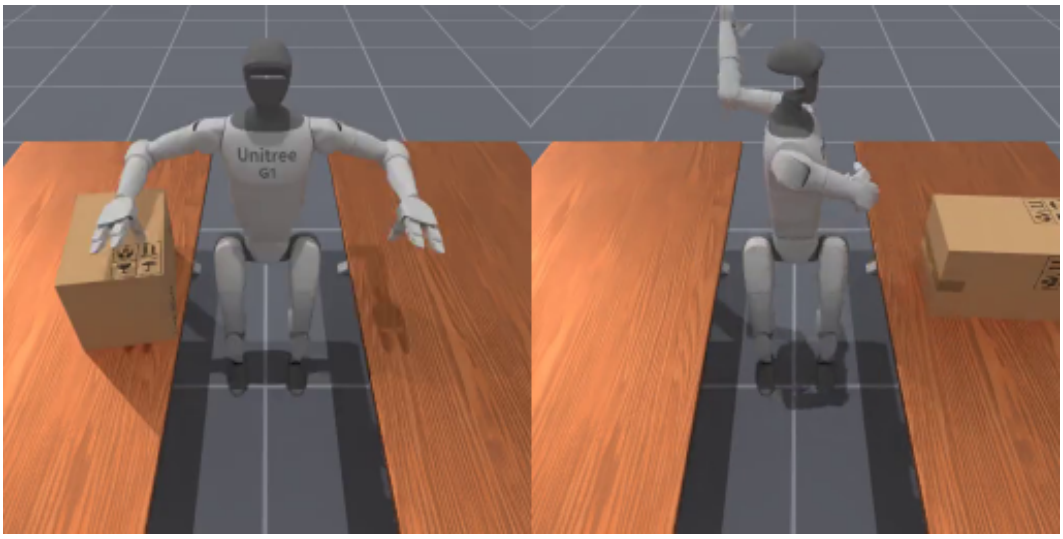


Figure 19: Example humanoid manipulation task showing the start state and the solved state. The humanoid must use both arms to grasp the box and transport it to the other table.

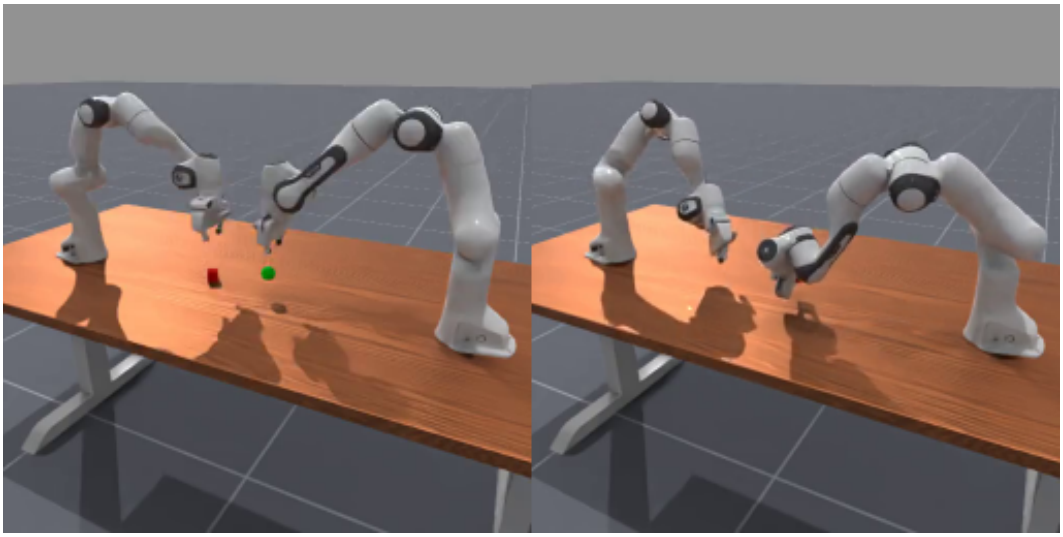


Figure 20: Example multi-agent robotics task showing the start state and solved state. The left robot must push a cube over to the other robot to pick up.

A.7 DRAWING/CLEANING

Drawing/Cleaning refers to tasks for dynamically "adding/removing" objects to simulate the effect of drawing or cleaning. A task could be to draw the outline of a shape on a canvas or clean dirty spots on a table surface. Example task shown in Figure 21.

ManiSkill3 is the only framework that supports this kind of task out of hte box with GPU parallelization and rendering.

Implementation Details: The drawing/cleaning effect is achieved by building ahead of time 1000s of small thin cylinders that represent "ink" or dirty spots. For a drawing task, all of these cylinders are hidden away from the camera view. When a robot moves a drawing tool close to a surface/canvas, the cylinders have their pose set to be on top of the surface right under where the drawing tool is. For a cleaning task, all the cylinders/dirty spots are visible and removed once the cleaning tool moves over the dirty spot. Currently the drawing/cleaning environments in ManiSkill3 do not require

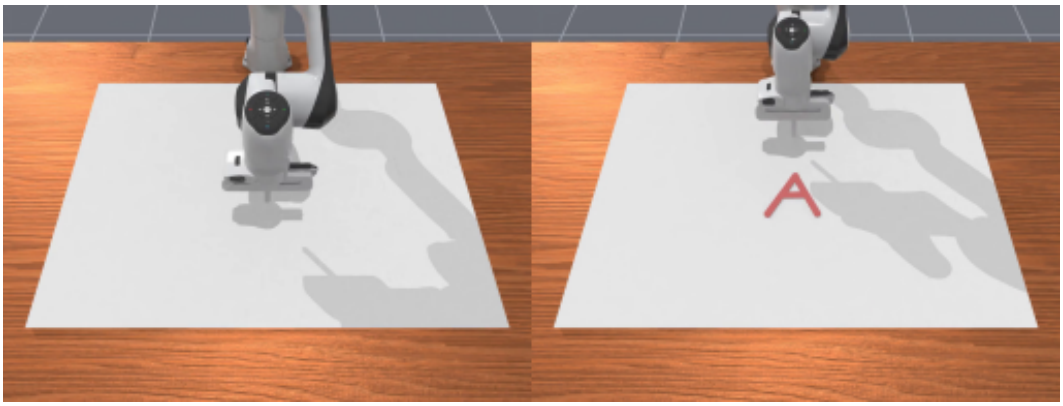


Figure 21: Example drawing/cleaning task showing the start state and solved state. The robot must draw a given SVG file (which describes a set of lines), in this case the letter A.

intricate grasping (nor is it the focus) of the drawing/cleaning tool, so the solver position/velocity iteration values are tuned down.

A.8 DEXTROUS MANIPULATION

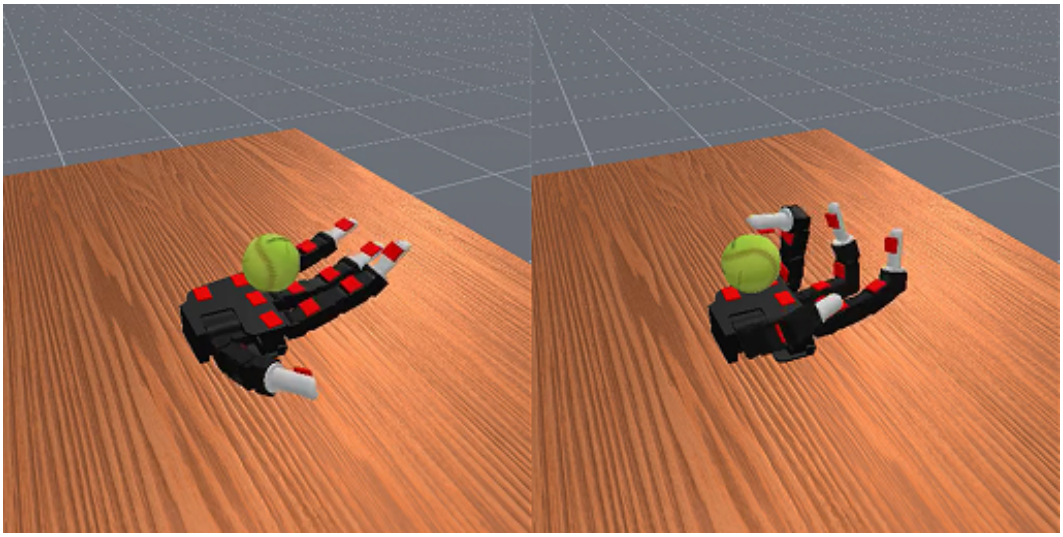


Figure 22: Example dextrous manipulation robotics task showing the start state and solved state. The robot hand must rotate the object in place to a desired orientation.

Dextrous Manipulation refers to tasks often involving multi-fingered hands and dense/rich contacts occurring during manipulation. An example task is in-hand rotation. Example task shown in Figure 22.

This type of task has been heavily explored and exists in Isaac Lab and ManiSkill3 with GPU parallelization.

Implementation Details: Not many special optimizations are made here. Tactile sensing is further provided in environments via touch sensors placed on the dextrous hand at various points.

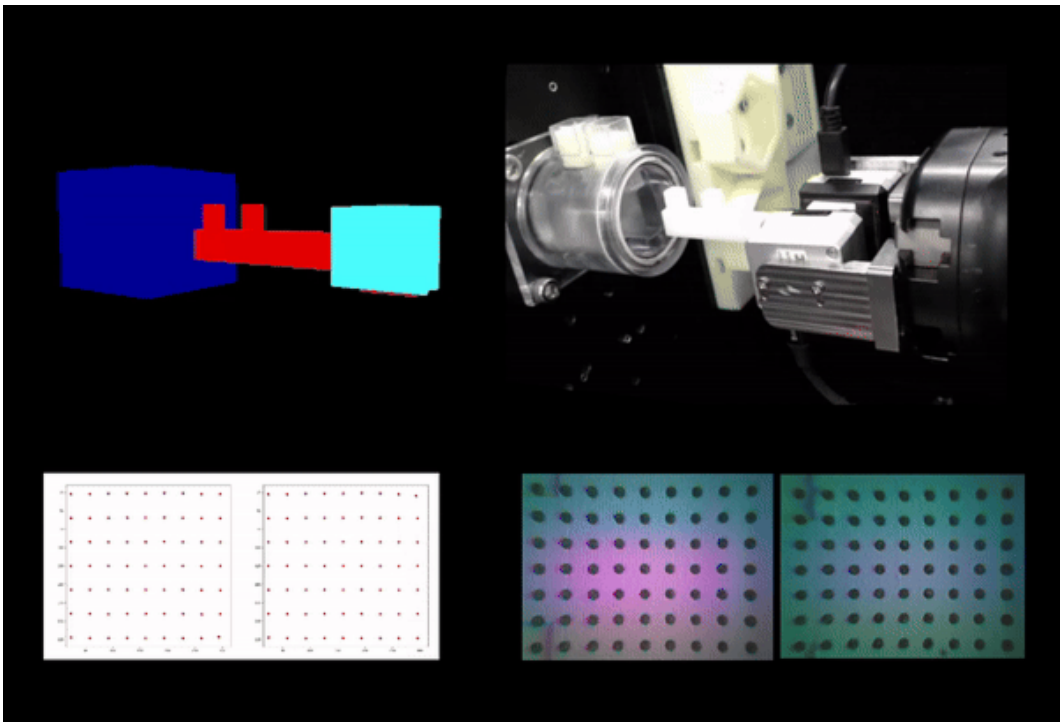


Figure 23: Example task of a key insertion task in simulation (top left), the real world equivalent (top right), and plots of the tactile feedback data (bottom).

A.9 Vision Tactile Manipulation

Vision Tactile Manipulation refers to tasks that rely on processing tactile information like images to solve manipulation tasks. Tasks could include key insertion which require tactile inputs to solve due to visual occlusions. Example task shown in Figure 23

Implementation Details: The vision-tactile, sim2real, manipulation environments are ported over from the 2024 ManiSkill Vision-based-Tactile Manipulation Skill Learning Challenge (Chen et al., 2024).

A.10 Classic Control

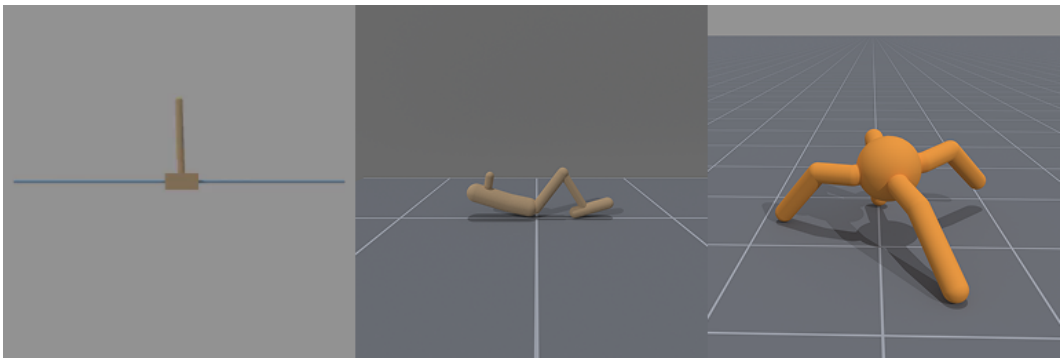


Figure 24: Example control environments: CartPoleBalance, Hopper, and Ant.

Classic control is quite broad but in this context refers to fake robots tasked with achieving some stable pose or moving in a direction at desired speeds. Tasks include cartpole balancing, hopper etc. Examples are shown in Figure 24

DM-Control (Tunyasuvunakool et al., 2020) has the most implemented control tasks and Isaac Lab has a few GPU parallelized variants. ManiSkill3 has GPU parallelized simulation+rendering variants of the most control tasks.

Implementation Details: ManiSkill3 uses its MJCF loader to load the MJCF robot definitions from the original DM-Control repository and tunes robot pd joint delta position controllers to align as closely as possible to the behavior seen in DM-Control/Mujoco.

A.11 Digital Twins

Digital twins have two variants included, environments for real2sim and environments for sim2real. The distinction here is real2sim environments simply need to be designed so that a model trained on a real world equivalent of the simulation environment achieves similar success rates when evaluated in simulation. Sim2real digital twins are environments designed so that models trained on simulation data can be more easily used for real world deployment.

For real2sim, ManiSkill3 ports over and GPU parallelizes some environments from SIMPLER (Li et al., 2024), which enables efficient evaluation of policies trained on real world data like the generalist RT-X and Octo models. The primary tricks include green-screening a real world image and texture matching which have been copied over and parallelized. We ensured the GPU parallelized port of SIMPLER achieves similar results/behaviors as the original CPU simulated environments as shown in Figure 25.

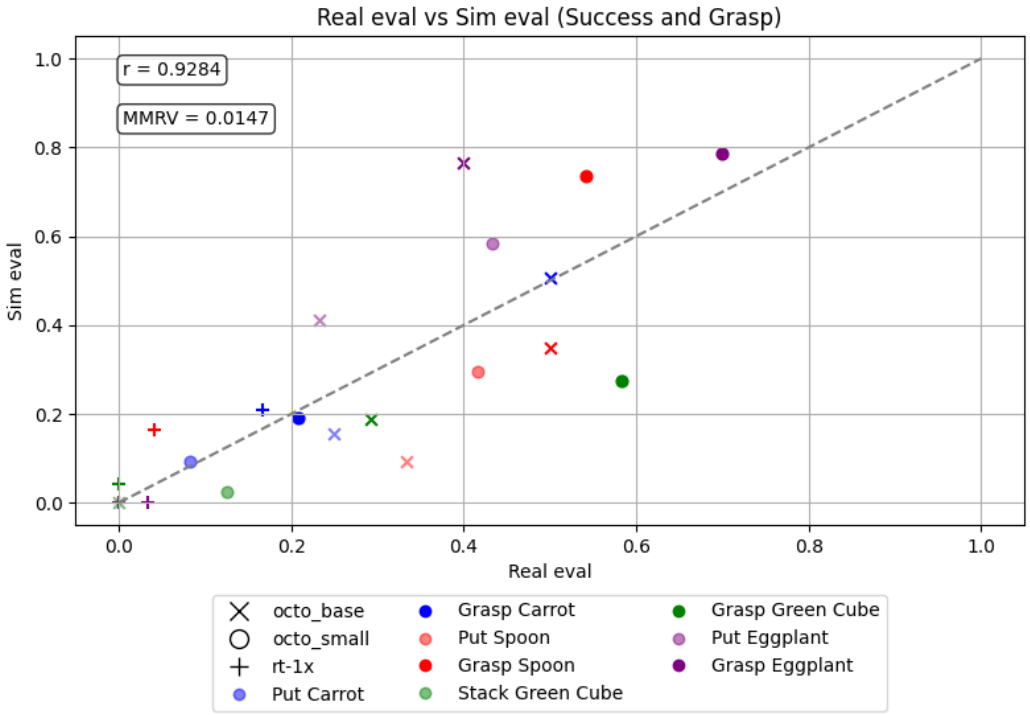


Figure 25: Evaluated success rates of generalist robotics models like Octo and RT-1X on 4 different tasks. The correlation and MMRV metrics are close to that of the original paper. MMRV is Mean Maximum Rank Violation (lower the better), which assess the accuracy of the consistency of the rankings of real and sim policy evaluations.

For sim2real, we provide tools useful for training policies in simulation and directly deploying into the real-world. Given the number of details involved in sim2real, we explain the implementation details of vision-based sim2real in detail in Appendix B.

A.12 SOFT BODY MANIPULATION ENVIRONMENTS

Soft body manipulation refers to the manipulation of soft body objects that can deform and morph in shape. Tasks include excavating sand particles, pouring water etc.

ManiSkill3 soft body manipulation environments are the same as ManiSkill2 which uses 2-way coupled rigid-MPM simulation that enables rigid body objects to interact with soft body objects. We point readers to the ManiSkill2 paper for more details on soft body simulation.

B VISION-BASED SIM2REAL

We describe how one can use ManiSkill3 to perform end-to-end vision-based sim2real by training with PPO a RGB-based manipulation policy in simulation and zero-shot deploying it in the real world. Past work such as TRANSIC-Envs (Jiang et al., 2024) provide reproducible sim2real digital twin setups but rely on state estimation via ArUco markers for sim2real transfer and does not support visual feedback. Dextreme (Handa et al., 2023) provides a realistic in hand cube rotation environment but also does not support efficient visual feedback and relies on accurate state estimation. ManiSkill3 on the other hand has fast visual data generation that enable mimicing real-world cameras inputs for training at scale and fast. We note however the example setup here is a task less complex than that of e.g. Dextreme but point out that our reproducible setup is easily extendable and could be the start of opening a new avenue of sim2real research with efficient and fast large-scale simulation training on image inputs.

B.1 HARDWARE

We use the low-cost $300 Koch v1.1. robot arm and the LeRobot library for a simple accessible python interface to control the robot arm. Due to the simple but accessible hardware, it is generally harder to deploy policies on this robot as the motors are less precise compared to more expensive robot arms. Thus the vision-based sim2real demonstrated here can easily work on other robot arms. Apart from the robot we also setup one third view iPhone camera (any LeRobot compatible camera can work) to record RGB observation data.

B.2 CONTROLLER / ACTION SPACE

We use a target delta joint position controller in both simulation and the real-world at a control frequency of 30Hz. At timestep $t = 0$ after an (real or simulation) environment reset we save an initial target joint position value $\bar{q}_t = q_t$ where q_t is the joint position of the robot arm at timestep t. Given an action a_t at timestep t, we set $\bar{q}_{t+1} = \bar{q}_t + a_t$.

At each timestep t, we drive the robot's joint positions to $\bar{q}_t$. In simulation this is equivalent to setting target joint position values in the PhysX physics engine and the behavior is defined by a PD controller. In the real world this is equivalent to setting the next joint position of the robot arm to be $\bar{q}_t$, and most robot arms have APIs that enable this functionality and will internally run their own control loop to reach the set joint position.

The actual action space of the environment is normalized to a range of -1 to 1 for each joint. Given a predicted action from a policy $\hat{a}_t$ we clip it first to the range of $[-1, 1]$ and then unnormalize it to obtain $a_t = \frac{\hat{a}_t(h-l)}{2}$, which is the delta action applied to the $\bar{q}_t$. For the Koch robot arm $l = -0.05, h = 0.05$, which represent a maximum change of 0.05 radians in either direction of each joint.

We choose the target delta joint position controller as the sim2real dynamics gap for quasi-static tasks (tasks where objects only move when the robot manipulate them) is minimized. In both simulation and real-world in quasi-static settings with small enough action sizes as long as within the timeframe of one control step the real world robot and simulated robot reach $\bar{q}_t$ at timestep t then the simulation and real-world are aligned well. We note that we leave it to future work to explore controller and sim2real design for addressing more dynamic tasks such as catching a thrown object which require higher control frequencies.

B.3 OBSERVATION SPACE

The observation space follows that of all visual RL baselines, which contains both proprioceptive data and the camera input RGB data. For the simulated tasks the priopriocepive data contains both the current robot joint positions q_t as well as the target joint positions $\bar{q}_t$ which is required to make the problem markov and more easily solvable via RL. The RGB data is a 128×128 RGB image captured during each control step of the environment.

We further include a binary label of whether or not the robot is grasping something, which is a simple check of whether the joint position of the gripper joint is within 0.02 radians of the target

joint position of the gripper joint. If the joint position of a robot cannot reach the target joint position this indicates something is blocking that joint, e.g. an object being grasped. Empirically we find that due to occlusions that occur once the cube is grasped due to the camera angle and the gripper link covering the cube, the trained policy has difficulty discerning whether or not it has grasped a cube. The binary is grasped feature generated from $q_t, \bar{q}_t$ values is sufficient to overcome the issue of occlusions.

All of these observation data points are easily obtained in the real world without needing to set up perception stacks to observe e.g. object poses. No observation delays are added but can be useful for higher frequency tasks.

B.4 Environment Setup and Domain Randomizations

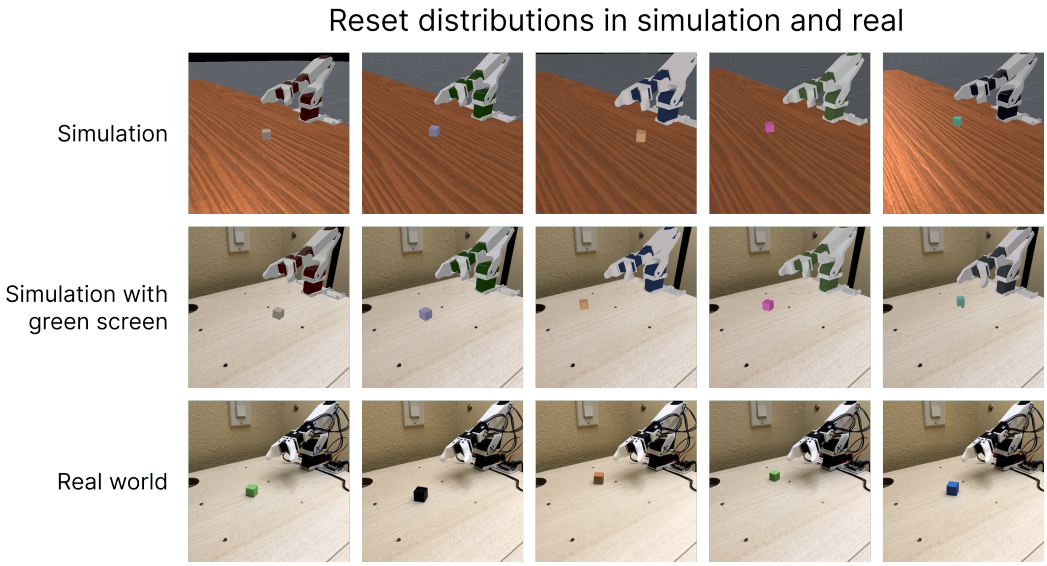


Figure 26: Sample images of the start states of the simulated environment and real-world environments, showing the reset distribution. Videos of the reset distribution can be found on our website.

The simulated environment contains 4 objects, the floor, the table, the cube, and the robot arm. A visualization of the reset distribution can be found in Figure 26. We perform the following randomizations at each timestep of the environment:

- Camera pose is randomized by randomly shifting the camera position up to 2.5 cm in x, y, and z axes in the world frame. This randomization scale sufficiently covers possible misalignments between simulated and real-world cameras.

We perform the following randomizations each time the environment is reset:

- The initial robot joint positions q_0 are sampled from $\mathcal{N}(q_R, 0.02)$, where q_R represents the joint positions we set for the real Koch v1.1 robot at each reset.
- The position of the cube is uniformly sampled within a 10cm^2 square in front of the robot arm, and the cube's z-axis rotation is randomized completely.

We perform the following randomizations each time the environment is reconfigured, which permit the environment to use different object assets, materials, camera settings. This occurs only during environment resets, but typically is done less often as it slows down reset times.

- Cube side lengths are sampled uniformally in the range $[1.5, 2.25]$ cm
- Cube frictions are sampled from $\mathcal{N}(0.3, 0.025)$
- Cube color is sampled uniformally from all possible RGB values.

- Robot chassis color is sampled from $\mathcal{N}(c, 0.05)$, where c is the base robot chassis RGB color scaled from 0 to 1.
- Robot motor color is sampled from $\mathcal{N}(m, 0.05)$, where m is the base robot motor color scaled from 0 to 1.

B.5 REWARD

$\mathbb{I}_{grasped}$ indicates whether the robot is grasping the cube. $\mathbb{I}_{touch_table}$ indicates if the robot is robot's gripper is colliding with the table. $\mathbb{I}_{tcp_close}$ indicates whether the tool center point, the point between the robot's two gripper fingers x_t^{tcp}, is within a half of the cube's side length from the cube's position, x_t^{cube}. We also define the vector from the first gripper finger to the second as $\vec{g}$ and the cube size as c. The reward function, for a state s and action a is:

$$\begin{aligned} r_t(s,a) = & \, r_{dist}(x_t^{tcp}, x_t^{cube}, 15) + r_{orient}(\vec{g}) \\ & + 2\mathbb{I}_{tcp_close} r_{dist}(||\vec{g}||, c, 40) \\ & + \mathbb{I}_{grasped}\left(1 + 3r_{dist}(q_t, q_R, \frac{4}{\pi})\right) \\ & - 2\mathbb{I}_{touch_table} \end{aligned}$$

r_{dist} is a common distance reward function used in ManiSkill3, where $r_{dist}(x,y,z) = 1 - tanh(z * ||x-y||)$. $r_{orient}(\vec{g}) = 1 - (\vec{u} \cdot \frac{\vec{g}}{||\vec{g}||})$, where $\vec{u}$ is the up vector. Intuitively, this means rewarding the agent with a value of one for keeping $\vec{g}$ perpendicular to the table normal and a value of zero if it is parallel instead. q_t and q_R represent the joint positions at timestep t and upon environment reset, respectively.

We find that action magnitude, $||a||$, does not need to be considered within the reward, as the controller's restrictions of l and h are appropriately tuned for safe deployment and for the realistic maximum joint acceleration magnitudes for each real robots' motors.

B.6 TRAINING

PPO with RGB inputs is used for training, details on the overall RL implementation is discussed in Appendix C. Specific to this task we use a reconfiguration frequency of 10 to ensure the RL policy sees a sufficiently diverse set of randomizations over object geometry and materials while also ensuring we are not performing slow reconfiguration resets too often. Ultimately training takes about 1 hour on a 4090 GPU. The final training curves can be found in the main paper in Figure 13.

C REINFORCEMENT LEARNING BASELINES

We most comprehensively test PPO and TD-MPC2 as reinforcement learning baselines, which represent two ongoing sides of RL research, walltime efficient RL and sample efficient RL. SAC is also provided but is not heavily benchmarked as of now. When possible RL baselines have sample-efficient and wall-time efficient hyperparameters. These baselines are implemented with PyTorch (Ansel et al., 2024).

C.1 PROXIMAL POLICY OPTIMIZATION (PPO)

Neural Network Architecture: For state based tasks the actor and critic networks are separated and are comprised of a MLP with three 256 unit hidden layers followed by Tanh activation. For vision based tasks the actor and critic share a feature processing backbone, namely a NatureCNN following the default setup that the popular Stable Baselines 3 RL library (Raffin et al., 2021) uses.

Training Details and Hyperparameters: The key hyperparameters that were tuned per environment were largely the discount factor, GAE lambda factor, the max episode steps/horizon, as well as the number of parallel environments. Typically we try to use up to 4096 parallel environments when possible for state based training and about 256 to 1024 environments for RGB based training. Discount factor by default is kept at a low value of 0.8 which empirically works well for most tasks and increased up to 0.99 if 0.8 does not work. Episode horizon is tuned such that the task can be solved within that horizon and the horizon is not much longer than the optimal solve time. We further tune the number of steps sampled from each parallel environment in each rollout. Generally a value of 16 or 32 is used and is tuned down if possible. However, we observe most tasks cannot be solved with low number of steps sampled per parallel environment like PegInsertionSide, likely due to long horizon temporal dependencies that are harder to model.

For all experiments we always turn partial resets on, which enables environments to reset early upon reaching a fail or success state if there are any. Partial resets generally enable faster training by reducing the time spent sampling the environment in irrelevant states.

ManiSkill environments support setting environment reconfiguration freqencies, which control how often during an environment reset we will also permit resetting/randomizing simulation properties that are not changeable after starting simulation (such as the object shape as done in the PegInsertionSide task). By default all experiments have a reconfiguration frequency of 0 for training environments as generally with sufficiently high number of parallel environments there is enough diversity in the training data. For more advanced use-cases like training robust sim2real policies as done in Appendix B, we use a non-zero reconfiguration frequency to better randomize the object geometries and other properties.

Due to the number of hyperparameters and environments we do not include them in the paper itself. All training runs are included on a Weights and Biases dashboard, the link of which will be shared after de-anonymization.

C.2 TEMPORAL DIFFERENCE LEARNING FOR MODEL PREDICTIVE CONTROL 2 (TD-MPC2)

Neural Network Architecture: For tasks with state-based observations, we used the original TD-MPC2 architecture from the original paper. For vision-based tasks, we modified the encoder to handle larger 128×128 input images and additional external state information (e.g., joint position, goal position). In particular, we introduced an additional convolutional layer to the RGB encoder, as well as a separate MLP (composed of one 256-unit and one 512-unit layer-normed, linear layer) to process the extra state data. The outputs from the RGB encoder and state encoder are then combined via a 512-unit linear layer. All other network components (e.g., actor and critic) remain unchanged from the original implementation.

Training Details and Hyperparameters: We evaluate TD-MPC2 with the wall-time-efficient mode (using 32 parallel GPU environments). Due to the number of hyperparameters and environments we do not include them in the paper itself. All training runs are included on a Weights and Biases dashboard, the link of which will be shared after de-anonymization.

We only tune two parameters, the replay buffer size and the environment control mode. For all vision-based tasks, we use a smaller replay buffer size of 200K, while for state-based tasks we use 1M. Generally, we employ the delta joint position control mode for wall-time-efficient runs as the environment runs faster relative to IK-based end-effector control; However, for tasks like PickCube-v1, in which the robotic arm cannot easily remain static at the goal, we use the delta end-effector position control mode to achieve better performance. TD-MPC2 baseline also has both a sample-efficient set of environment configurations. The sample-efficient setting uses 1 CPU environment whereas the wall-time efficient setting uses 32 GPU parallelized environments. Less parallel environments relative to the number of updates per frame of data sampled generally lead to better sample efficiency.

D IMITATION LEARNING BASELINES

D.1 BEHAVIOR CLONING BASELINES

We evaluate three offline imitation learning baselines on PickCube, PushCube, StackCube, and PegInsertionSide using state and RGB observations. The baselines are Behavior Cloning (BC), Diffusion Policy (DP), and Action Chunking Transformer (ACT). Motion planning is used for the data sources. Table 2 and 3 show the best success rates obtained during training on different tasks for state and RGB observations, respectively. Overall we find that Diffusion Policy performs the best, especially when there are few demonstrations.

	PickCube-v1		**PushCube-v1**		**StackCube-v1**		**PegInsertionSide-v1**	
	Demos	Success	Demos	Success	Demos	Success	Demos	Success
BC	1000	0.03	1000	0.81	1000	0.00	1000	0.00
ACT	1000	1.00	1000	0.97	1000	0.97	1000	0.43
DP	1000	1.00	1000	0.96	1000	0.99	1000	0.66
BC	100	0.00	100	0.69	100	0.00	100	0.00
ACT	100	0.70	100	0.99	100	0.50	100	0.14
DP	100	1.00	100	0.95	100	0.90	100	0.38

Table 2: Comparison of imitation learning baselines on different tasks using state observations with varying number of demonstrations.

	PickCube-v1		**PushCube-v1**		**StackCube-v1**		**PegInsertionSide-v1**	
	Demos	Success	Demos	Success	Demos	Success	Demos	Success
BC	1000	0.03	1000	0.81	1000	0.00	1000	0.00
ACT	1000	0.98	1000	0.89	1000	0.80	1000	0.00
DP	1000	1.00	1000	0.86	1000	0.81	1000	0.00
BC	100	0.00	100	0.00	100	0.00	100	0.00
ACT	100	0.28	100	0.30	100	0.33	100	0.00
DP	100	0.76	100	0.41	100	0.61	100	0.00

Table 3: Comparison of imitation learning baselines on different tasks using RGB observations with varying number of demonstrations.

D.2 VISION-LANGUAGE ACTION (VLA) MODEL BASELINES

We support training and/or evaluation of various vision-language action (VLA) model baselines. For evaluation, we support Octo and RT-x via the GPU parallelized version of the SIMPLER (Li et al., 2024) project. For RDT-1B, we support fine-tuning RDT-1B on a few ManiSkill3 demonstrations to then be evaluated on various tasks.

For both training and evaluation, a more classical sense-plan-act style method PerAct (Shridhar et al., 2022) is provided as a baseline in ManiSkill3. Different from RGB-based baselines like Octo and RT-x, PerAct operates on voxelized inputs. Our implementation follows the original PerAct design with modifications to demonstration generation and action execution: We obtain demonstrations by replaying trajectories with point cloud observations, which are voxelized directly during inference. This approach eliminates the need to explicitly convert RGBD observations to point clouds, as required in the original implementation. We hard-coded text descriptions for each task based on ManiSkill3 task cards. During inference, the end-effector pose-based actions are converted into

joint position sequences using MPlib, which are then executed under ManiSkill3's joint-position (pd_joint_pos) control mode.

We trained PerAct on PushCube-v1 and StackCube-v1 using 50 demonstration trajectories and evaluated over 100 episodes. The results are presented in Table 4 and Figure 27.

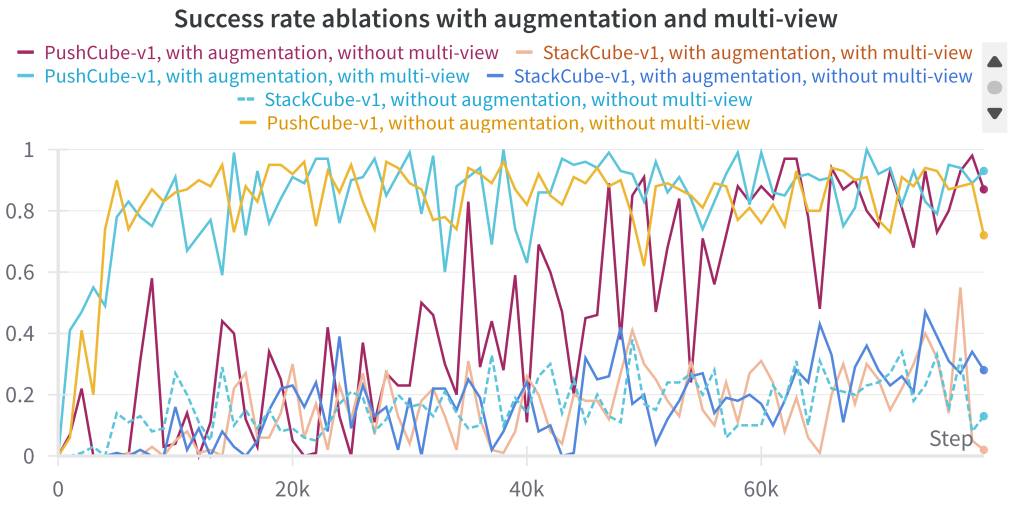


Figure 27: Success rate curves of PerAct over 80k training steps on PushCube-v1 and StackCube-v1

D.2.1 Multi-View and SE(3) Augmentation

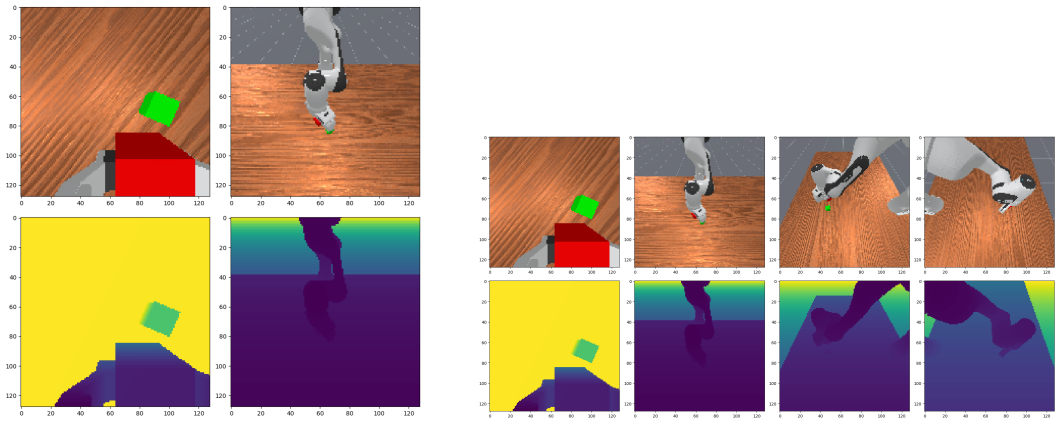
We also reported ablation results under **multi-view** and **SE(3) augmentation** settings: The **multi-view settings** include 4 cameras: a base camera, a left shoulder camera on the left of the arm, a right shoulder camera on the right, and a wrist camera mounted on the arm. Figures 28 and 29 illustrate that multi-view cameras provide finer voxelization details. Yet the difference for StackCube-v1 is marginal since the cubes and their surroundings are already visible via the hand and base cameras. Table 5 details the cameras used for each experiment.

(a) Default-view voxelized scene.

(b) Multi-view voxelized scene.

Figure 28: Comparison between default-view and multi-view voxelized images in StackCube-v1. The red dot indicates the predicted action.

The **SE(3) augmentation** follows the PerAct approach, perturbing point clouds and actions with random translations ([$\pm$0.025m $\pm$0.025m $\pm$0.025m]) and rotations ([$\pm 0°$ $\pm 0°$ $\pm 5°$]).



(a) Default-view RGBD observations. Cameras from left to right: hand_camera, base_camera

(b) Multi-view RGBD observations. From left to right: hand_camera, base_camera, left_shoulder_camera, right_shoulder_camera

Figure 29: Comparison between default-view and multi-view RGBD observations in StackCube-v1. Each column shows one camera's observations.

Setting	**PushCube-v1**	**StackCube-v1**
No aug., no m.v.	0.96	0.38
aug., no m.v.	0.98	0.47
aug., m.v.	1.00	0.55

Table 4: Best success rates of PerAct over 80k training steps for different tasks and settings. aug.: demo augmentation. m.v.: multi-view

Task	**Default Cameras**	**Multi-View Cameras**
PushCube-v1	base	base, l_shldr, r_shldr, wrist
StackCube-v1	base, wrist	base, l_shldr, r_shldr, wrist

Table 5: Camera configurations for each task and setting. l_shldr = left_shoulder, r_shldr = right_shoulder.

To summarize the observations and results: (1) SE(3) augmentation helps improve the success rate, though making convergence slower. (2) Multi-view observations can speed up convergence and enhance success rates by providing a more comprehensive 3D scene understanding. (3) As the number of cameras increases, the point cloud size grows. However, voxelized inputs help maintain consistent GPU memory consumption despite the increased data size.

E SIMULATION AND RENDERING BENCHMARKING

We carefully analyze the performance of ManiSkill3's parallel simulation and simulation+rendering performance compared to Isaac Lab. Currently we only have accurate results for a simple cartpole environment. While one can try and compare more complex environments it is difficult to align things perfectly. We compare against the Isaac Lab v1.2.0 which was first version with parallel rendering support that was released a few months after we initially released our framework.

E.1 SETUP

To keep things as fair as possible as both Isaac Lab and ManiSkill3 use PhysX, we ensure the following simulation configurations are the same:

- Simulation frequency: 120
- Control frequency: 60
- Solver Position Iterations: 4
- Solver Velocity Iterations: 0

We further try to ensure the objects in the scene are as similar as possible. Finally, reward/termination computations are explicitly removed to not factor in benchmark timings. Tests were conducted on a RTX 4090 GPU.

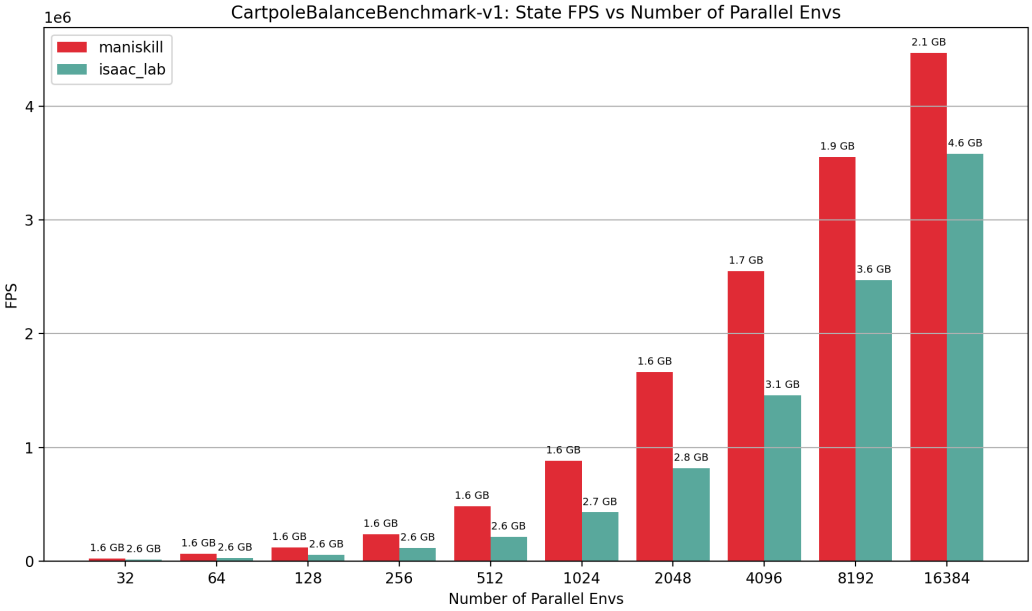


Figure 30: Simulation FPS against number of parallel environments without rendering. Annotated numbers on top of data points indicate the GPU memory usage. ManiSkill speed for this task is on par with Isaac Lab.

E.2 SIMULATION ONLY BENCHMARK RESULTS

Testing on the cart pole task, we see that ManiSkill3 runs a little faster and uses about 1.5x to 2x less GPU memory in Figure 30. We note that while ManiSkill3 appears faster here, one could make many simulation specific optimizations (simplified collisions meshes, tuned physics solver configurations etc.) to increase simulation speed and trade off simulation accuracy. We further note it is difficult to make truly apples-to-apples comparisons between simulators and that this is simply just one data point and may not extend to other environments. We specifically create documentation/tutorials on

the various tricks/optimizations a user can perform to improve simulation speed and/or fidelity from the environment object/robot models to simulation configurations.

E.3 Simulation+Rendering Benchmark Results

We ablate on a number of aspects of visual data collection where we simulate and render an environment. We record the FPS of taking 1000 random actions in the environment and fetching the 1000 visual observations. Overall Isaac Lab shows about 2-4x higher GPU memory usage compared to ManiSkill3, with the minimum amount of GPU memory taken up by just environments with cameras enabled being 4.8GB compared to 1.7GB in ManiSkill3.

E.3.1 Ablation on Realistic Camera Settings

We test on two realistic camera resolutions/setups based on the setup of the Open-X dataset (Collaboration et al., 2023) and the Droid dataset (Khazatsky et al., 2024). Many datasets in Open-X like the Google RT datasets have a single 640x480 resolution RGB camera, and the Droid dataset uses three 320x180 resolution RGB cameras. The results are shown in Figures 31 and 32. In both cases ManiSkill3 has about 2-4x better GPU memory efficiency and runs about 2x faster. Note that Isaac Lab's rendering output is a bit different from ManiSkill3, we share some qualitative examples in Appendix E.4. We further note that both ManiSkill3 and Isaac Lab support more photorealistic, high quality renders with full ray-tracing that is not parallelized. Example ManiSkill3 high-quality ray-traced renders can be seen in Figure 1.

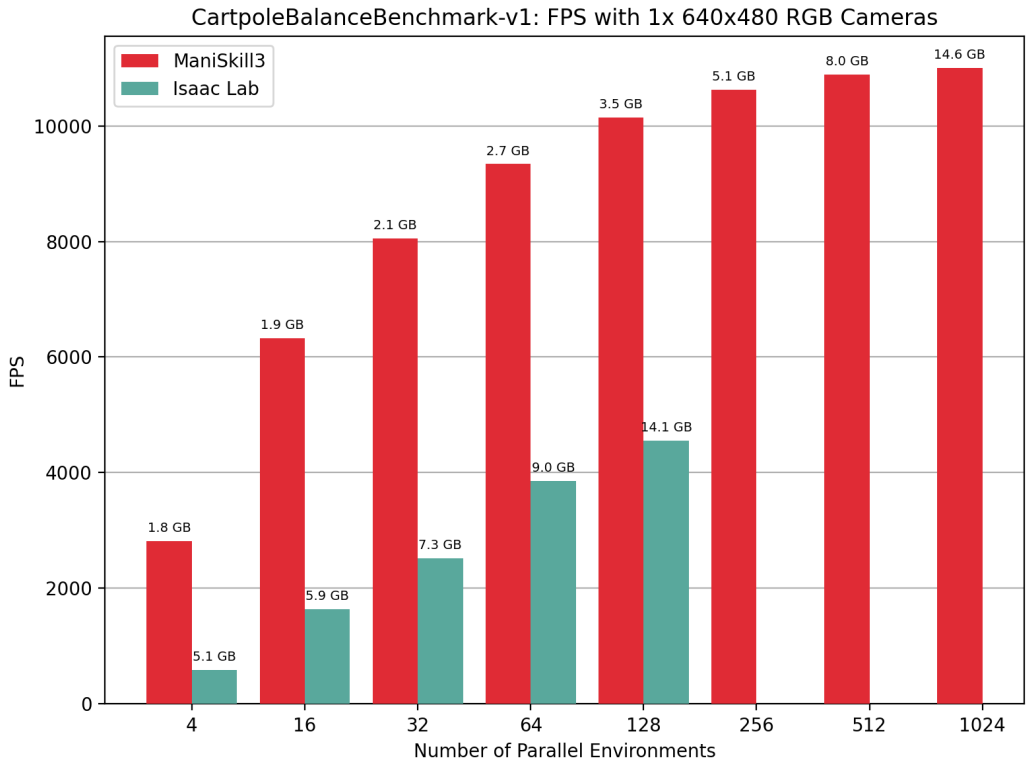


Figure 31: Simulation+Rendering of RGB or Depth FPS against number of parallel environments with 1x640x480 camera. Annotated numbers on top of data points indicate the GPU memory usage.

E.3.2 Ablation on Camera Size

We ablate on square camera sizes from a large 512x512 resolution to a small 128x128 resolution shown in Figures 33 34 35. We observe that Isaac Lab runs at most about 1.25x faster when there are

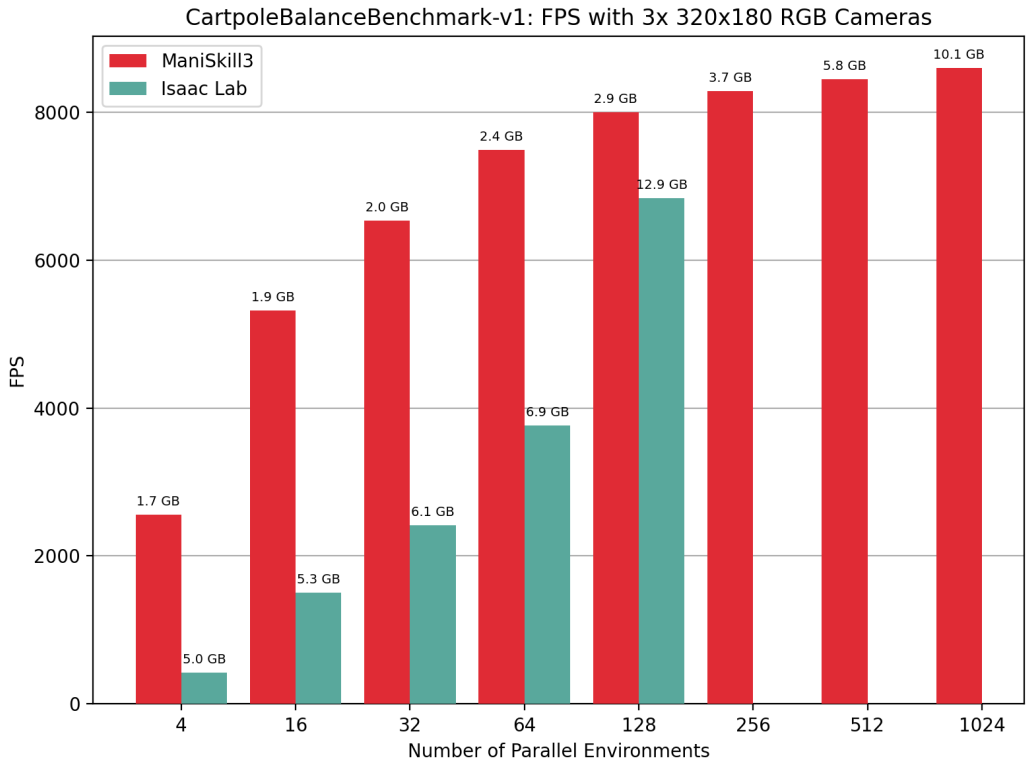


Figure 32: Simulation+Rendering of RGB or Depth FPS against number of parallel environments with 3x320x180 cameras. Annotated numbers on top of data points indicate the GPU memory usage.

a high number of parallel environments with small camera resolutions compared to ManiSkill3. At larger camera resolutions ManiSkill3 outperforms up to 2x in speed and 4x in GPU memory usage. Notably, ManiSkill3 always outperforms about 2-4x in speed and 2-3x in GPU memory usage for smaller number of parallel environments. While Isaac Lab is fast at smaller resolutions, we note that some manipulations tasks are fairly impractical and impossible to solve at small resolutions. Moreover memory efficiency is absolutely critical for RL applications that can remain fast by keeping large replay buffers on the GPU instead of using CPU memory. For this reason all visual RL baselines in ManiSkill3 typically do not use more than 256 to 1024 parallel environments as RL rollouts only run marginally faster with more environments beyond that point (for both Isaac Lab and ManiSkill3) but GPU memory use worsens a lot more.

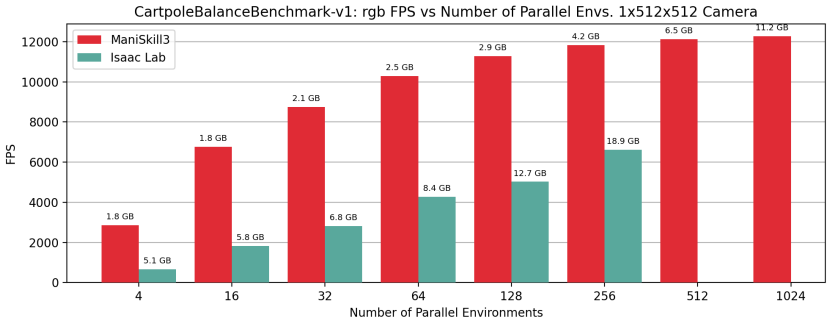


Figure 33: Simulation+Rendering (RGB) FPS against number of parallel environments 1x512x512 camera. Annotated numbers on top of data points indicate the GPU memory usage.

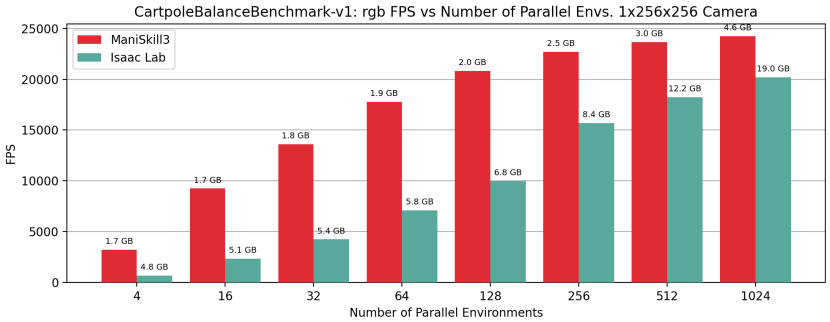


Figure 34: Simulation+Rendering (RGB) FPS against number of parallel environments 1x256x256 camera. Annotated numbers on top of data points indicate the GPU memory usage.

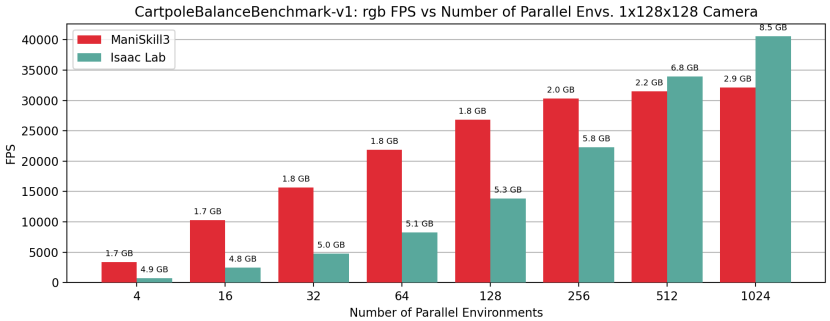


Figure 35: Simulation+Rendering (RGB) FPS against number of parallel environments 1x128x128 camera. Annotated numbers on top of data points indicate the GPU memory usage.

E.4 Qualitative Rendering Results

To the best of our capabilities we try to make the benchmarked environments look as similar as possible in terms of visuals. It is not possible to keep everything the same given fundamental differences in the parallel rendering system of Isaac Lab compared to ManiSkill3 or easily quantify the rendering differences. Section 3.2 details a little bit about the differences. Figure 3 in the main paper shows the cartpole RGB+Depth rendering results. Figures 36 and 37 shows RGB renders of a environment with a Franka Panda arm with different resolutions.

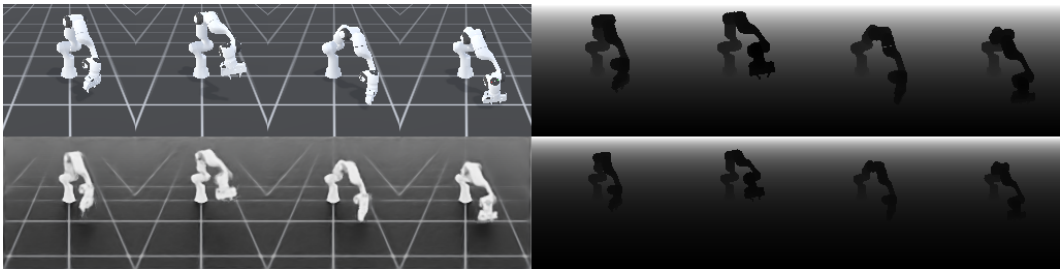


Figure 36: Comparison of ManiSkill3 (Top row) and Isaac Lab (Bottom Row) parallel rendering 128x128 RGB+Depth image outputs of the Franka Panda arm.

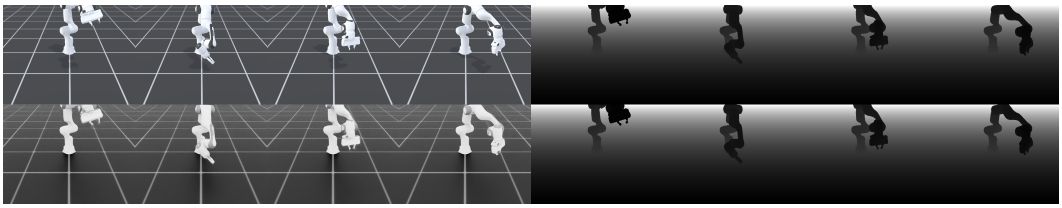


Figure 37: Comparison of ManiSkill3 (Top row) and Isaac Lab (Bottom row) parallel rendering 640x480 RGB+Depth image outputs of the Franka Panda arm.

F VR TELEOPERATION

ManiSkill3 provides VR support for all mainstream VR devices by implementing the OpenVR client protocol designed by Steam. Specifically, ManiSkill3 receives camera intrinsic and extrinsic parameters of the head-mounted display, VR controllers' poses, and the operator's hand and wrist poses if supported by the hardware; and it sends stereo video streams at 4K resolution for OpenVR to display in the headset. Under the hood, OpenVR communicates with SteamVR and ALVR which translate hardware-dependent VR implementation into the unified OpenVR client protocol.

The VR feedback loop runs at 60 Hz which is crucial for smooth user experience, while an asynchronous action translator translates sensed VR poses to robot actions at 20 Hz, so we can tolerate computationally intensive translation algorithms.

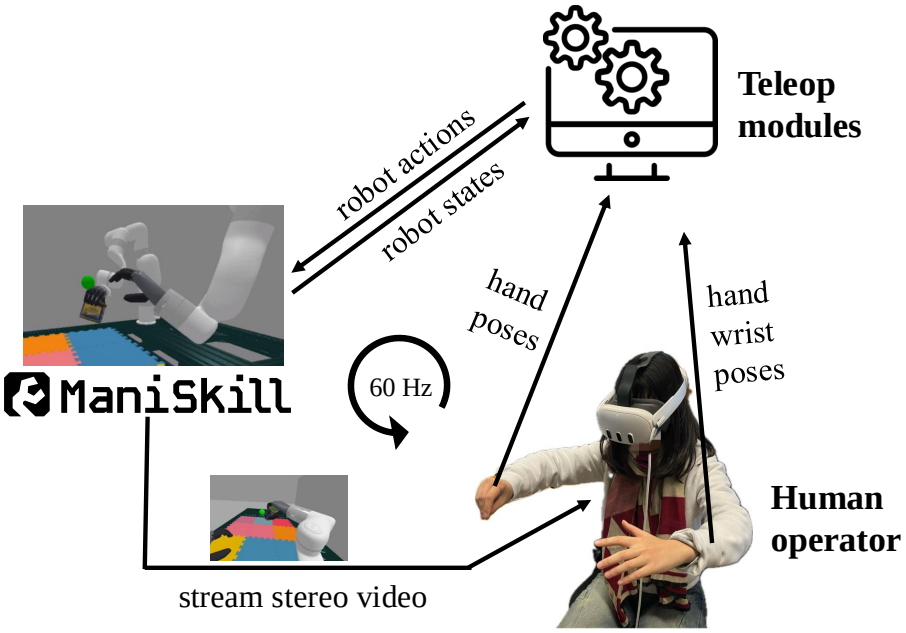


Figure 38: Overview of our VR teleoperation system. The operator controls the robot's arm and hand through real-time tracking of wrist and hand poses, while receiving stereo video feedback. The VR devices stream human pose data to a server, which retargets and sends joint commands to the robot.

Overview for our VR teleoperation system is shown in Figure 38. In a simulation setup, the stereo video is rendered based on simulated environments. For real-world robot teleoperation, the stereo video is generated from point clouds captured by depth cameras. We utilize the SAPIEN engine to achieve high-speed rendering of these high-resolution stereo videos. To simplify the complex SteamVR setup, we provide a Docker image for a smoother and faster setup process for users.

F.1 CONTROL

The human-motion-to-robot-action conversion system consists of three main modules:

1) **Arm Control Module**, which converts human wrist poses into robot arm joint positions. However, directly mapping the absolute orientation and position of wrist poses to robot's end-effector poses can lead to strange behaviors. For example there can be a mismatch between the coordinate frames of human wrist poses and the robot's end-effector frame as illustrated in Figure 40. To address this, we provide configurations out-of-the-box for the most common robotics arms. Additionally, we offer a GUI and tools to assist users in quickly computing the transformation matrix for their customized robots.

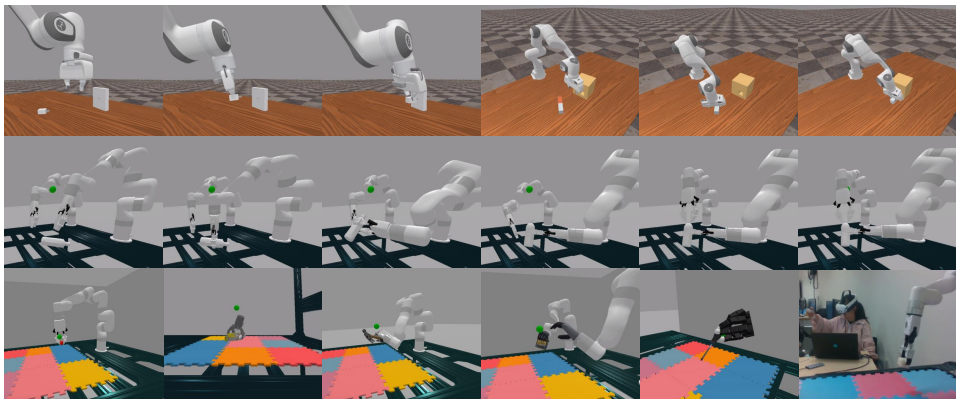


Figure 39: Demonstration of VR teleoperation across various tasks and robotic setups. Top row: Sequential images of a Panda arm performing precise insertions – charging connector insertion (left) and peg insertion (right) – showcasing accurate teleoperation for tasks requiring high precision. Middle row: Sequence of dual XArm7 robots with grippers demonstrating the grasping and unscrewing of a bottle, highlighting multi-arm coordinated teleoperation. Bottom row: Various teleoperation setups and manipulations, including (from left to right) XArm7 with a gripper, a floating Delto 3-finger hand, XArm7 with Inspire hand, dual XArm7 with Ability hand, a floating Allegro hand, and real-time teleoperation of an XArm7 with a gripper grasping objects.

Our system utilizes a modified version of the Closed-loop Inverse Kinematics (CLIK) algorithm, implemented with the Pinocchio library (Carpentier et al., 2015–2021; 2019), to calculate the joint angles of the robot's arm.

The approach uniquely addresses the problem of inverse kinematics (IK) for two coupled end-effectors simultaneously, which is critical in dual-arm robots with a sliding joint at the shoulders.

For a robot with n joints and two coupled end-effectors EE_i, we aim to find joint position q that minimizes the pose error for both end-effectors relative to their respective target poses T_i. Solving IK independently for two end-effectors suffers from large errors. Alternatively, iterative approaches usually result in slow convergence. Our solution extends the standard CLIK by computing a concatenated Jacobian that accounts for both end-effectors simultaneously. This enables faster and more accurate optimization of dual-arm configurations.

Additionally, the system allows users to control the movement of specific joints during coordinated motion. Instead of using a binary mask (0 for no movement, 1 for full movement) to constrain joints, we employ a soft mask with values between 0 and 1. A lower mask value reduces motion in the corresponding joint, leading to a solution that minimizes undesired movement. To ensure smooth arm motions, we apply an SE(3) group filter to the input end-effector poses before the IK computations.

2) **Hand Control Module**, which translates human finger poses into corresponding robot hand joint positions. Following (Qin et al., 2023; Cheng et al., 2024a; Ding et al., 2024), we formulate the hand motion retargeting process as an optimization problem. The objective function for this optimization is defined as follows:

$$\min_{q_t} \sum_{i=0}^{N} \|\alpha^i v_t^i - f_i(q_t)\|^2 + \beta \|q_t - q_{t-1}\|, \tag{1}$$

where q_t denotes the robot hand joint positions at time t, v_t^i is the i-th keypoint vector of human hand, and $f_i(q_t)$ gives the corresponding i-th keypoint vector of robot hand using forward kinematics with joint positions q_t. The scaling factor α_i compensates for the differences in hand size between the human and robot hands and treat each i-th keypoint differently as thumb finger size and pinky finger size can vary a lot, β weights the regularization term to ensure temporal consistency between consecutive joint positions. The optimization is implemented by NLopt solver Qin et al. (2023).

For dexterous robot hands, we map vectors from human hand fingertips to palm base to corresponding vectors on the robot hand. In certain cases, such as with the Inspire hand, additional

vector mappings are used to improve motion accuracy–for example, an extra vector from the thumb metacarpophalangeal joint to the thumb tip is employed. For grippers, we use a single vector optimization between the human thumb and index fingertips, which corresponds to the gripper's upper and lower ends. This allows intuitive control over the gripper's opening and closing motions through simple pinching gestures with the operator's index finger and thumb (Cheng et al., 2024a).

We fine-tune configurations for several common robots and provide calibration tools to assist users in adapting the system to their custom robots.

3) **Controller Control Module**, which is also used to control gripper motion, allowing simple and effective control of grippers. Though Apple Vision Pro does not include a physical controller, our system still supports controller-based input. By clipping the controller, users can trigger the closing action of the gripper, enabling intuitive and responsive control during operation.

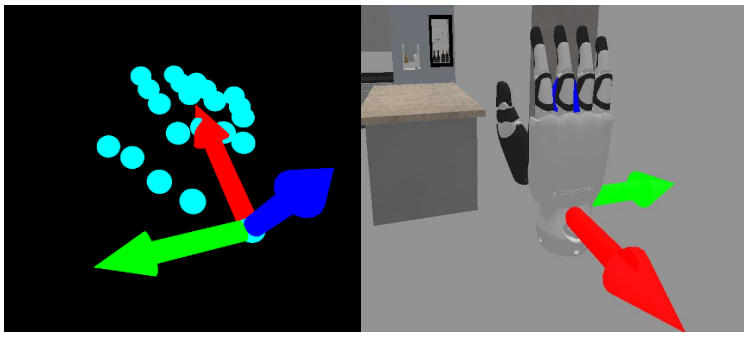


Figure 40: Illustration of frame mismatch between human wrist poses and the robot's end-effector frame. Left: human wrist frame, Right: Inspire hand frame as an example.

F.2 Sim-to-Real Interface

Our system employs a unified interface for both simulation and real-world setups by aligning the robot's end-effector with the absolute position of human hand. This alignment helps teleoperators better understand spatial information. Previous approaches (Qin et al., 2023; Cheng et al., 2024a; Ding et al., 2024) struggled to achieve spatial alignment between the human hand and the robot hand, forcing operators to compensate for parallax effects, which made the teleoperation experience less intuitive.

A key challenge arises when attempting to align the human hand with the robot hand inside a VR headset, especially since the real robot may be spatially displaced in the physical world. To solve this, we project the point clouds captured by depth cameras positioned around the real robot into the VR headset. The camera poses are calibrated using EasyHec (Chen et al., 2023).

This setup ensures that both the simulation environment and the real-world point cloud are aligned in a "digital twin" manner as illustrated in Figure 41. Though we do not require the visual textures to match the real world, critical elements such as the robot's position, forward and inverse kinematics, and control interface must be aligned. This alignment allows the same human control signals to produce identical robot actions in both the simulator and the real-world environment. Consequently, teleoperation in the real world becomes as intuitive and consistent as it is in simulation.

Figure 41: Illustration of spatial alignment of simulation and real-world environments. Left: simulation environment, where robot control are tested, Middle: real-world teleoperation setup, Right: point cloud captured from the depth cameras accurately aligned with the virtual robot in the simulation, demonstrating a ”digital twin” setup.