X2Real Technical Report

*An e**X**tensive simulation benchmark for **real**-world generalist policies*

X Square Robot

Generalist robot manipulation policies have developed rapidly, yet their reliable evaluation remains challenging due to fundamental flaws in existing simulation benchmarks: prominent sim-to-real gaps, narrow task coverage, and unfair evaluation caused by ambiguous training-test pipelines. Prior works only partially resolve these issues and lack simultaneous faithfulness, diversity, and fairness, while static benchmark designs fail to sustain long-term policy development. We presents X2Real, an evolvable simulation benchmark for faithfully evaluating the real-world performance of robotic manipulation policies based on Nvidia Isaac Lab-Arena. Following three core principles—faithfulness, diversity, and fairness—X2Real calibrates simulation visual and physical properties to align with real hardware, achieving a 0.84 linear correlation between simulated and real-robot evaluation results. It features a comprehensive taxonomy with 10 capability dimensions and 44 hierarchical long-horizon tasks, covering basic manipulation skills and advanced capacities such as visual grounding, language understanding, and bimanual control. We further adopt multi-axis domain randomization and strictly disjoint training-evaluation pipelines to mitigate benchmark exploitation and ensure credible evaluation. Powered by a custom physical domain-specific language, the Mana simulation ecosystem supports modular task design and iterative performance analysis, alongside a nearly 300-hour annotated simulation trajectory dataset. X2Real offers a faithful, diverse, and fair evolving evaluation infrastructure, effectively bridging the sim-to-real evaluation gap and supporting the advancement of generalist robotic manipulation policies.

Code: https://github.com/X-Square-Robot/x2real
Project Page: https://x2robot.com/en/pages/x2real

1 Introduction

As generalist robot manipulation policies like RT-2 (1), $\pi_{0.5}$ (2), DreamZero (3), wall-oss-0.5 (4), wall-wm (5), Cosmos3 (6) advance rapidly, evaluating what they can actually do becomes increasingly important and difficult. Real-robot evaluation (7, 8) is the gold standard, but it is costly, slow to yield feedback, and hard to reproduce. Simulation benchmarks sidestep these problems by running robot–environment interactions in parallel, at low cost and with full reproducibility. Yet three caveats keep current simulation benchmarks from serving as a faithful proxy for real-world ability.

Sim2real gap. A simulation score is faithful only if it predicts real-robot performance, yet discrepancies in visual appearance, physical dynamics, and robot control create a gap between the two, a.k.a, the simulation-to-real gap (sim2real gap). More specifically, we care about the linear correlation between the scores of the same model evaluated in simulation and on a real robot; ideally, this correlation should approach 1.0. Without such predictivity, a high simulation score may reflect only how well a policy exploits the simulator, rather than how well it will perform once deployed.

Narrow evaluation distribution. A simulation task is inherently multi-dimensional: it determines which policy capabilities are exercised, which atomic manipulation skills are required, and—crucially—whether it probes a policy's true competence or merely its (over)fitting to a fixed setup. For an evaluation to effectively reflect a policy's capability, its tasks must span a diverse distribution along every such dimension: assets, layouts, instructions, task designs, and so on. Constrained by the availability of assets, simulators, and demonstration data, however, prior benchmarks typically cover only a handful of largely pick-and-place tasks built on a fixed

Figure 1 X2Real Task Taxonomy. X2Real features a comprehensive task taxonomy spanning 10 core capability dimensions and 44 hierarchical long-horizon tasks. It leverages full multi-axis domain randomization with strictly disjoint training-evaluation pipelines to remove benchmark biases and enhance real-to-sim generalization fidelity.

set of assets. Such a scope was adequate for early policies, but these benchmarks saturate quickly as policies improve.

Benchmark exploitation. Ideally, a policy would be evaluated fully zero-shot, with every task unseen. In practice, current policies still generalize poorly zero-shot and typically require some post-training to perform these tasks, so a benchmark must also supply fine-tuning data. This calls for a strict separation between the data used for training and the setup used for evaluation. Such separation is hard to maintain in simulation: because a simulated environment is deterministic and easily reproduced, a policy can be optimized toward the specific evaluation setup, and in the extreme the leaderboard can be gamed by fixing random seeds or memorizing particular scenes. A narrow task distribution only widens this room for exploitation.

Together, these three issues—the sim2real gap, narrow evaluation distribution, and benchmark exploitation—determine whether a simulated evaluation can reflect a policy's capability in a faithful, effective, and fair manner, respectively. Prior work has largely addressed only one of them at a time, directly or indirectly. SIMPLER studies sim–real correspondence through paired simulation and real-robot evaluation (9), followed by PolaRis (10) and REALM (11) and other works, but their task suites cover only a limit distribution of robotic capabilities. BEHAVIOR-1K (12), Robocasa365 (13), Robotwin2.0 (14) and many other works cover a wide range of tasks, but do not ground their simulation score on the real performance. Recently as our concurrent work, Robodojo (15) provides a comprehensive sim-and-real benchmark on a diverse task distribution, but regrettably does not include an explicit sim2real analysis. Moreover, even once these issues are resolved, any static benchmark will eventually saturate as policies improve; what is ultimately needed is an infrastructure that can continually evolve its environments and tasks to serve the long-term development of embodied intelligence.

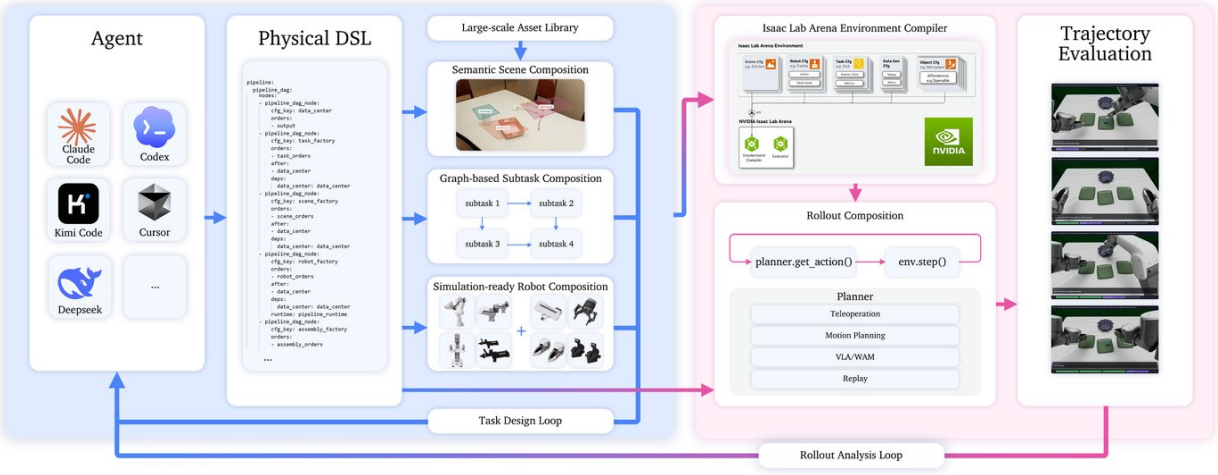


Figure 2 Mana System Overview. Physical DSL enables agents to drive task design and rollout analysis in closed loops, from semantic scene, subtask, and robot composition to environment compilation, policy rollout, and trajectory evaluation.

To tackle these problems, we propose **X2Real**, an e**X**tensive simulation benchmark aimed to test policies' capabilities under **real-world** deployment. Guided by three core design principles—faithfulness, diversity, and fairness—X2Real mitigates the aforementioned limitations through hardware-aligned simulation calibration, a systematic capability taxonomy, comprehensive domain randomization, and a strictly separated training-evaluation data pipeline.

To reduce the sim2real gap, we calibrate multiple robot embodiments visually and physically, and optimize low-level control parameters to achieve millimeter-level trajectory-replay consistency against real-world hardware. We achieve an **0.84** linear correlation coefficient for simulation and reality evaluation scores on a model trained only on real data. Moving beyond the limited scope of conventional pick-and-place benchmarks, X2Real covers hierarchical reasoning and manipulation skills including visual perception, language grounding, memory, and precise bimanual control. In total, the benchmark comprises 44 simulation tasks organized across 10 capability dimensions, as shown in Fig. 1. These tasks encompass fundamental atomic skills such as pick, place, twist, push, and pull, while further increasing difficulty by composing these primitives into long-horizon challenging sequences.

Unlike previous benchmarks that mostly only perform visual randomization, our framework enables task-customized randomization across three orthogonal axes: **appearance, task setting, and embodiment**. This design ensures that evaluation results reliably reflect the policy's true manipulation capabilities. To counter benchmark exploitation, we enforce strict separation between training and evaluation environments. All demonstration trajectories are collected within a green-booth setup adapted from ManipArena (8) with constrained randomization, which is kept distinct from evaluation configurations. Via teleoperation and automatic synthesis, we assemble a high-quality dataset of nearly **300 hours** of simulation trajectories, annotated with frame-level subtask progress labels.

To realize X2Real's demanding requirements for high task diversity, physical faithfulness, we construct an agent-driven unified simulation ecosystem **Mana** based on Nvidia Isaac Lab-Arena, depicted in Fig. 2. At its core lies our physical domain-specific language (Physical DSL), a declarative abstraction layer that unifies the specifications of scenes, robot embodiments, subtask dependency graphs, and multi-axis domain-randomization rules into modular, reusable primitives. It enables the agent to interact with the underlying simulation stack in a safer, more controllable, and fully declarative fashion. The DSL drives two main loops: the **Task Design Loop** for benchmark authoring and the **Rollout Analysis Loop** for rollout execution and iterative refinement. Within the Task Design Loop, scene, subtask-graph and robot-embodiment definitions are assembled and compiled into runnable simulation environments via the Isaac Lab Arena Environment Compiler. In the Rollout Analysis Loop, multiple execution backends generate frame-annotated trajectories; evaluation signals flow back to agents to analysis model performance. These together make X2Real an evolving benchmark for future

development of generalist policies.

To support standardized evaluation across heterogeneous policy architectures, X2Real further introduces **Policy Space**, a unified runtime interface that decouples policy-specific inference pipelines from the simulation and evaluation infrastructure. Policy Space standardizes observation mapping, action adaptation, inference scheduling, history and internal-state updates, and episode lifecycle management, while allowing each policy to retain its native preprocessing and action representation. This abstraction enables policies with substantially different runtime requirements to be evaluated under exactly the same task definitions, observations, randomization rules, and evaluation criteria. Using this protocol, we benchmark four representative fine-tuned generalist policies across the full X2Real task suite. The results reveal substantial capability differences across reasoning and manipulation dimensions, as well as a pronounced degradation from in-distribution (ID) to out-of-distribution (OOD) evaluation. For example, even the strongest evaluated policy decreases from an overall ID success rate of **53.9%** to an OOD success rate of **34.3%**, with particularly large gaps on tasks requiring generalization, language reasoning, memory tracking, and fine-grained manipulation. These results demonstrate that current generalist policies remain far from saturating X2Real and highlight the importance of evaluating not only task completion, but also robustness across diverse, unseen deployment conditions.

2 Benchmark Overview

2.1 Multi-dimensional Task Suite

We systematically evaluate the overall competencies of generalist robotic manipulation policies by dividing policy behaviors into two canonical, complementary dimensions: Reasoning, which governs task decision-making (i.e., what to do), and Manipulation, which governs physical execution (i.e., how to do it). The overall taxonomy is visualized in Fig. 1. The Reasoning dimension further decomposes into four standardized sub-capabilities:

1. Generalization: It characterizes the ability of a policy to stably perform basic task operations under diverse environmental perturbations.
2. Visual Understanding: It characterizes the ability of a policy to perceive and distinguish visual task cues, including object shapes, colors, and numbers, etc.
3. Language Understanding: It characterizes the ability of a policy to accurately follow structured linguistic instructions, such as parsing logical descriptions and interpreting row-column positional specifications.
4. Memory: It characterizes the ability of a policy to retain and utilize historical information for completing context-dependent sequential tasks.

Accordingly, the Manipulation dimension consists of five execution-oriented core capabilities:

1. Precision Operation: It characterizes the ability of a policy to control robot end-effectors for stable, high-precision motion execution.
2. Bimanual Coordination: It characterizes the ability of a policy to implement collaborative motion control for dual robotic arms.
3. Tool Usage: It characterizes the ability of a policy to operate daily functional tools (e.g., hammers, brooms, screws) to satisfy task demands.
4. Dynamic Operation: It characterizes the ability of a policy to interact steadily and effectively with moving objects.
5. Mobile Operation: It characterizes the ability of a policy to accomplish manipulation tasks that couple mobile-base navigation with object interaction across spatially separated workspaces.

Beyond standard capability assessment, we design three challenging tasks to probe the performance ceiling and quantify the upper competence bound of generalist manipulation policies:

1. Stack Blocks (Hard): The robot stacks 15 uniformly sized blocks into the tallest feasible structure. This task evaluates the upper precision limit of policy motion execution.
2. Press Buttons (Hard): The robot presses buttons strictly according to a predefined color sequence with more than ten ordered cues. This task examines the upper bound of policy long-term memory capacity.
3. Hanoi Tower: The robot solves the classic 5-layer Tower of Hanoi problem. This task assesses the high-level reasoning and sequential planning capabilities of the policy.

Our full task suite is documented in the Appendix A.1. These tasks span diverse low-level atomic manipulation skills, such as picking, placing, twisting, pushing, and pulling. All tasks decompose into hierarchical subtasks that form a Directed Acyclic Graph (DAG). A rule-based evaluator determines the success of each individual subtask. This design supports fine-grained episode-level progress monitoring in addition to conventional final success rate evaluation. Detailed configurations of the task construction pipeline are provided in Sec. 3.3.

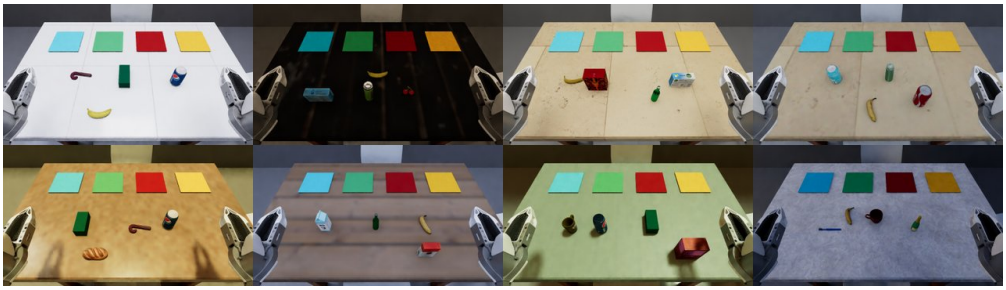


Figure 3 Domain Randomization Example. The lighting condition, table material, task assets can be randomized for the Classify by Color task.

2.2 Diverse Evaluation Distribution

To mitigate the narrow evaluation distribution and benchmark exploitation limitations prevalent in existing simulation benchmarks, we equip all designed tasks with multi-dimensional domain randomization. Our randomization scheme falls into three orthogonal axes to guarantee diverse, robust, and unbiased evaluation:

1. Appearance: It covers background scenes, lighting conditions, and table surface textures.
2. Task Setting: It covers task assets, object placement, distractor layout, linguistic instructions, and puzzle solution configurations.
3. Embodiment: It covers gripper types, camera intrinsic and extrinsic parameters, and table heights.

This multi-axis diversity design ensures that each task reliably targets its corresponding policy capability and avoids overfitting to fixed simulation configurations. We take the "Classify by Color" task as a representative case, as shown in Fig. 3: the robot places red, blue, green, and yellow objects onto color-aligned target papers. By dynamically randomizing scene appearance and object assets, the task strictly verifies visual understanding robustness. Policies must extract discriminative color features for zero-shot classification instead of relying on superficial cues such as object shape, function, or placement position.

We adopt hierarchical randomized settings for category-specific evaluation. Generalization-oriented tasks enable all three axes of randomization; other Reasoning tasks adopt appearance and task-setting randomization; Manipulation tasks adopt appearance-only randomization. This stratified design balances evaluation diversity and task specificity.

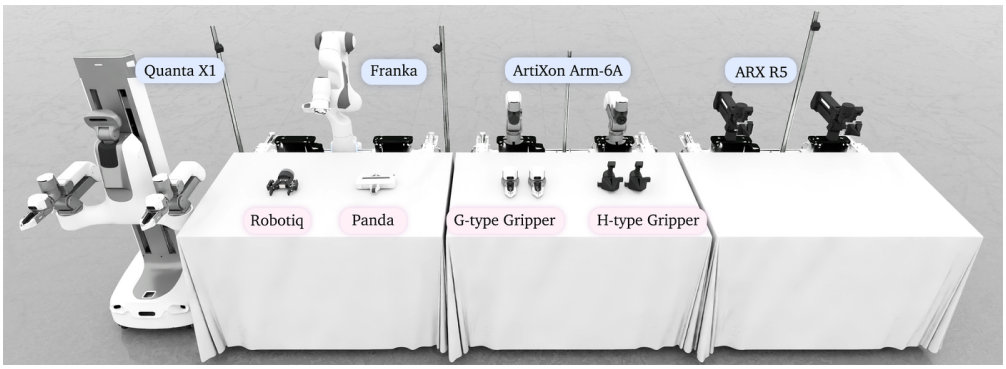


Figure 4 X2Real Embodiments. X2Real includes mobile, single-arm, dual-arm robots with replaceable grippers.

We support diverse robot embodiments with replaceable gripper configurations, covering multiple commonly used and self-developed robotic platforms. The supported systems include the single-arm Franka equipped with Robotiq Gripper and Panda Hand, the self-developed dual-arm ArtiXon Arm-6A assembled with G-type Gripper and H-type Gripper, the dual-arm ARX R5, as well as the self-developed mobile manipulation platform Quanta X1. We calibrate the visual appearance and physical parameters of each simulated embodiment strictly against their real-world counterparts to reduce simulation bias.

We conduct the majority of data collection and experimental validation on the self-developed ArtiXon Arm-6A and Quanta X1 platforms. To further narrow the real2sim gap, we optimize the low-level control parameters of robotic arms. This optimization guarantees millimeter-level positional deviation between simulated and real robot arms under identical control action inputs. We provide detailed implementation and calibration specifics in Sec. 4.1.

2.3 Training-evaluation Separation

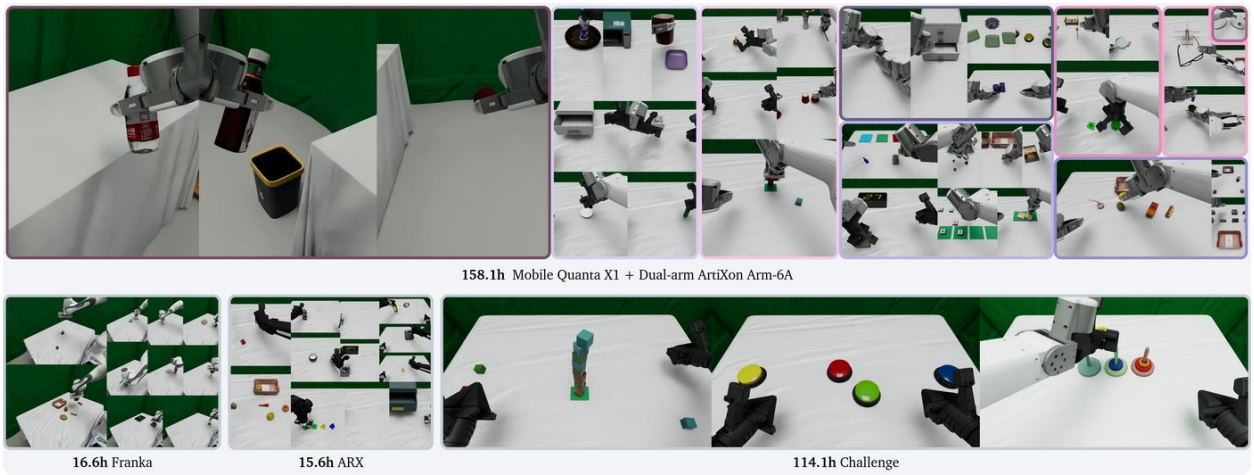


Figure 5 X2Real Dataset. we collect nearly 300 hours high-quality demonstration data on different embodiments. The image area of each task is proportional to its data length.

To strictly decouple training configurations from evaluation setups and avoid benchmark exploitation, we build a digital twin of the green-screen simulation environment aligned with ManipArena (8) and conduct all data collection within this standardized virtual setting, as visualized in Fig. 5. We collect 300 demonstration trajectories for each desktop manipulation task, 1000 trajectories for each mobile operation task, and over 20000 trajectories for the three challenging tasks. For data source distribution, approximately 80% of the

ArtiXon Arm data comes from human teleoperation, while the remaining 20% is generated via automatic data synthesis. In contrast, all data from the Quanta X1 mobile platform, Franka, ARX R5, and all challenging tasks are fully synthesized automatically. In total, we collect 158.1 hours of trajectory data from Quanta X1 and ArtiXon Arm as the primary post-training dataset for our benchmark. We further supplement 16.6 hours of Franka data, 15.6 hours of ARX R5 data, and 114.1 hours of challenging task data, yielding a total dataset duration of nearly 300 hours.

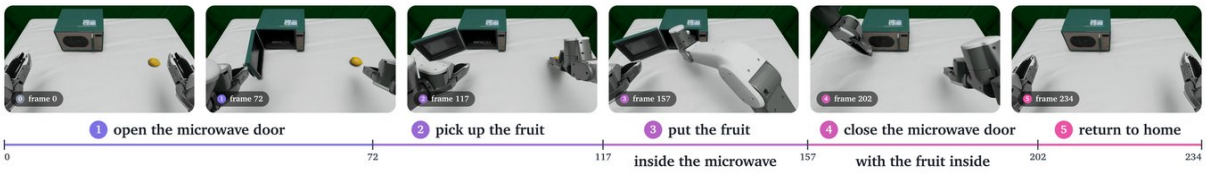


Figure 6 Frame-level Labeling. The Fruit in Microwave task contains 5 subtasks. Our system can automatically label corresponding frame duration during data collection.

We enable the subtask-level success checker throughout both teleoperation and automatic synthesis pipelines, which supports automated temporal labeling for all collected trajectories. As illustrated in Fig. 6, each trajectory not only retains raw camera recordings, robotic action sequences, and linguistic instructions, but also contains fine-grained frame-level subtask annotations that precisely document the complete task execution process.

Importantly, X2Real goes beyond a static separation between training and evaluation configurations. Powered by the procedural generation capabilities of the Mana ecosystem, it can continuously instantiate new combinations of assets, layouts, task parameters, instructions, and embodiment configurations under predefined randomization rules. As a result, policies cannot simply memorize a finite collection of evaluation setups or optimize against fixed scene configurations. More importantly, the evaluation distribution itself can evolve over time by introducing new randomized configurations and task variants, reducing benchmark-specific overfitting as policy capabilities improve.

2.4 Unified Evaluation Protocol

Generalist robot policies often differ in their runtime dependencies, observation formats, history and state requirements, action representations, and inference schedules. To evaluate these policies under a unified protocol, X2Real adopts a dependency-isolated architecture (7, 15, 16, 17), in which the simulation client and each policy service run in separate processes and communicate through a lightweight remote interface. Building on this architecture, we introduce **Policy Space**, which defines a unified runtime contract between the simulation client and policy services across robot embodiments. As shown in Fig. 7, Policy Space coordinates observation mapping, action adaptation, and the episode lifecycle. Policy-specific preprocessing and inference remain within each policy service, allowing policies with different native interfaces and inference requirements to be evaluated under the same protocol.

X2Real uses an episode as the basic unit of closed-loop evaluation. Each episode is instantiated from an evaluation configuration that specifies key settings such as the task definition, robot embodiment, language instruction, randomization rules, and sampling seed. It starts with environment initialization and ends when the task succeeds, a failure condition is triggered, or the time limit is reached. For each policy and task, X2Real runs multiple episodes under randomized conditions and reports the success rate, mean progress score, and subtask completion rates.

At each control step, the environment produces an observation. Based on the current embodiment configuration, Policy Space maps the observation into the common format used by policy services. The observation content may vary with the task and embodiment. When inference is requested, the model adapter converts the mapped observation into the policy's native input format and passes it to the policy for inference. The policy output is then returned to Policy Space, which converts it into commands executable by the current robot.

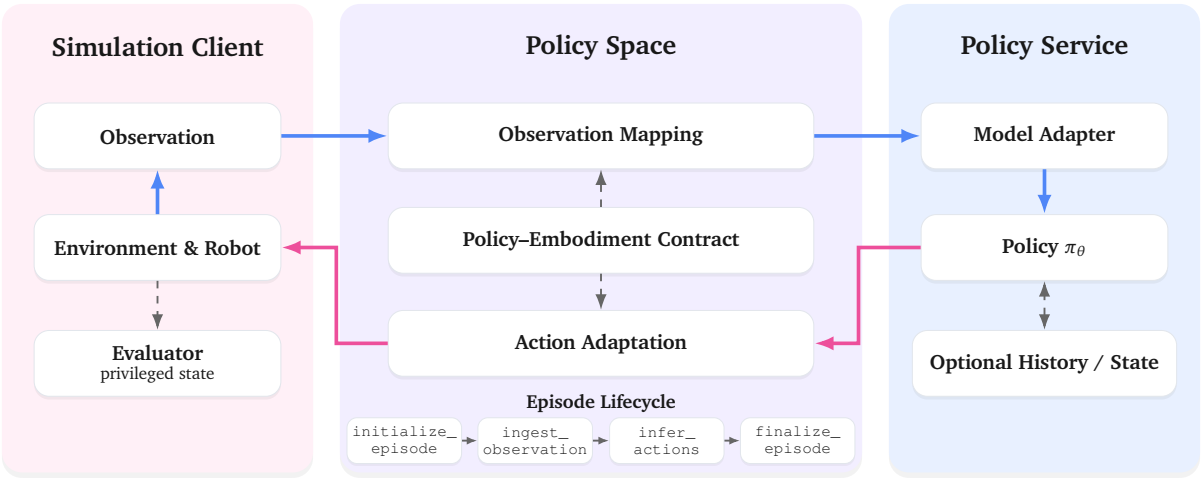


Figure 7 Policy Space Overview. Policy Space coordinates observation mapping, action adaptation, and the episode lifecycle. Within each policy service, a model adapter converts mapped observations into model-specific inputs for policy inference. Privileged simulator state is accessible only to the evaluator.

Policy Space uses configurable modules to accommodate policy- and embodiment-specific processing beyond the common observation–action flow. Examples include coordinate and action-representation transformations, embodiment-specific unit conversion, Real-Time Chunking (RTC) (18), and history updates for history-dependent policies (3). Related modular interfaces are used in existing robot-learning toolchains and policy-integration frameworks (19, 16, 20).

Policy Space defines the episode lifecycle through four operations. `initialize_episode` prepares the policy for a new episode; `ingest_observation` passes the current observation to the policy service to update any required history or internal state; `infer_actions` requests the next policy output; and `finalize_episode` marks the end of the episode and clears episode-specific state. The evaluation configuration also specifies when observations are passed to the policy service, when inference is requested, and how policy outputs are executed. The same lifecycle applies to policies with different history and state requirements.

Policy inference is separated from task evaluation. Policies receive only the observations provided through Policy Space, while the evaluator uses privileged simulator state that is not exposed to the policy. During each episode, the evaluator records stage completion, task success, progress, and failure information based on the task definition. The resulting evaluation metrics are detailed in Sec. 3.3.3. For each evaluation setting, all policies use the same task definition, randomization rules, observation content, and evaluation criteria.

3 Mana Simulation Platform

3.1 Agent-driven System Design

Large-language-model (LLM) agents exhibit strong capabilities in coding and mathematical reasoning (21, 22). Nevertheless, their potential for physical AI research remains largely untapped. Ideally, such agents could assist the full pipeline of embodied research: on one hand supporting benchmark and evaluation workflows, including task design, environment composition, and policy rollout; on the other hand facilitating model evolution via trajectory generation, data augmentation, and fine-grained performance analysis. However, the inherent complexity of low-level simulation stacks prevents LLM agents from realizing these capabilities at scale. Direct interaction with simulators requires handling sprawling low-level APIs, physical parameter tuning, robot embodiment configurations, scene assembly, and hand-crafted task success logic. Unmediated LLM calls to such interfaces frequently produce invalid configurations, runtime errors, and brittle task implementations, making large-scale autonomous iteration impractical. Isaac Lab Arena (23) simplifies low-level simulation interfaces by factoring environments into three composable building blocks: scene, task, and embodiment, which can be flexibly assembled to construct diverse simulation scenarios. However, it lacks

built-in asset management facilities and integrated data pipelines, and offers no agent-centric framework to enable model-evolution workflows. To unlock this agent-simulation closed-loop workflow, we present Mana, a comprehensive agent-driven simulation infrastructure.

As shown in Fig. 2, Mana is organized around a YAML-based physical Domain-Specific Language (DSL). Rather than replacing Arena's scene–task–embodiment factorization, the DSL adopts it as the runtime abstraction and supplies the pieces Arena leaves out: asset registration and placement, a way to name and compose the three factors, and a unified rollout path. Everything an agent touches is expressed as a modular YAML primitive, and a compile-and-check harness sits behind these primitives: it expands `include` references, lowers the result through a *Translator* into factory orders, and validates the whole pipeline before any simulator process starts. An agent can therefore assemble a valid environment by editing declarative text, without ever calling a low-level simulator API. We describe this design in four steps—the authoring surface, compilation into factories and orders, assembly, and rollout—and then return to the two loops the agent runs on top of it.

3.1.1 Authoring Surface

The DSL is organized at two levels. A *pipeline* file is the top-level program: it declares which factories are active, the order in which they compile, and the streams of orders they consume. The substance of an environment, however, lives in *fragments*—independently authored YAML files for scenes, tasks, robot embodiments, policies, and collection presets—which the pipeline pulls in with `include`. Because inclusion expands a referenced fragment and then lets the including file override individual fields, a new variant is written as a small overlay rather than a duplicated file, as in Listing 1.

```
include: scene/pick_place.yaml
definition:
  scene_type: pick_place_ood
```

Listing 1 A scene variant authored as an overlay.

Keeping independent concerns in independent fragments is what makes an edit local: changing one fragment does not perturb the others. A scene fragment, for instance, describes only the world—its assets, a background with the surfaces it must expose, and a layout—and says nothing about the robot or the success criteria (Listing 2). Each object carries an identity (a concrete asset or a pool of candidates), a set of `semantic_tags` that later act as role handles, and a placement, which may be a fixed pose or a sampled region on a surface. Optional variant blocks randomize assets, materials, or lighting, and a layout solver discards colliding placements before the scene is admitted, so an authored scene is guaranteed to be physically realizable.

```
objects:
  - id: mug
    asset: {type: pool, candidates: [{id: mugs, assets: [mug_a, mug_b]}]}
    semantic_tags: [pickable]
layout:
  nodes:
    mug:
      object: mug
      placement: {type: surface, surface: tabletop, ranges: {x: [0.10, 0.20], y: [-
    0.10, 0.10]}}
```

Listing 2 A scene fragment: identity, semantic tags, and placement.

A task fragment describes the logic as a directed acyclic graph. Graph-level fields hold the instruction and the reset-time sampling, while each node is an atomic subtask equipped with role slots, declarative success terms, optional sensors, and a partial score; the edges impose a partial order over these nodes (Listing 3). Since node files can themselves be included, a single stage is reused across many graphs. Success terms are named functions resolved at compile time rather than hand-written checkers, and a task refers to the scene only through tags such as `@pickable`, never through concrete asset paths. Atomic skills (`pick`, `place`, `push`,

and related primitives) may additionally attach on the data-generation path, where a skill or motion planner consumes them; evaluation itself relies only on the success terms.

```
graph_task_generator:
  nodes:
    - {id: place, include: task/pick_place/stages/place.yaml, score: 3}
    - {id: return, include: task/pick_place/stages/return.yaml, score: 1}
  edges:
    - {src: place, dst: return}
# place.yaml
success:
  _func: object_near_destination
  object_cfg: '@pickable'
  destination_cfg: '@container'
```

Listing 3 A task fragment: a DAG whose nodes carry declarative success terms.

A robot fragment, finally, specifies an embodiment—its kinematics, gripper variant, cameras, and actuator law—while policy and collection fragments specify how a model is served and how a teacher is attached. Crucially, no fragment ever embeds a complete environment; each describes exactly one concern.

3.1.2 Factories and Orders

These authoring files are never interpreted directly at runtime. Instead, the Translator expands every `include` and lowers the pipeline into two artifacts: factory configurations and *orders*. A factory is simply a lifecycle—setup, receive, fulfill, cleanup—whereas an order is a named request to generate one thing: a particular scene, robot, task, assembly, or rollout (Listing 4). This separation is what keeps agent edits safe: because factories consume already-lowered orders rather than raw YAML, an agent extends the system by writing another order, not by patching factory code.

```
scene_factory: {type: default_scene_factory}
scene_orders:
  - include: scene/pick_place.yaml          # scene_type: pick_place
  - include: scene/pick_place_ood.yaml      # overlay, scene_type: pick_place_ood
robot_orders:
  - {include: robot/arm.yaml, name: ARM-G, gripper_variant: G}
task_orders:
  - include: task/pick_place/vnext_dag.yaml
```

Listing 4 Order streams reference fragments by name.

Fulfilling an order follows a single chain: the factory batches and caches work, a generator composes one complete object, and components implement the atomic capabilities—background, objects, lighting, termination, events, metrics—so that extending the language usually means adding a component rather than branching a factory. The scene, robot, and task factories act as independent part suppliers; once shared data services are up, they can run in parallel and store their outputs by name in a shared data center.

3.1.3 Assembly

An assembly order is a triple of generator names. Given such a triple, the assembly factory fetches the three parts from the data center, binds each task role to the scene assets whose tags satisfy it (so `@pickable` resolves to objects tagged `pickable`), and emits an environment for the Arena compiler to instantiate (Listing 5). Episode length and step limits remain properties of the task, so assembly stays purely compositional and never owns the MDP.

```
assembly_orders:
  - assembly_generator:
```

```
name: arm_G_pick_place_env
scene_generator_name: pick_place
robot_generator_name: ARM-G
task_generator_name: pick_place
```

Listing 5 An assembly order binds scene, robot, and task by name.

Because binding is by name, a large matrix of environments follows from a small set of parts without duplication: one scene order can appear in many assemblies, and one task graph can be paired with several gripper variants. The pipeline DAG fixes the only legal compile order—data services, then the three part factories, then assembly, then rollout—and a program that violates it, for example by assembling before its parts exist, is rejected at dry-run, before the simulator is ever launched.

3.1.4 Unified Rollout

A rollout order reuses an environment rather than rebuilding it: it names an existing `env_ref` and the teacher that will drive it, be that teleoperation, a motion or skill planner, policy inference, or trajectory replay. All teachers share the same get-action-and-step loop, so a single assembly can carry both a data-generation order and an evaluation order (Listing 6). Both emit trajectories annotated with per-node DAG scores and instruction logs, which makes fine-grained analysis an intrinsic property of the task graph rather than a separate instrumentation layer.

```
rollout:
  - {env_ref: arm_G_pick_place_env, modes: [gendata], include:
    datacollection/teleop.yaml}
  - {env_ref: arm_G_pick_place_env, modes: [benchmark], planner: {PolicyPlanner: {
    policy_cfg: {include: policy/vla.yaml}}}}
```

Listing 6 One assembly, two rollout modes: data generation and evaluation.

3.1.5 Agent-driven Loops

Equipped with this DSL, an agent drives the two loops named in the opening—benchmark construction and model evolution—entirely through the harness rather than through simulator APIs. The agent remains the author of each program; the harness merely guarantees that every edit is compilable and every run attributable.

In the task-design loop, the agent writes or patches fragments, appends or rewires orders, validates the pipeline, and dry-runs the compile graph. Such edits are naturally local: including a different scene overlay, retargeting an assembly to another robot, adding a node to a task DAG, or attaching a new teacher to an existing `env_ref`. The Translator then lowers the edited program and the assembly factory instantiates fresh environments. Should the schema prove insufficient, the agent can still extend a component directly, but the common path never requires touching a simulator API.

In the rollout-analysis loop, the agent launches data generation or evaluation on selected environments, reads the node-level metadata and failure trajectories they produce, and feeds these observations back—either as further edits to the DSL or as sliced demonstrations for augmentation. The human contributes only high-level intent at the pipeline scale, while the agent carries out the iteration within the language itself.

3.2 Simulation-ready Assets and Semantic Scene Construction

X2Real connects three content-side capabilities: simulation-ready asset preparation, semantic scene construction, and task-conditioned variation with state-aligned replay. Their factorization supports evaluation fidelity by reducing content-side simulation-to-real discrepancies, evaluation breadth through task-role coverage and constraint-preserving composition, and benchmark integrity through explicit identities, controlled content hold-outs, and resampling.

Asset preparation, including AI-generated 3D (AIG3D) synthesis, creates simulation-ready packages; the scene program and compiler resolve semantic placement intent into compiled scenes; reset-time variation and replay diversify episodes and observations.

Figure 8 Task-oriented tabletop asset spectrum. Representative packages span 44 rigid categories and nine articulated interaction groups. Rigid examples are arranged in four visual groups for readability; the collection taxonomy contains ten semantic families.

3.2.1 Task-oriented Simulation-ready Assets

A simulation-ready asset is a reusable package that bundles visual, metric, semantic, collision, inertial, kinematic, and physical-material records and has passed the simulator checks required by its declared roles. These packages determine interaction coverage and support reliable instantiation of heterogeneous content in task scenes.

X2Real organizes the tabletop collection around *task roles and interaction modes*; raw catalog size and generic taxonomies alone do not establish manipulation coverage. Manipulated objects, receptacles, tools, fixtures, and articulated mechanisms must support the benchmark interactions, while category breadth provides visual and semantic diversity within each role.

Existing ecosystems emphasize complementary parts of the asset–task interface, including reusable task definitions, realistic content, metric or grasp annotations, and language-conditioned interaction metadata (24, 25, 13, 26, 14, 15). X2Real integrates these signals with validated metric scale, part- and task-level interaction records, physical-material authoring, and simulator-side acceptance in a common simulation-ready asset package.

The collection combines Objaverse/CanoVerse, Omni6DPose, BlenderKit, GSO, ManiTwin-100K, HSSD, ProcTHOR, ABO, Poly Haven, and ReplicaCAD with task-specific and generated additions (27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37). Each accepted unit is released as one *benchmark package*. The indexed rigid catalog contains 96,480 source records; the tabletop release contains 12,485 rigid packages spanning 44 leaf categories and 10 semantic families, and the articulated collection contains 670 packages across nine interaction groups. AIG3D additions are included in the tabletop rigid total, while deformable assets form a separate collection. Counts are package-level, so geometrically similar content from different sources may remain distinct packages. Figure 8 shows the visual and interaction breadth of the released collections; detailed provenance appears in Appendix A.2.

To turn collected content into simulation-ready packages, X2Real uses seven dependency-ordered stages that account for interactions among geometry, appearance, scale, semantics, and physics (Fig. 9). The pipeline preserves trustworthy source evidence, repairs recoverable defects, and revalidates affected downstream records before release.

Source intake and scope screening. Sourced and generated content enters a common OpenUSD representation. Intake screens parseability, benchmark-domain fit, supported roles, and object coherence while preserving reliable geometry, texture, and material evidence. Compound scenes are decomposed when possible; ambiguous content is reviewed, and malformed or out-of-scope content is excluded.

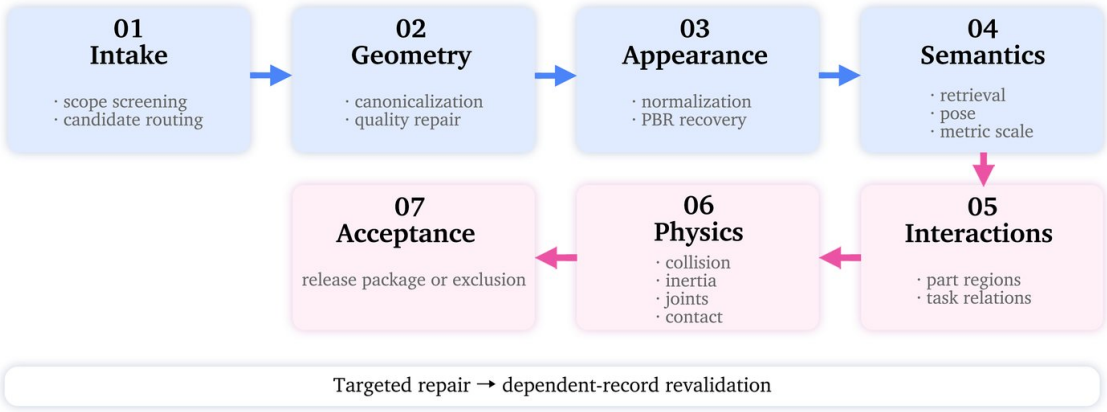


Figure 9 Dependency-ordered simulation-ready asset preparation. Seven stages transform sourced or generated content into a benchmark release package; failed candidates return for targeted repair and dependent-record revalidation.

Geometric canonicalization and quality repair. The pipeline resolves source hierarchy and transforms, normalizes axes and reliable source units, and checks metric extent, topology, component structure, surface orientation, and complexity at mesh and merged-object levels. Repaired assets are rechecked for bounds, origin, orientation, and components. A normalized annotation frame supports visual reasoning without changing physical scale.

Appearance normalization and PBR recovery. The pipeline maps source materials to a common PBR model according to optical behavior while preserving trustworthy maps and scalar values. Missing or degraded appearance first draws from category-conditioned candidates; generative PBR synthesis is used when these are insufficient. Standardized views then verify the whole-object rendering-material record and its asset bindings, which remain separate from physical-material records.

Object-level semantic and metric grounding. Standardized multiview observations support instance description, open-vocabulary retrieval, task-role matching, and semantic orientation. A multimodal annotator combines these views with category context and source metadata, using controlled role vocabularies and free-form text for long-tail distinctions. Candidate rotations relative to a category reference pose establish directional meaning (28). When source scale is unreliable, category- and function-matched GSO, ABO, and YCB anchors condition a relative-scale estimate followed by plausibility checks (31, 35, 38). Semantic pose supplies directional meaning; metric scale controls clearance, placement domains, and physical proxies.

Part semantics and task-conditioned interaction grounding. Object labels alone cannot identify the regions required by contact-rich tasks. PartSAM proposes candidate 3D regions from surface evidence (39); multiview observations, 3D containment, and directional relations reconcile them into semantically consistent parts. The pipeline represents an affordance as a relation among a part, a task, and an admissible interaction—for example, placement, cap removal, or pouring. Unsupported assignments are rejected or reviewed, while interaction records bind contact regions and orientation requirements to scene and task roles so one asset can expose different regions for grasping, opening, pouring, or placement.

Physical, kinematic, and contact-material authoring. Collision construction first tests analytic boxes, spheres, and capsules, followed by a convex hull; V-HACD or CoACD decomposition handles task-relevant concavities

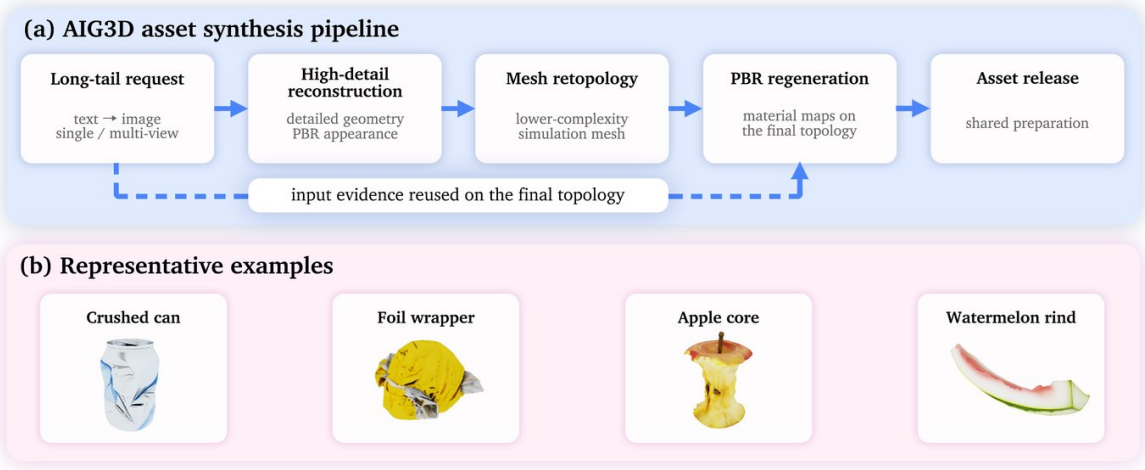


Figure 10 Coverage-directed AIG3D asset synthesis. Text or image evidence conditions detailed reconstruction, simulation-oriented retopology, and PBR regeneration on the final mesh; four examples show the resulting long-tail geometry and appearance.

that require multiple parts (40, 41). Mass and inertia follow a confidence-ranked hierarchy from reliable watertight volume to convex- or bounding-volume estimates. For articulated assets, link–joint topology, frames, signed axes, limits, collision shapes, inertials, and drives are normalized in one object frame. Visual PBR evidence, multiview appearance, category, and part semantics determine a coarse physical-material class. Simulator-specific contact-pair priors then assign physical properties and bind the resulting record to the corresponding collision parts.

Role-specific simulator acceptance and targeted repair. A package enters the benchmark release after the simulator exercises the interactions required by its declared role. Common checks cover metric extent, render–collision alignment, collider validity, and settling. Manipulable rigid objects are grasped, transported, released, and checked after placement; articulated objects and fixtures additionally exercise authored joints, limits, collision behavior, and task-relevant contacts. The same role-specific acceptance procedure applies to sourced and AIG3D candidates. Failed records are repaired and retested or excluded.

Implementation details for asset conversion, annotation, physical authoring, and acceptance appear in Appendix A.2.

Coverage-directed AIG3D Asset Synthesis This synthesis branch fills long-tail gaps in the task-role coverage of sourced collections, including crushed packaging, peels, cores, and refuse with unusual form, state, or appearance. Text requests are converted into reference images, while single- or multiview observations can condition reconstruction directly. AIG3D reconstructs high-detail geometry and PBR appearance before simulation-oriented remeshing and retopology, then reuses the input evidence to regenerate PBR materials on the final mesh. This ordering preserves silhouette and local structure before imposing the simulation mesh budget.

Generated outputs complete the same preparation and acceptance path as sourced content. Accepted additions join the tabletop rigid release and become selectable for scene compilation (Section 3.2.2). Figure 10 summarizes the synthesis path and examples; implementation details appear in Appendix A.2.

3.2.2 Semantic Scene Specification and Construction

A world-space pose that is valid for one tabletop or receptacle can become invalid when the support is resized or the object is replaced. A fixed transform does not preserve which surface supports the object, which volume contains it, or which spatial relation the task requires. Scene definitions must therefore preserve this intent as assets, supports, receptacles, backgrounds, and scales change.

The *scene specification* defines the abstract semantic model, and a *semantic scene program* is its concrete authoring artifact. The *X2Real scene compiler* resolves the program against selected geometry and produces either a *compiled scene* satisfying the declared static geometric and relational constraints or an infeasibility diagnostic. Figure 11 traces the program from authoring to the compiled scene.

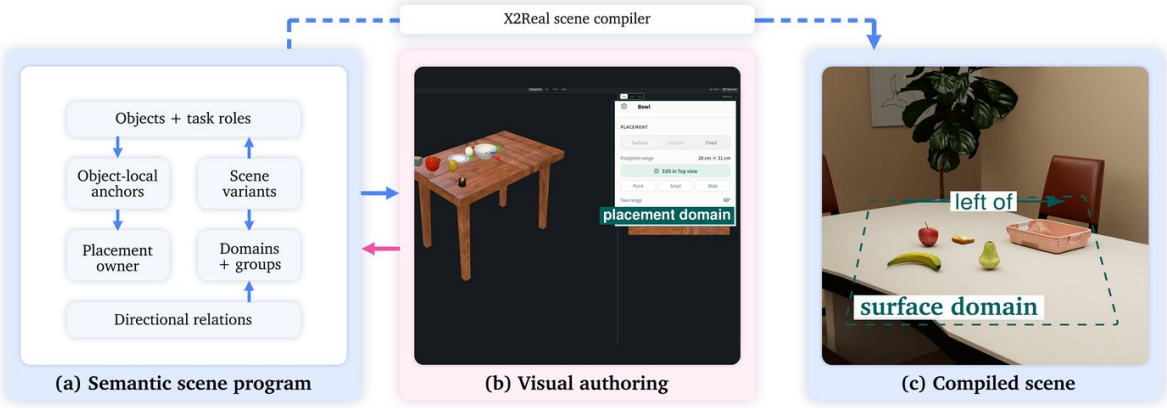


Figure 11 Semantic scene program and compilation. A semantic scene program can be edited visually and compiled against selected geometry into a concrete scene.

The scene program separates object-local geometry, placement ownership, grouped composition, cross-domain relations, and geometry-changing alternatives because each contributes a different constraint to compilation.

Scene Program Constructs Together, these constructs encode task intent in a form that the compiler can resolve against selected geometry.

Objects and object-local anchors define the scene's reusable entities and reference frames. Each background or object has a stable scene identity, task role, and binding to either a fixed asset or an authored, role-compatible candidate set. Object-local surfaces and volumes expose usable geometry such as tabletops, shelves, trays, drawers, and receptacle interiors, so dependent placements resolve in the support's current local frame. An instance may refine scale and semantic tags without changing the asset package.

Placement ownership assigns each placed node exactly one owner—the node itself or a joint group—to prevent conflicting poses. Surface, volume, and discrete pose domains encode admissible planar, contained, or complete task-specific configurations rather than already sampled poses.

Grouped composition represents multi-object arrangements that cannot be reduced to independent placements. A `region` distributes objects over a shared support, `scatter` represents denser or repeated collections, `volume` composes objects inside a receptacle, and `slot` binds nodes to authored locations; the group owns the correlated placement decision.

Spatial relations express directional constraints across placement owners, separate from support and containment. Relations such as `left_of`, `right_of`, `front_of`, and `behind` retain their meaning as support sizes and object footprints change.

Scene variants declare asset, background, and slot alternatives within one program. Geometry-changing variants are resolved before placement because their footprints, support fit, and collision shapes alter the admissible domains; only compatible alternatives are carried into runtime-plan construction.

Geometry-aware Compilation The compiler turns these declarations into concrete placements through two complementary mechanisms: dependency analysis determines coupling and solve order, while transactional solving realizes a consistent assignment.

Placement coupling and dependency analysis constructs a typed interaction graph over node and group domains. Footprint-expanded overlap edges define jointly solved components because one placement can

remove space from another. Directional relations remain explicit constraints and scheduling dependencies without merging independent components, while object-local support and containment references add parent–child dependencies and yield an owner-first schedule.

Transactional component solving first evaluates each coupled component through a temporary assignment. Forward checking rejects choices that empty dependent domains, while incremental validation tests activated support, containment, group, directional, collision, and clearance constraints. The compiler commits a complete assignment atomically; otherwise it rolls back all provisional poses and explores another alternative through bounded backtracking. Final validation precedes export, and exhausted search returns a diagnostic tied to the unsatisfied constraint and placement context.

Visual and Language-guided Authoring The visual editor and language-guided workflow both edit the same semantic scene program and invoke the same compiler. The editor exposes anchors, domains, groups, and relations in perspective and top-down views while previewing the compiled scene (Fig. 11(b,c)). Following prior language-guided scene-construction systems (42, 43), the X2Real authoring agent inspects asset metadata, visual previews, USD geometry, renders from selected cameras, and compiler diagnostics, then uses this evidence to revise asset selection, ownership, domains, groups, or relations.

3.2.3 Task-conditioned Runtime Variation and State-aligned Replay

X2Real supports two forms of content variation with different preservation boundaries. Reset-time variation samples compiled alternatives while preserving authored task structure, whereas replay recomposes static context while preserving recorded states and timestamps. The reset-time path follows a compile-once, sample-many principle: offline compilation resolves admissible alternatives, and each episode reset samples from the resulting plan. Figure 12 summarizes this path.

For episode reset, the compiler emits a `RuntimeSamplingPlan` containing coupled discrete layouts, continuous object-local surface and volume domains, allowed orientations, dependency order, and compatible asset, paired-material, and lighting variants. Coupled layouts capture cross-object choices that must be selected jointly, while local domains preserve residual pose freedom within each choice.

The compiler populates the plan by generating candidates from concrete geometry and feasible domains and coupling the choices that interact (Section 3.2.2). Within each support, a geometry-conditioned generator proposes layouts for placement domains whose feasible regions overlap. The compiler rejects static-constraint violations, removes geometric duplicates, and retains a compact pool balancing geometric quality and layout diversity. Overlapping footprint-expanded domains form joint case pools, object-local support dependencies define parent-before-child sampling, and task-specific restrictions filter or replace scene-level alternatives.

At reset, the runtime realizes the plan by selecting a coupled layout, sampling continuous poses in dependency order, and applying the associated asset, paired-material, and lighting variants. Geometry-changing choices are resolved during plan construction. Appearance adjustments modify bounded rendering channels while preserving the physical-material record. Paired substitutions atomically select authored rendering and physical-material records and update applicable mass, inertia, friction, and support-contact properties.

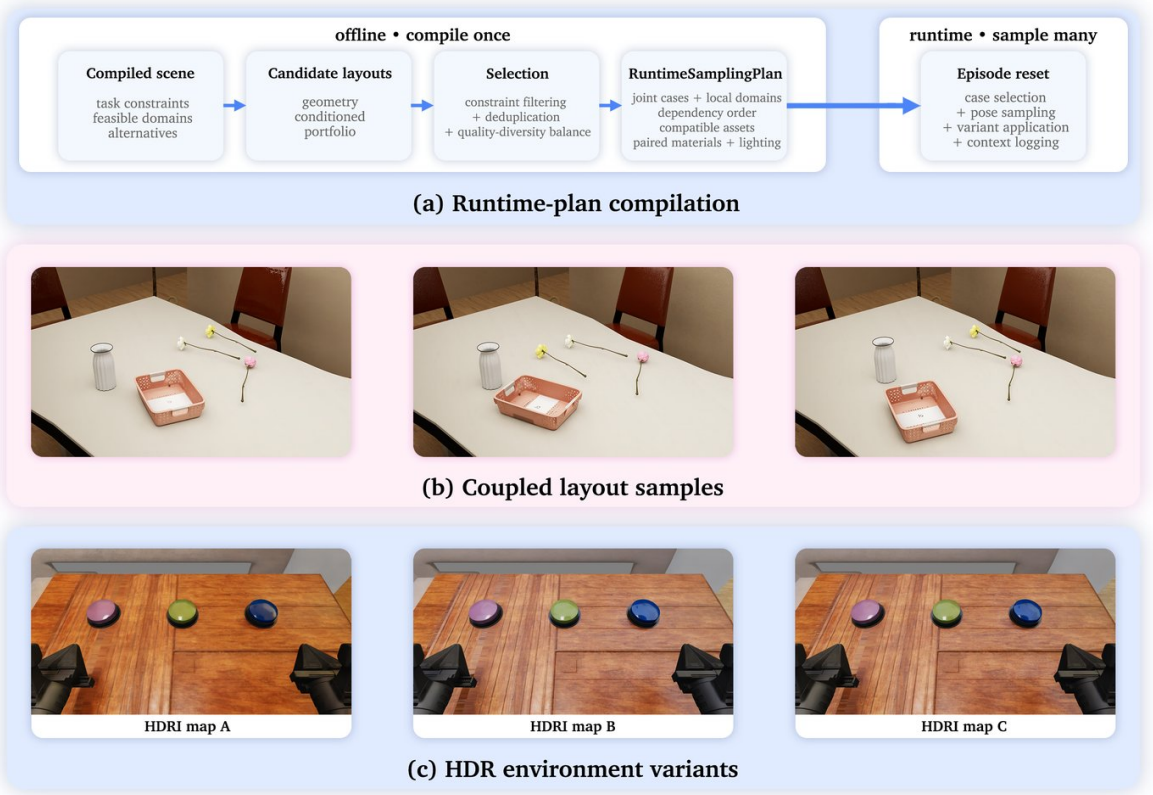


Figure 12 Runtime variation compilation and sampling. The compiler consolidates feasible alternatives into a reset-time plan; examples show coupled layouts and environment-lighting variation.

Swept-volume-aware Replay Augmentation A recorded episode supplies robot and task-object states, benchmark-camera transforms, and timestamps, while a selected scene variant supplies alternative static assets, appearance, and lighting. Replay uses these inputs to recompose observations for training-data augmentation (44).

Forward kinematics reconstructs robot-link motion, while recorded poses provide the task-object trajectories. Geometric proxies, adaptive temporal sampling, and error-bounded path compression capture the trajectory where endpoint interpolation is insufficient. Inflating these paths by spatial extent, approximation error, and clearance yields a swept exclusion envelope. Inserted objects must lie on valid support, outside the envelope and existing obstacles, and satisfy pairwise clearance; stable orientations and support offsets are used when available.

The procedure composes the recorded robot, task-object, and camera transforms with the new static scene layer at their original timestamps. This preserves kinematic state–time alignment while changing the observation context for training-data augmentation. Figure 13 illustrates the recomposition; implementation details for runtime-plan construction and replay geometry appear in Appendix A.3.

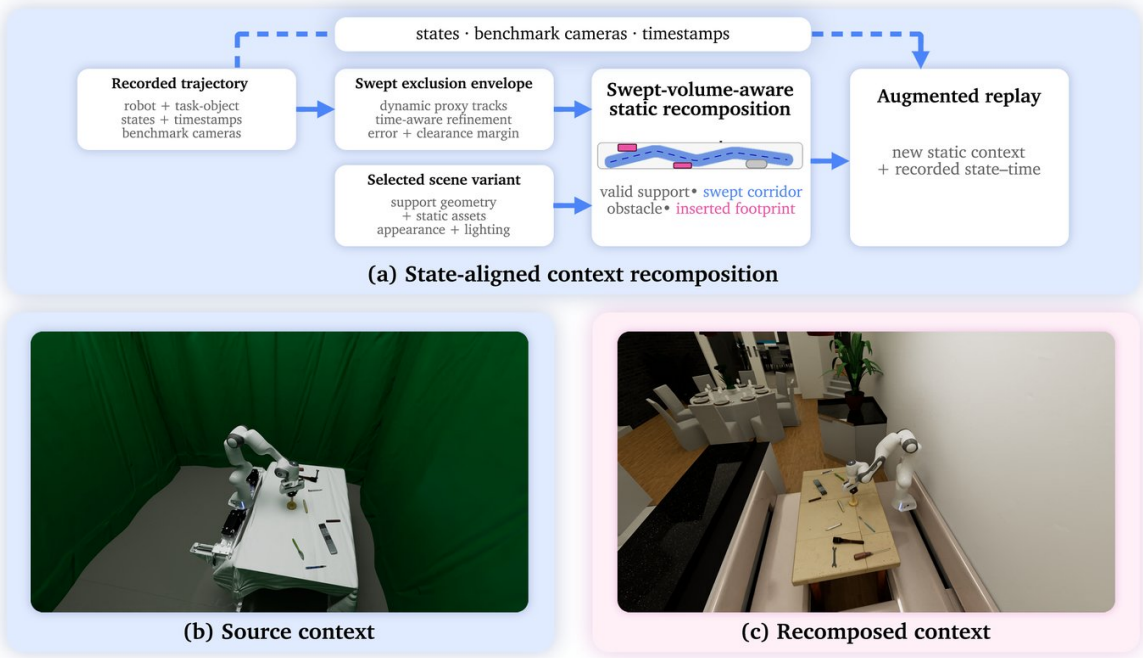


Figure 13 State-aligned replay augmentation. Recorded motion defines a swept envelope for static-context recomposition while preserving the state-time stream; panels (b) and (c) show source and recomposed views from the same recorded instant.

3.3 Atomic Task & Skill Composition

Robot manipulation tasks can typically be decomposed into multiple stages with explicit semantics and dependency relations. For example, placing a medicine bottle in a drawer and then closing the drawer involves several interdependent stages, including opening the drawer, picking up the bottle, placing it inside, and closing the drawer. However, conventional task definitions and evaluation protocols usually provide only a final success signal for the entire task, without a unified description of the intermediate process. This makes it difficult to determine which operations a policy has completed, how far it has progressed, or at which stage it failed. Collected demonstration trajectories also lack directly usable stage boundaries and subtask labels. Although these annotations can be obtained through manual frame-wise labeling or post-hoc parsing with an auxiliary model, both approaches are costly and make it difficult to maintain consistent and verifiable process semantics across tasks. In addition, automatic demonstration collection uses atomic skills such as picking, placing, and pushing to complete stage-level subtasks, and composes them according to their dependencies into complete demonstration trajectories.

To address this limitation, we represent each manipulation task as an executable **Directed Acyclic Graph** (DAG) composed of atomic tasks. Each node defines a verifiable atomic task, i.e., a local physical outcome that the robot must achieve, while each edge specifies a dependency between atomic tasks. The same task graph enables the system to record stage-completion events online and generate subtask labels for teleoperated demonstrations. It also allows DAG nodes to be paired with atomic skills for motion-planning-based demonstration synthesis, with the generated outcomes verified by node success conditions. During evaluation, the graph further provides subtask completion, task progress, and failure diagnostics beyond final task success.

3.3.1 Executable Task DAG

We represent a manipulation task as a directed acyclic graph $G = (V, E)$. Each node $v \in V$ defines an atomic task and is associated with a simulator-state-based success predicate $c_v(s_t)$, which evaluates physical conditions such as object poses, contact relations, grasp states, or joint states. Each edge $(u, v) \in E$ indicates that node v

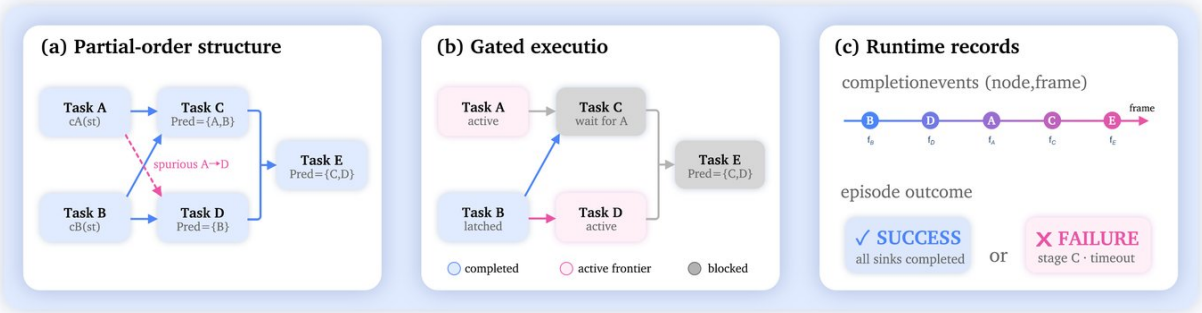


Figure 14 Executable Task DAG. (a) The Task DAG represents only the necessary partial-order dependencies and does not impose an order on independent nodes. (b) At runtime, a node becomes active only after all of its predecessors have completed, and its completion state is latched when its success predicate is first satisfied. (c) Node-completion events record the node ID and first completion frame, supporting task-success determination, progress measurement, and failure attribution. The success and failure records illustrate two alternative episode outcomes rather than simultaneous results from the same execution.

depends on the completion of node u. Unlike a fixed step list, a DAG specifies only the necessary precedence constraints and imposes no order between independent stages. As illustrated in Fig. 14(a), nodes A and B can be completed independently; node C waits for both A and B, whereas node D depends only on B; node E becomes active only after both C and D are complete. Consequently, when B is complete but A is not, D can already be executed while C must still wait for A. A fixed linear sequence would impose unnecessary ordering constraints. Recursively organizing the same task with only sequential and parallel stages would likewise require either introducing an additional $A \rightarrow D$ constraint or duplicating the shared predecessor B. A DAG represents these dependencies directly.

During execution, a node success predicate is not evaluated unconditionally. The system activates a node and evaluates its predicate only after all of its predecessors have completed. Let d_v^t indicate whether node v has been completed by time t. Its gate and latched completion state are defined as

$$g_v^t = \bigwedge_{u \in \mathrm{Pred}(v)} d_u^t, \qquad d_v^t = d_v^{t-1} \vee \left(g_v^t \wedge c_v(s_t)\right). \tag{1}$$

Here, g_v^t indicates whether the dependency conditions of node v have been satisfied. At time t, the unfinished nodes with open gates form the active node set, and the system evaluates their success predicates in parallel at every simulation step. The set of nodes newly completed at the current frame is

$$\Delta D_t = \left\{v \in V \mid \neg d_v^{t-1} \wedge g_v^t \wedge c_v(s_t)\right\}. \tag{2}$$

For every $v \in \Delta D_t$, the system emits a node-completion event (v, t). In Fig. 14(b), when B is complete but A is not, D becomes active because it depends only on B, whereas C continues to wait for A. Gating prevents a downstream stage from being marked complete merely because its predicate happens to hold in the initial state. Once a node is completed, its state is latched, converting a transient physical condition into a persistent stage milestone.

Completing an individual node indicates only that one stage objective has been achieved and does not terminate the episode by itself. After a node is completed, the system activates its successors according to the graph dependencies. The overall task is considered successful only when all sink nodes have completed and no stage-level failure has been triggered:

$$S^t = \neg F^t \wedge \bigwedge_{v \in \mathrm{Sink}(G)} d_v^t, \tag{3}$$

where F^t denotes a node failure predicate, stage timeout, or another task-level failure condition. Because a sink node can become active only after all of its predecessors have completed, this criterion also guarantees completion of the dependency stages leading to the final objectives.

The Task DAG further converts changes in node state into recordable runtime events. When a latched node state first changes from incomplete to complete, the system records the node ID and first completion frame. The sequence $B \rightarrow D \rightarrow A \rightarrow C \rightarrow E$ in Fig. 14(c) is one valid runtime completion order induced by the same DAG, rather than a fixed sequence prescribed by the task. When a node failure predicate or stage timeout is triggered, the system additionally records the first failed node, the failure frame, and the corresponding reason. Each episode therefore contains not only a final success signal, but also node-level completion states, first completion frames, the failed stage, and the failure reason. These runtime records are subsequently used for stage-level demonstration annotation and fine-grained policy evaluation.

3.3.2 DAG-guided Demonstration Collection

X2Real supports two sources of Task-DAG-guided demonstrations: real-time teleoperation and atomic-skill-based automatic synthesis. The two collection pipelines share the same scene generation, task definitions, DAG runtime, evaluation module, trajectory recorder, and exported data format. Their primary difference lies in how control trajectories are produced. The teleoperation pipeline maps control signals from a teleoperation device to robot control targets, whereas the automatic pipeline uses atomic skills bound to DAG nodes to generate targets and a motion planner to solve the corresponding execution trajectories. Consequently, both sources produce demonstrations with consistent observation–action structures, stage semantics, and node-level evaluation results.

Teleoperated Demonstrations X2Real provides a unified teleoperation interface supporting physical master arms, VR controllers, and keyboard input. The human demonstrations in the benchmark are collected through physical master–slave teleoperation. An operator uses the physical master arms to control a simulated dual-arm robot. The collection loop receives master-device states asynchronously and sends dual-arm commands to the simulated robot at a control frequency of 30 Hz while providing real-time visual feedback to the operator. Within the same control loop, the system synchronously records camera observations, robot states, actions, and DAG runtime states, producing complete trajectories aligned by simulation step.

During trajectory execution, the DAG runtime uses privileged simulator state to evaluate the success predicates of the currently active nodes online. When the completion state of node v first changes from false to true, the system records the stage-completion event

$$e_v = (v, f_v), \qquad f_v = \min\{t \mid d_v^t = 1\}, \tag{4}$$

where f_v is the first completion frame of node v. Let $\pi_\tau = (v_1, \ldots, v_K)$ denote the temporally ordered node-completion events observed in a demonstration, and let $f_{v_0} = 0$. For a trajectory in which stages are executed sequentially, the segment between the completion of the preceding node and the first completion of node v_k is assigned the subtask label v_k:

$$z_t = v_k, \qquad f_{v_{k-1}} < t \leq f_{v_k}. \tag{5}$$

Each trajectory segment is labeled by the atomic task newly completed at its endpoint and thus reflects the physical outcome actually achieved by that segment. Because the first completion frames are determined online from simulator state and node success predicates, neither manual boundary annotation nor post-hoc parsing with an auxiliary model is required. In the Classify by Shape example in Fig. 15, the three nodes for placing the sphere, cylinder, and cube have no dependencies between them and may therefore be completed in any order, whereas the return-home node waits for all three placement nodes. The Task DAG does not prescribe the placement order; the figure shows one valid order produced by this particular demonstration.

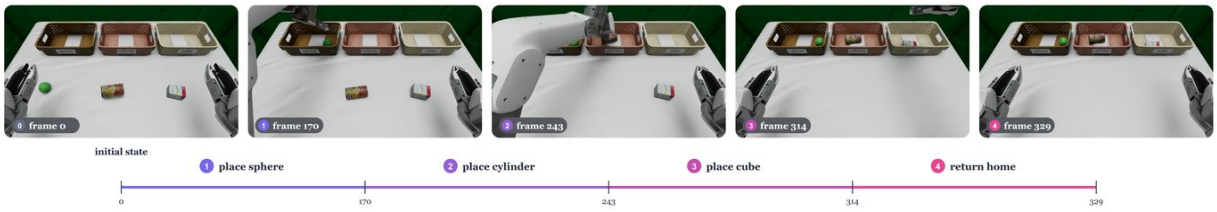


Figure 15 DAG-guided Teleoperation Labeling. The first completion frames of the Task DAG nodes divide a Classify by Shape demonstration into trajectory segments with atomic-task labels.

Automatic Data Synthesis X2Real constructs a closed-loop data-generation process driven by the Task DAG. At runtime, the DAG determines the currently active nodes from task progress. The atomic skill sequence bound to a selected node generates sparse motion targets from the current scene state, and a motion planner expands these targets into an executable trajectory. The node success predicate then independently verifies the resulting physical outcome. Upon successful verification, the system latches the node completion state and activates successor nodes whose dependency conditions have become satisfied. If skill generation, motion planning, or predicate verification fails, the system resamples the target or motion path. Through this closed loop, the same Task DAG can produce demonstrations with different node orders, skill parameters, and motion trajectories while preserving the same task semantics.

Specifically, each node v is associated with an ordered sequence of parameterized atomic skills,

$$\mathcal{K}_v = \left[k_{v,1}(\theta_{v,1}), \ldots, k_{v,m_v}(\theta_{v,m_v})\right], \tag{6}$$

where m_v is the number of skills associated with node v, and $k_{v,i}$ and $\theta_{v,i}$ denote the i-th atomic skill and its parameters, respectively. The node and its success predicate specify the physical outcome to be achieved, i.e., *what* constitutes completion, whereas the skill sequence $\mathcal{K}_v$ describes *how* the robot attempts to achieve that outcome. The relationship is not one-to-one: a node may compose multiple sequential skills, while the same skill can be reused across nodes and tasks through different object roles and parameters. Different target samples and motion plans can also yield multiple valid realizations of the same node.

For node scheduling, the collector computes the active node set from the completed nodes and selects a node v whose skill sequence will be instantiated. Nodes without dependencies between them can be active simultaneously, allowing the collector to choose the next node and thereby produce different valid execution orders. A successor node becomes active only after all of its predecessors have completed.

After selecting a node, the collector instantiates the atomic skills in $\mathcal{K}_v$ sequentially. An atomic skill exposes a unified target-generation interface but does not directly solve inverse kinematics or issue low-level control commands. It reads the current robot state together with privileged simulator state for objects, joints, and target regions, and generates a sparse target sequence with stage semantics. Each target may specify an end-effector position and orientation, gripper state, execution phase, arm selection, and optional auxiliary targets. A collision-aware motion planner then expands the sparse targets into a dense joint-control trajectory. Our current implementation uses cuRobo for inverse kinematics and trajectory optimization, after which an executor tracks the planned trajectory in closed loop. For example, `pick` generates targets for approach, descent, gripper closure, and lifting, whereas `place` generates targets for pre-placement, placement, release, and retraction. Different execution phases may use different motion-planning strategies and motion speeds.

Skill parameters are decoupled from specific scenes through semantic roles defined by the task. For example, `@pickable` and `@destination` denote the manipulated object and target region in the current episode, respectively, and are automatically bound to concrete asset instances after scene resampling. The same `pick` and `place` skills can therefore be reused across tasks with different object and target parameters. Within a node, multiple atomic skills can form a sequential skill sequence, such as `pick` → `place` for achieving the outcome that an object lies within a target region. Across the task, the skill sequences associated with different nodes are scheduled according to the Task DAG dependencies to complete the full task.

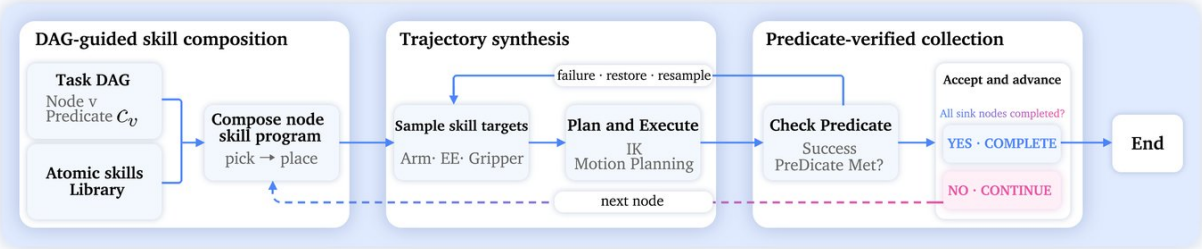


Figure 16 DAG-guided Data Synthesis. The system selects an active node, instantiates its skill sequence to generate sparse targets, and plans and executes the resulting trajectory. The node success predicate independently verifies the outcome. On success, the system latches the node state, saves the trajectory segment and scene checkpoint, and updates the Task DAG; on failure, it restores the latest valid checkpoint and resamples the target or motion path. Once all sink nodes have completed, the stage-wise verified segments form a complete demonstration.

Completing motion planning does not imply completion of the atomic task. After executing a candidate trajectory, the system evaluates the node's own success predicate against the resulting physical state. The node and its trajectory segment are accepted only when the predicate is first satisfied and the completion state is latched. If the target is unreachable, motion planning fails, a collision occurs during execution, or the node predicate remains unsatisfied after execution, the collector records the failed stage and reason and resamples the skill target or motion path. Atomic skills and the motion planner therefore propose candidate behaviors for *how* to perform the task, while the Task DAG independently determines whether the desired physical outcome has actually been achieved. This separation between generation and verification ensures that automatically synthesized demonstrations are validated using the same geometric, contact, and joint-state conditions used in subsequent policy evaluation, rather than being declared successful by the generator itself.

This closed loop also provides behavior-level data augmentation. For the same node, the system can resample scene layouts and object instances, grasp candidates and approach directions, placement poses, arm assignments, skill parameters, and collision-free motion paths. For tasks with multiple simultaneously active nodes, it can also sample different valid node orders. Each sample undergoes a new reachability check and motion-planning process, and its outcome is verified by the same node predicate. The resulting demonstrations may therefore differ in robot states, action sequences, trajectory durations, and intermediate paths while sharing the same atomic-task semantics.

For tasks containing multiple stages, the Task DAG additionally supports stage-wise collection, failure recovery, and trajectory stitching. After a node is completed, the system saves its trajectory segment, node-completion event, and resulting scene checkpoint. If a subsequent node fails, the collector restores the most recent successful checkpoint and resamples and replans only the current failed node, rather than replaying the completed prefix from the initial state. Successful segments retain lineage information such as their node IDs, parent samples, and completion progress. The system finally removes seam frames introduced by checkpoint restoration and stitches the segments according to the actual execution order to form a complete demonstration. This mechanism localizes the cost of failures in later stages and allows a successful prefix to connect to different successor targets and motion paths, yielding traceable failure-recovery and trajectory-recomposition data.

3.3.3 DAG-guided Evaluation

Conventional robot-manipulation evaluation typically reduces an episode to a binary final outcome. Although this metric directly indicates whether a policy completes the entire task, it cannot distinguish between failures that occur before a critical operation begins and those that occur near the end after most stages have been completed. Nor can it identify the specific stage that limits overall performance. During a policy rollout, X2Real executes the same Task DAG used in the task definition and directly derives fine-grained evaluation metrics from node completion states, first completion events, and failure events, without additional manual annotation or post-hoc video parsing. Final task success remains the primary metric, while node-level metrics measure partial progress and diagnose failures.

Final Task Success. For an episode τ, let T_τ denote its termination time. Following the definition in Sec. 3.3.1, the episode is successful only if all sink nodes have completed and no failure condition has been triggered:

$$S_\tau = \neg F_\tau^{T_\tau} \wedge \bigwedge_{v \in \mathrm{Sink}(G_\tau)} d_{v,\tau}^{T_\tau}. \tag{7}$$

Over a set of N evaluation episodes, the final task success rate is

$$\mathrm{SuccessRate} = \frac{1}{N} \sum_{\tau=1}^{N} S_\tau. \tag{8}$$

This metric requires the policy to complete every dependency stage leading to the sink nodes. It therefore cannot be triggered by a final-state condition that happens to hold at initialization, nor does partial node completion cause premature success.

Subtask Completion Rate. Let $\mathcal{I}_v$ denote the set of evaluation episodes that contain node v. The completion rate of subtask v is

$$C_v = \frac{1}{|\mathcal{I}_v|} \sum_{\tau \in \mathcal{I}_v} d_{v,\tau}^{T_\tau}. \tag{9}$$

C_v is the fraction of rollouts in which the policy achieves the physical outcome defined by node v. Compared with reporting only overall success, per-node completion rates reveal which manipulation stages the policy can complete reliably and which nodes constitute the largest performance bottlenecks. We report this metric separately for each node to compare the reliability of different stages within the same task.

Task Progress. X2Real assigns a stage score w_v to every node in a Task DAG such that the scores within each task sum to 10:

$$\sum_{v \in V_\tau} w_v = 10. \tag{10}$$

Node scores are assigned according to each stage's contribution and semantic importance to the overall task objective: nodes representing critical task outcomes receive higher scores, while auxiliary stages receive lower scores. All X2Real tasks use this ten-point node-scoring scheme, providing a common progress scale across tasks with different graph structures and numbers of nodes. For episode τ, the cumulative task score and normalized task progress at time t are defined as

$$Q_\tau^t = \sum_{v \in V_\tau} w_v d_{v,\tau}^t, \qquad P_\tau^t = \frac{Q_\tau^t}{10}. \tag{11}$$

Because node completion states are latched, both the cumulative score Q_τ^t and task progress P_τ^t are monotonically nondecreasing throughout execution. At termination, $Q_\tau^{T_\tau} \in [0, 10]$ provides a partial-completion score for an unsuccessful episode, whereas an episode that completes every node receives a score of 10. The subtask completion rate C_v measures the reliability of a semantic node across multiple rollouts, while P_τ^t describes the important stages completed within an individual rollout and their cumulative contribution at a given time.

Figure 17 Physics-render scheduling in Mana. (a) *sync/sync* serializes physics and rendering. (b) *sync/async* overlaps $R(S_{n+1})$ with main-thread work. (c) *async/async* overlaps $R(S_n)$ with physics advancing to S_{n+1}; camera observations use S_n, while state observations and evaluation use S_{n+1}.

Failure Case Analysis. For an unsuccessful episode, the DAG runtime records the node associated with the first failure event, its frame index, and the corresponding reason. Failure reasons arise from node-level failure predicates, stage timeouts, or other task-level failure conditions. If an episode terminates because of a task-level timeout without emitting a node-level failure event, the system additionally stores the set of currently active nodes that remain incomplete at termination. These records support a two-dimensional distribution over failure node and failure reason. Together with first completion frames, they distinguish episodes that never reached a stage, reached but failed to complete it, or completed it before failing at a successor stage.

3.4 Efficiency Optimization

Built on the Isaac ecosystem, Mana treats each environment step as a heterogeneous CPU–GPU pipeline. Isaac Lab's default manager-based reinforcement-learning environment serializes action processing, the decimated physics loop, rendering, observation, and evaluation. Within each of the N physics substeps, actuator computation precedes simulation. Mana controls physics stepping and rendering independently, yielding the three execution modes in Fig. 17. The *async/async* mode is the default Mana configuration, while *sync/sync* and *sync/async* provide controlled baselines.

Optimized Camera Pipeline. All three scheduling modes use the same optimized camera pipeline. It tracks camera updates on the CPU, stores rendered outputs in CUDA double buffers, and moves image layout conversion and post-processing to a dedicated GPU stream. This design reduces per-frame CPU–GPU synchronization. In a controlled *sync/sync* ablation with the same environment and camera configuration, it reduced mean latency by 28.41% relative to Isaac Lab's default camera pipeline.

CPU Physics. For rigid-body manipulation, Mana places physics and actuator computation on the CPU. This profile enables continuous collision detection (CCD) to reduce high-speed tunneling and improve collision robustness. It also reserves the GPU for multi-camera rendering, allowing physics and rendering to overlap without competing for GPU resources. The dual-asynchronous path is currently validated only for CPU PhysX rigid-body workloads; deformable bodies, cloth, and particles remain outside its scope.

Physics-render Scheduling. The *sync/sync* baseline completes the N actuator–simulation substeps, renders the resulting state S_{n+1}, executes the remaining main-thread work, and finally evaluates and records observations. The *sync/async* mode keeps physics synchronous but submits $R(S_{n+1})$ asynchronously, allowing rendering to overlap with main-thread work. The environment waits for both paths before returning, so camera and state observations correspond to S_{n+1}.

The dual-asynchronous design extends Mana's earlier two-thread CPU-physics/rendering pipeline with an explicit frozen-state boundary. In *async/async*, a persistent CPU worker executes the complete N-substep actuator and physics sequence. In parallel, the renderer consumes a frozen snapshot S_n, and the main thread performs independent work. At the synchronization boundary, the caller joins the worker and renderer, consumes and records the S_n camera result, publishes S_{n+1}, and completes state observation, evaluation, and post-step recording. Camera observations therefore incur a deliberate one-step delay, whereas state observations and evaluation use the current state; no rendering work crosses the environment-step boundary.

Performance. Let $T_{\text{physics}} = \sum_{i=1}^{N}(T_{\text{actuator},i} + T_{\text{simulation},i})$ denote the complete CPU physics sequence, and let T_{render}, T_{main}, and T_{obs} denote rendering, main-thread work, and final observation/evaluation. The critical paths of *sync/sync*, *sync/async*, and *async/async*, denoted T_A, T_B, and T_C, are approximated by

$$
\begin{aligned}
T_A &\approx T_{\text{physics}} + T_{\text{render}} + T_{\text{main}} + T_{\text{obs}}, \\
T_B &\approx T_{\text{physics}} + \max(T_{\text{render}}, T_{\text{main}}) + T_{\text{obs}} + T_{\text{sync},B}, \\
T_C &\approx \max\left(T_{\text{physics}}, T_{\text{render}}, T_{\text{main}}\right) + T_{\text{obs}} + T_{\text{sync},C},
\end{aligned}
$$

where $T_{\text{sync},B}$ and $T_{\text{sync},C}$ denote mode-specific submission and synchronization overheads. These expressions summarize the overlap structure; all reported latencies are measured end to end.

We evaluate Mana's three scheduling modes on the same workstation (AMD Ryzen 9 9950X CPU and NVIDIA GeForce RTX 5090 GPU), using the same seed and three cameras: two 640×480 wrist cameras and one 1280×720 head camera. Over one complete EX001-G episode from the X2Real benchmark, the mean per-step latencies of *sync/sync*, *sync/async*, and *async/async* are 46.79, 24.68, and 22.43 ms, respectively. Relative to *sync/sync*, the two asynchronous modes reduce latency by 47.24% and 52.06%; *async/async* further reduces it by 9.13% relative to *sync/async*. For tasks that tolerate the one-step camera delay, these controlled single-episode results indicate that the default dual-asynchronous schedule improves data-collection and policy-evaluation efficiency without changing the physics time step or control frequency; they do not estimate cross-task throughput.

4 Experiments and Results

4.1 Sim2real Alignment

4.1.1 System Identification

To reduce the discrepancy in end-effector tracking behavior between the real and simulated robots, we calibrate the control parameters of the simulator by matching end-effector trajectories. Real-robot trajectories are collected during teleoperation and form a dataset. The corresponding end-effector pose commands are replayed in simulation in a time-synchronized manner, converted into joint targets via inverse kinematics, and executed using the Stable Proportional–Derivative (StablePD) controller (45). The proportional gain K_p and derivative gain K_d are shared across all 12 bimanual arm joints.

We optimize the shared gains using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (46). The collected trajectories are partitioned into calibration, validation, and test sets for parameter search, model selection, and final evaluation, respectively. Given a calibration set of n trajectories, we solve:

$$
\left(K_p^*, K_d^*\right) = \arg\min_{K_p, K_d} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i. \tag{12}
$$

The per-trajectory loss combines average and tail tracking errors:

$$\mathcal{L}_i = w_p \,\mathrm{RMSE}(e_{p,i}) + w_R \,\mathrm{RMSE}(e_{R,i}) + w_{p,95}\, P_{95}(e_{p,i}) + w_{R,95}\, P_{95}(e_{R,i}), \tag{13}$$

where w_p, w_R weight position and orientation root mean square error (RMSE), and $w_{p,95}, w_{R,95}$ weight their 95th-percentile errors. The percentile terms explicitly penalize intermittent large deviations that are not well captured by RMSE. The frame-wise error signals are defined as

$$e_{p,i} = 10^3 \|\mathbf{p}^{\mathrm{real}} - \mathbf{p}^{\mathrm{sim}}\|_2, \quad e_{R,i} = \frac{180}{\pi} \left\| \left(\mathrm{Log}\left((\mathbf{R}^{\mathrm{real}})^\top \mathbf{R}^{\mathrm{sim}} \right) \right)^\vee \right\|_2, \tag{14}$$

where $\mathbf{p}$ and $\mathbf{R}$ are end-effector position and orientation, and $\mathrm{Log}(\cdot)^\vee$ maps relative rotations to axis-angle vectors (47). The scaling factors 10^3 and $180/\pi$ convert meters to millimeters and radians to degrees, respectively. RMSE and P_{95} are computed over all frames and both arms. We set $(w_p, w_R, w_{p,95}, w_{R,95}) = (1, 5, 0.1, 0.5)$ in all experiments. CMA-ES optimizes in the log-parameter space with $K_p \in [100, 10\,000]$ and $K_d \in [5, 500]$. Since the measured input–response delays of the real and simulated systems are comparable, trajectories are compared at synchronized frame indices without temporal alignment or delay compensation. Figure 18 shows representative tracking on a held-out test trajectory.

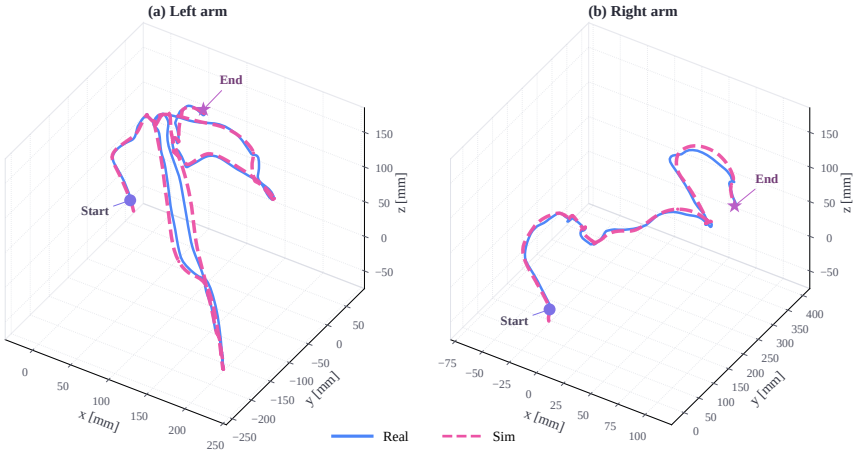


Figure 18 Sim2real Trajectory Alignment. Representative real and simulated end-effector trajectories on a held-out test trajectory.

A similar gain-only real-to-sim system identification approach is also adopted in RobotArena∞ (48). Broader system-identification studies motivate a staged extension to effective actuator parameters such as armature, damping, and friction (49, 50, 51), while avoiding the simultaneous optimization of structurally confounded quantities (51). However, gains that minimize trajectory-matching error may not be optimal for downstream manipulation (50): contact-rich tasks couple actuator dynamics with object properties, compliance, impacts, and unobserved forces. Task-relevant excitation and force sensing are therefore promising directions for manipulation-oriented real-to-sim system identification.

4.1.2 Sim2real Correlation

To verify that our realistic simulation environment can serve as a faithful proxy for real-world evaluation, we replicate the green booth scene from ManipArena (8) in simulation by manually aligning all the setting like light sources, assets, and embodiments. As shown in Fig. 19, we choose 8 tasks from ManipArena that mainly contain rigid object interactions. The first 5 tasks are general manipulation tasks, including putting spoons into the bowl, putting three blocks onto matching colors, placing all items into the basket, pressing the buttons in given sequence, selecting all the fruits and placing them into the basket. The last three are harder manipulation tasks requiring control precision, including putting the glasses on the wood shelf by bimanual coordination, sliding the ring through the rod, stacking three cups into a triangle tower.

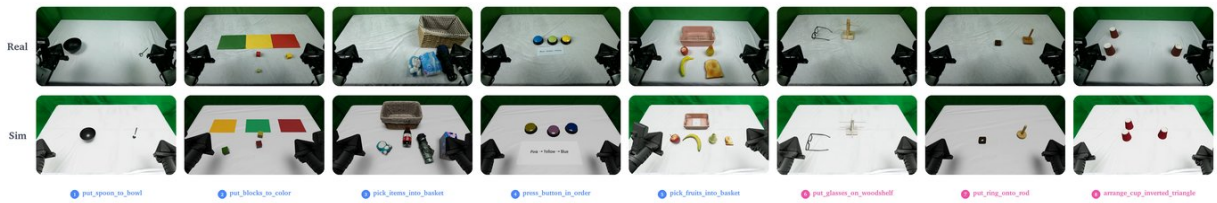


Figure 19 Sim2real Tasks. We create digital twins (bottom row) of 8 tasks from ManipArena (8) (top row).

We select the wall-oss-0.5 model as our base model for its best performance on ManipArena. After finetuning it with real demonstration data, we directly evaluate it in our simulation environment, and compare the results with real evaluation depicted in Fig. 20. Note that zero simulation data is used in the pretraining and finetuning process.

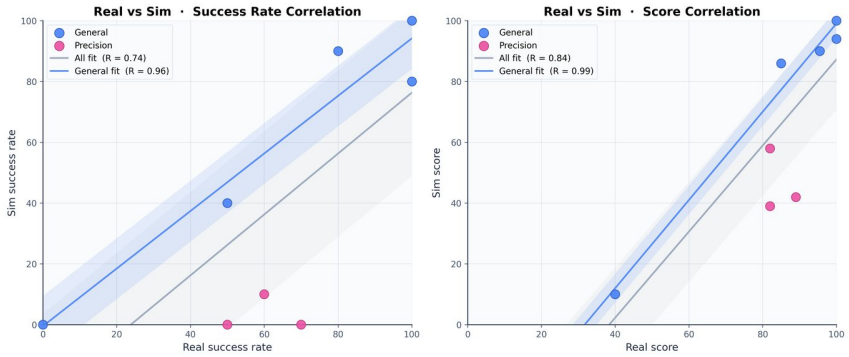
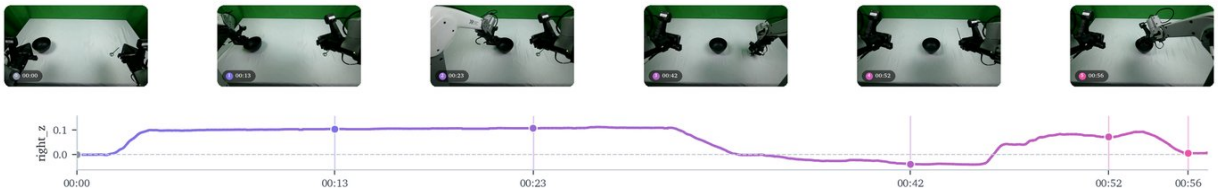


Figure 20 Sim2real Correlation. Wall-oss-0.5 trained on zero simulation data achieves high sim2real linear correlation in success rate and progress score.

From Fig. 20, we observe a strong correlation between simulation evaluation and real evaluation. For general manipulation task only, we achieve 0.96 linear correlation coefficient for success rate and 0.99 for progress score. However the policy struggles on three precision tasks in simulation with low success rates. Taking all tasks into account, our simulation environment achieves 0.74 linear correlation coefficient in success rate and 0.84 in progress score. Taking a closer look at each evaluation episode we can better understand why this gap appears.

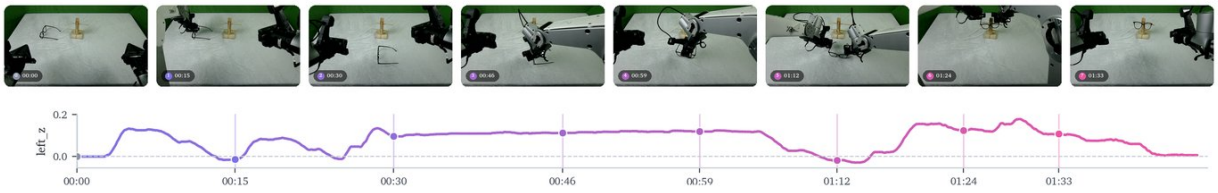
As shown in Fig. 21, we show example real and simulation evaluation trajectories for two tasks: putting the spoon into the bowl and putting glasses onto the wood shelf. For the first task, the policy achieve high success rate in both real world (100%) and simulation (80%), but in different ways. In real world, the robot can smoothly pick up the bowl and put it onto the center of the table, then pick and place the spoon in the first try. However in simulation, after it successfully place the bowl, the robot try for 10 times before it finally pick up the spoon, which can be visualized both from the camera and from the height of right gripper. This phenomenon does not appear solemnly but in nearly every episode. For the glasses-on-wood-shelf task, the policy has a 70% success rate in reality, but 0% in simulation. From the camera and the height trajectory of the left gripper in simulation, the robot still tries to finish the task through multiple attempts, but once the robot fail to properly place the glasses, the legs of the glasses might be folded, resulting into an unprecedented out-of-distribution scenario that the policy cannot handle. For other precision tasks, the failure case is similar, the policy can no longer achieve high success rate through multiple re-tries.

These evidences prove that our simulation environment is adequate for evaluating real robot's performance, given the total score correlation 0.84. But there still exist a real2sim gap due to the simulator, controller, or rendering, that makes real-world policies can't work perfectly in simulation as in reality.

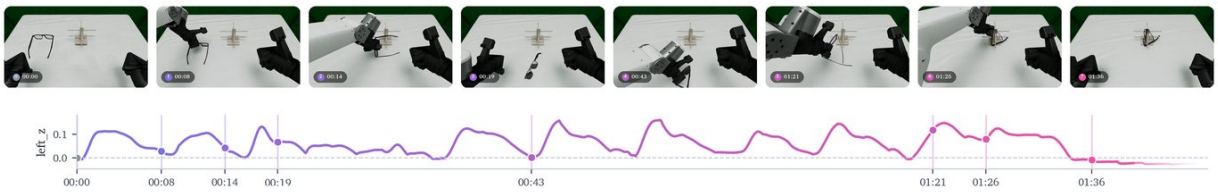


(a) The real success trajectory of putting a spoon in a bowl with statistics of right gripper height.

(b) The simulated success trajectory of putting a spoon in a bowl with statistics of right gripper height.



(c) The real success trajectory of putting glasses on the shelf with statistics of left gripper height.



(d) The simulated failure trajectory of putting glasses on the shelf with statistics of left gripper height.

Figure 21 Sim2real Trajectory Analysis. The policy trained on real data can overcome sim2real gap through multiple attempts in general manipulation tasks, but deteriorate in precision tasks due to the high failure penalty.

4.2 Model Evaluation

4.2.1 Tabletop Manipulation

We evaluate four generalist robot policies on all tabletop manipulation tasks in X2Real: Wall-OSS-0.5 (4), DreamZero (3), $\pi_{0.5}$ (2), and our internal model Wall-x-preview. All models are post-trained using the same task demonstrations collected on the ArtiXon Arm-6A embodiment and are evaluated through the unified Policy Space protocol described above. To ensure a controlled comparison, each model is fine-tuned for approximately one epoch using only the raw observations, actions, and language instructions, without access to the frame-level subtask annotations used by the evaluator. We otherwise follow the native training and action representations of each policy:

1. **Wall-OSS-0.5**: EEF control, batch size 256, 20k iterations.
2. **DreamZero**: joint-space control, batch size 64, 64k iterations.
3. $\pi_{0.5}$: EEF control, batch size 64, 80k iterations.
4. **Wall-x-preview**: EEF control, batch size 144, 25k iterations.

The complete results under both in-distribution (ID) and out-of-distribution (OOD) evaluation settings are reported in Tab. 1.

For each task, we report both the success rate and the progress score, with the latter measuring how far a policy progresses through the task dependency graph even when the full task is not completed. To avoid over-weighting capability dimensions containing more tasks, the final benchmark score is computed using hierarchical macro-averaging. We first average episode-level scores within each task, then average equally across tasks within each subcategory, across subcategories within each major category, and finally across the Reasoning and Manipulation categories. The same aggregation procedure is applied to success rates. Consequently, each capability dimension contributes equally to the final result regardless of the number of tasks it contains, as shown in Eq. 15.

$$\mathrm{SR}(\mathrm{Score}) = 100 \times \operatorname*{Avg}_{\text{Reasoning/Manipulation}} \left[\operatorname*{Avg}_{\text{List}} \left[\operatorname*{Avg}_{\text{Task}} \left[\operatorname*{Avg}_{\text{Setting}} \left[\operatorname*{Avg}_{\text{Episode}} \left(\mathrm{SR}(\mathrm{Score}) \right) \right] \right] \right] \right]. \tag{15}$$

Several observations emerge from the benchmark results. First, **OOD evaluation consistently exposes a substantial generalization gap**. All four policies obtain lower aggregate performance under OOD randomization than under ID evaluation. For example, Wall-x-preview drops from an overall success rate of 53.9% to 34.3%, while its progress score decreases from 67.3 to 56.6. Similar degradation is observed across the other policies, although the magnitude varies substantially across individual tasks. This suggests that performance measured under training-like configurations can considerably overestimate robustness to unseen deployment conditions.

Second, **success rate alone does not fully characterize policy capability**. Policies frequently make meaningful progress without completing the entire long-horizon task. For example, DreamZero achieves only 4.85% overall ID success rate but obtains a substantially higher progress score of 34.21. This discrepancy is particularly common on long-horizon and precision-sensitive tasks, where failure at a late stage turns an otherwise largely correct trajectory into a binary failure. The progress metric therefore provides complementary information about where a policy fails along the task execution process.

Third, the results reveal several **shared capability bottlenecks across model families**, with a particularly clear gap between partial task progress and successful completion. On many challenging tasks, policies obtain non-trivial DAG scores despite very low success rates. For example, contact-rich manipulation tasks such as `ring_to_rod`, `plug_in_charger`, and `screw` often exhibit near-zero success while still achieving measurable intermediate progress. This suggests that current policies can frequently infer plausible action sequences and complete early subtasks, but struggle to reliably execute the final, precision-sensitive stages of long-horizon behaviors. A second bottleneck appears in tasks requiring tightly coupled perception, reasoning, and action, such as `adjust_temperature`, `alphabet_word`, and several memory-tracking tasks, where performance degrades substantially under OOD conditions. In contrast, much stronger results are observed on coarse object-centric manipulation and familiar semantic compositions, including `put_glasses_on_woodshelf`, `sweep_trash`, and several language reasoning tasks under ID settings. Taken together, these results indicate that current generalist policies are often capable of producing partially correct behavior, but remain substantially less reliable at converting such progress into successful task completion when precise contact control, persistent state tracking, or closed-loop adaptation is required.

4.2.2 Mobile Operation

We evaluate our internal model Wall-x-preview and $\pi_{0.5}$ (2) on three Mobile Operation tasks: *Move Bottle to Bin*, *Move Fruit to Basket*, and *Bottle on Shelf*. Each task comprises five sequential DAG stages: source navigation, object picking, object transport, placement, and arm retraction. Using Mana, we automatically synthesize approximately 1,000 successful demonstrations per task and jointly post-train each model across the three tasks using the same datasets. We report checkpoints at 30k training steps for Wall-x-preview and 45k for $\pi_{0.5}$.

Both models are evaluated under the same protocol, with 40 episodes per task in each of the in-distribution (ID) and out-of-distribution (OOD) settings. ID uses the green-booth environment, whereas OOD varies

Table 1 X2Real evaluation results. Each cell reports ID/OOD.

Task	Wall-x-preview		$\pi_{0.5}$		DreamZero		Wall-OSS-0.5	
	SR (%)	Score	SR (%)	Score	SR (%)	Score	SR (%)	Score
Reasoning								
Generalization								
atom_objects_beside_block	92.50/75.00	94.50/84.75	50.00/7.32	70.25/53.30	5.00/2.50	35.80/47.00	15.00/7.50	50.00/44.00
atom_bottle_on_plate	70.00/55.00	84.50/78.50	22.50/15.00	58.50/52.00	25.00/5.00	47.80/67.50	0.00/0.00	41.25/32.50
atom_fruit_in_basket	97.50/87.50	97.50/96.50	65.00/67.50	67.00/86.50	65.00/17.50	56.00/80.50	17.50/20.00	21.50/34.25
atom_spoon	82.50/80.00	89.50/87.25	12.50/32.50	31.50/58.50	0.00/12.50	36.00/48.30	35.00/12.50	68.75/38.75
atom_drawer	100.00/55.00	100.00/86.00	40.00/5.00	62.50/27.50	0.00/17.50	15.20/54.70	2.50/0.00	15.50/11.00
cup_on_plate	90.00/0.00	96.00/27.00	37.50/0.00	77.80/25.50	12.50/0.00	75.30/17.30	0.00/0.00	41.25/25.50
open_door	90.00/2.50	94.00/36.00	62.50/0.00	77.20/33.00	0.00/0.00	63.00/9.50	0.00/0.00	17.00/4.50
average	**88.93/50.71**	**93.71/70.86**	**41.43/18.19**	**63.54/48.04**	**15.36/7.86**	**47.01/46.40**	**10.00/5.71**	**36.46/27.21**
Visual Reasoning								
classify_objects_color	92.50/62.50	96.00/83.50	35.00/20.00	66.50/54.50	0.00/0.00	19.50/20.50	0.00/0.00	5.00/6.00
classify_object_shape	42.50/30.00	56.00/55.50	30.00/5.00	60.00/36.50	5.00/0.00	41.20/21.70	0.00/0.00	6.00/0.70
adjust_temperature	12.50/2.50	58.50/14.50	0.00/2.50	18.00/14.50	0.00/0.00	2.00/0.00	5.00/0.00	11.00/4.00
image_puzzle	35.00/22.50	61.75/52.25	7.50/5.00	45.00/42.25	0.00/0.00	22.70/28.50	0.00/0.00	11.75/10.00
adjust_balance	56.25/18.33	69.25/35.92	31.25/6.67	44.88/14.08	0.00/0.00	32.25/12.77	0.00/0.00	7.25/1.25
rotate_book	80.00/37.50	80.00/41.50	60.00/15.00	70.00/31.00	2.50/0.00	26.00/18.00	0.00/0.00	4.00/0.00
alphabet_word	16.25/1.88	33.75/10.88	0.00/0.00	3.15/0.94	0.00/0.00	1.45/0.62	0.00/0.00	0.38/0.25
average	**47.86/25.03**	**65.04/42.01**	**23.39/7.74**	**43.93/27.68**	**1.07/0.00**	**20.73/14.58**	**0.71/0.00**	**6.48/3.17**
Language Reasoning								
fruit_to_basket	52.50/32.50	91.50/69.25	27.50/20.00	50.30/40.80	0.00/2.50	54.00/37.70	2.44/0.00	13.17/8.20
and_expression	87.50/52.50	95.00/92.50	75.00/37.50	77.50/62.50	2.50/0.00	45.00/52.00	15.00/15.00	19.00/21.00
or_expression	87.50/57.50	95.00/62.50	75.00/37.50	77.50/44.50	2.50/7.50	45.00/59.00	15.00/7.32	19.00/7.32
negative_expression	87.50/2.50	95.00/37.50	75.00/0.00	77.50/0.00	2.50/2.50	45.00/9.50	15.00/0.00	19.00/0.00
confusion_instruction	87.50/75.00	95.00/82.50	75.00/52.50	77.50/70.00	2.50/2.50	45.00/47.00	15.00/5.00	19.00/9.00
specific_position	92.50/77.50	92.50/77.50	0.00/2.50	0.00/2.50	10.00/15.00	38.00/34.00	12.50/17.50	24.50/19.50
average	**82.50/49.58**	**94.00/70.29**	**54.58/25.00**	**60.05/36.72**	**3.33/5.00**	**45.33/39.87**	**12.49/7.47**	**18.95/10.84**
Memory Tracking								
track_object_under_cup	17.50/5.00	29.50/20.00	2.50/5.00	19.00/11.00	0.00/0.00	27.70/22.50	0.00/2.50	0.75/7.75
buttons_random_order	38.75/11.25	54.87/32.88	31.25/18.75	46.75/37.25	0.00/0.00	23.65/16.60	8.75/2.50	26.25/24.00
hit_times	65.00/55.00	81.75/81.00	25.00/27.50	50.00/55.50	0.00/0.00	78.80/78.80	7.50/17.50	36.50/43.50
find_drawer	80.00/5.00	92.50/40.75	17.50/0.00	62.00/10.00	0.00/0.00	11.50/9.00	0.00/0.00	11.00/1.00
put_block_back	60.00/25.00	75.75/49.00	12.50/17.50	53.00/48.50	10.00/5.00	50.50/49.00	2.50/0.00	10.00/2.50
average	**52.25/20.25**	**66.88/44.73**	**17.75/13.75**	**46.15/32.45**	**2.00/1.00**	**38.43/35.18**	**3.75/4.50**	**16.90/15.75**
total average	**67.88/36.39**	**79.91/56.97**	**21.79/14.61**	**40.50/33.62**	**5.44/3.46**	**37.88/34.01**	**6.74/4.42**	**19.70/14.24**
Manipulation								
Fine Manipulation								
stack_blocks	70.00/10.00	90.00/52.25	2.50/0.00	33.00/24.50	0.00/0.00	26.50/0.50	0.00/0.00	1.00/0.50
triangle_cup	20.00/5.00	64.50/61.25	2.50/0.00	52.25/49.75	15.00/10.00	41.50/36.00	0.00/0.00	5.75/3.50
ring_to_rod	17.50/20.00	80.50/78.25	2.50/0.00	39.00/43.00	2.50/0.00	56.70/37.50	2.50/0.00	6.50/0.00
plug_in_charger	0.00/0.00	13.50/20.25	0.00/0.00	9.00/6.80	0.00/0.00	0.70/0.70	0.00/0.00	0.75/0.75
insert_flower	62.50/45.00	93.00/92.00	25.00/12.50	75.75/69.50	0.00/2.50	31.00/38.00	0.00/2.50	16.50/20.25
average	**34.00/16.00**	**68.30/60.80**	**6.50/2.50**	**41.80/38.71**	**3.50/2.50**	**31.28/22.54**	**0.50/0.50**	**6.10/5.00**
Bimanual Collaboration								
hand_over_coin	40.00/25.00	64.75/58.00	0.00/0.00	31.50/31.50	5.00/5.00	46.50/45.80	0.00/0.00	24.00/21.70
hand_over	65.00/47.50	75.50/67.75	7.14/10.00	34.50/36.20	12.50/10.00	45.00/40.50	7.50/5.00	31.50/30.50
put_glasses_on_woodshelf	95.00/50.00	99.50/89.00	72.50/47.50	94.50/84.50	0.00/0.00	57.00/57.20	10.00/5.00	53.30/33.50
object_to_cup	52.50/17.50	74.00/43.75	2.50/0.00	29.80/19.50	0.00/0.00	25.00/18.00	0.00/0.00	15.25/11.75
average	**63.12/35.00**	**78.44/64.62**	**20.54/14.38**	**47.58/42.93**	**4.38/3.75**	**43.38/40.38**	**4.38/2.50**	**31.01/24.36**
Tool Use								
Whack_a_Mole	95.00/67.50	96.00/73.50	50.00/15.00	70.50/49.00	25.00/2.50	50.80/20.50	0.00/0.00	13.00/7.50
sweep_trash	92.50/67.50	97.50/87.25	55.00/22.50	74.00/45.50	2.50/5.00	45.30/51.70	2.50/0.00	30.25/22.50
screw	0.00/0.00	18.50/18.50	0.00/0.00	19.00/18.00	0.00/0.00	20.00/20.00	0.00/0.00	19.50/20.00
average	**62.50/45.00**	**70.67/59.75**	**35.00/12.50**	**54.50/37.50**	**9.17/2.50**	**38.70/30.73**	**0.83/0.00**	**20.92/16.67**
Dynamic Objects								
pick_and_place_on_turntable	0.00/32.50	1.00/40.00	9.76/35.00	12.00/38.00	0.00/5.00	8.80/27.30	0.00/4.88	6.00/9.76
average	**0.00/32.50**	**1.00/40.00**	**9.76/35.00**	**12.00/38.00**	**0.00/5.00**	**8.80/27.30**	**0.00/4.88**	**6.00/9.76**
total average	**39.91/32.12**	**54.60/56.29**	**17.95/16.09**	**38.97/39.28**	**4.26/3.44**	**30.54/30.24**	**1.43/1.97**	**16.01/13.95**
All								
total average	**53.90/34.26**	**67.25/56.63**	**26.12/16.13**	**46.19/37.75**	**4.85/3.45**	**34.21/32.12**	**4.08/3.20**	**17.85/14.10**

Table 2 Mobile Operation Evaluation. Each cell reports ID/OOD.

Task	Wall-x-preview		$\pi_{0.5}$	
	SR (%)	Score	SR (%)	Score
`move_bottle_to_bin`	35/7.5	67/45	37.5/0	70.5/25
`move_fruit_to_basket`	85/2.5	93.5/45	15/0	38/30.5
`put_bottle_on_woodshelf`	55/5	72/42.5	35/2.5	50/19
average	**58.33/5.00**	**77.50/44.17**	**29.17/0.83**	**52.83/24.83**

scene backgrounds and table appearance while retaining the same object pools, layout distributions, and task definitions. The episode time limits are 90 seconds for ID and 120 seconds for OOD.

As shown in Table 2, Wall-x-preview achieves higher task-mean SR and Score on both splits, with the largest ID success-rate difference on *Move Fruit to Basket* (85.0% versus 15.0%). Nevertheless, both models have low OOD success: Wall-x-preview completes 6 of 120 episodes and $\pi_{0.5}$ completes one. Their OOD Scores of 44.17 and 24.83, respectively, indicate partial progress despite limited end-to-end success.

Figure 22 reveals distinct execution profiles behind similar task-level outcomes. On *Move Bottle to Bin* under ID, Wall-x-preview and $\pi_{0.5}$ achieve similar success rates (35.0% and 37.5%), but their most frequent first incomplete stages differ: transport for Wall-x-preview (16 episodes) and placement for $\pi_{0.5}$ (20 episodes). Under OOD, picking becomes the most frequent first incomplete stage for $\pi_{0.5}$ (32 episodes), whereas placement is the most frequent for Wall-x-preview (14 episodes).

For *Move Fruit to Basket*, $\pi_{0.5}$ completes picking in only 14 ID and 15 OOD episodes, revealing a limitation already present on ID. Wall-x-preview completes picking in 39 ID and 31 OOD episodes, but its placement completion decreases from 36 to three. On *Bottle on Shelf*, nine Wall-x-preview OOD episodes complete placement, but only two satisfy the final arm-retraction node, which also checks that the object remains validly placed. These DAG records distinguish early picking limitations from unfinished placement and finalization, without attributing them to specific physical failure causes.

4.3 Automatic Synthesis vs. Teleoperation

We compare Task-DAG-guided automatic synthesis with master–slave teleoperation on the dual-arm `stack_blocks (hard)` challenge and further study how full-task and stagewise instructions affect the two data sources. The task requires the robot to stack 12 blocks consecutively and return both arms to their home configurations; as the tower grows, increasingly precise end-effector alignment and placement are required. Crossing the two data sources with the two instruction settings yields four experimental groups. Automatic synthesis and teleoperation each provide 1,398 complete task trajectories and use the same training budget. Within each data source, the two models use the same underlying trajectories; the stagewise variant only reorganizes episodes according to the stage boundaries recorded by the Task DAG and assigns the corresponding subtask instruction to each segment. All four models are initialized from OpenPI $\pi_{0.5}$ and trained for two epochs using identical visual and proprioceptive inputs, training hyperparameters, and action representations.

For each group, we report one representative checkpoint evaluated over 20 closed-loop episodes. Each episode runs for at most 300 seconds. An episode is considered successful only if all 12 layers are completed and both arms return to their home configurations. For incomplete episodes, task progress is measured by the number of stably completed layers latched by the Task DAG.

Under the full-task instruction setting, the automatically synthesized data yield an average of 3.75 completed layers and a maximum of 6, whereas the teleoperated data yield an average of 1.90 layers and a maximum of 4. Automatic synthesis therefore improves the mean progress by 1.85 layers and maintains a higher layer-wise completion rate beyond the third layer. On this precision-intensive, multi-stage stacking task, the result indicates that Task-DAG-guided automatic synthesis provides more effective policy-training data than master–slave teleoperation. The two demonstration sources differ substantially in their temporal structure. As the tower grows, teleoperated trajectories devote progressively more time to end-effector alignment, visual

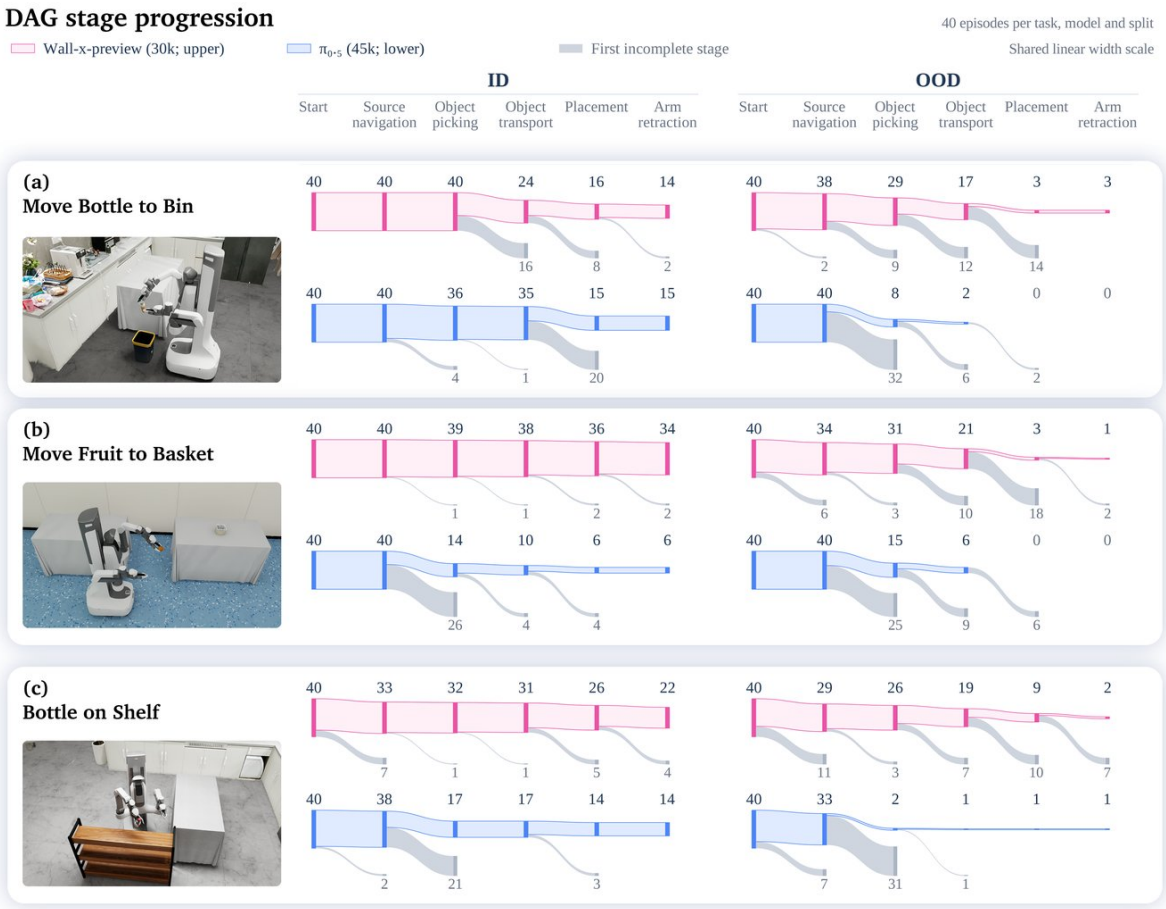


Figure 22 DAG-based Mobile Operation Analysis. ID and OOD stage progression for the three mobile tasks. Pink (upper) and blue (lower) ribbons represent Wall-x-preview and $\pi_{0.5}$, respectively. Each flow starts with 40 episodes, and all ribbon widths use a shared linear count scale. Colored flows indicate completion of successive DAG stages; gray branches count episodes whose first incomplete stage at termination is the corresponding column. Numbers denote episode counts; zero-count branches are omitted.

inspection, and local position correction. When fixed-length action chunks are sampled from the temporal sequence, these long, low-motion segments occupy a larger fraction of the training distribution, reducing the relative frequency of grasping and placement actions that advance the task. In contrast, automatic synthesis generates more compact trajectories at the level of Task DAG nodes and can control coverage across stages, thereby reducing the training-distribution bias caused by differences in stage duration.

The DAG-aligned stagewise setting further improves task progress. For automatic synthesis, the mean number of completed layers increases from 3.75 to 4.40 and the maximum from 6 to 10. The completion rate at layer 5 rises from 25% to 45%, and that at layer 6 from 15% to 30%; the policy progresses from never reaching layer 7 to completing as many as 10 layers, with the improvement concentrated in later stacking stages. Under a single full-task instruction, all 12 stacking stages share the same language condition even though their local objectives and action distributions differ, requiring the policy to infer the current operation primarily from visual observations. The stagewise setting instead uses Task DAG completion boundaries to organize training segments and assigns the corresponding local objective to each segment, thereby reducing the action ambiguity caused by associating one instruction with multiple operation stages. Because the automatic trajectories are themselves generated at the node level and verified by node success predicates, their stage boundaries, instruction semantics, and physical outcomes are tightly aligned. For teleoperation, the mean number of completed layers increases from 1.90 to 2.05 while the maximum remains 4, indicating that explicit stage

Table 3 Closed-loop performance of automatic synthesis and teleoperation under full-task and stagewise instruction settings. Each row reports one representative checkpoint evaluated over 20 episodes.

Data source	Instruction	Checkpoint	Mean layers	Max layer
Automatic synthesis	Full-task	55k	3.75	6
Teleoperation	Full-task	50k	1.90	4
Automatic synthesis	Stagewise	40k	4.40	10
Teleoperation	Stagewise	50k	2.05	4

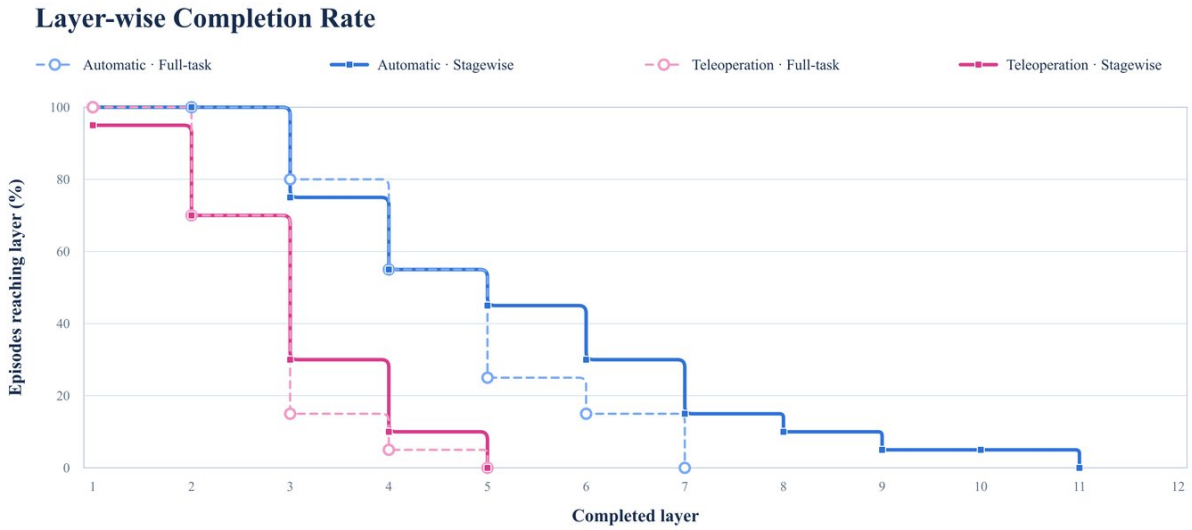


Figure 23 Layer-wise completion rates across 20 closed-loop episodes. Blue and pink denote automatic synthesis and teleoperation, respectively. Solid lines denote stagewise instructions, whereas dashed lines denote a single full-task instruction. Each curve extends to the first layer with a zero completion rate.

objectives benefit both data sources, with a more pronounced gain for automatic synthesis.

5 Related Work

5.1 Evaluating Generalist Robot Manipulation Policies

Generalist robot policies are often trained to acquire diverse manipulation skills and generalize across multiple dimensions, such as objects, environments, language instructions, task configurations, and robot embodiments. Representative approaches to building such policies include vision-language-action models (VLAs) (1, 52, 53, 2, 54, 55, 4), as well as world action models (WAMs) and related video–action models that jointly capture visual dynamics and robot actions (56, 57, 58, 3, 5, 59). Evaluating these policies across diverse manipulation skills and generalization dimensions therefore poses new challenges.

For single-task policies, success rates under a fixed evaluation protocol often serve as the primary measure of performance. For generalist policies, however, evaluation must also assess both the range of manipulation skills and their generalization beyond the objects, environments, language instructions, task configurations, and robot embodiments encountered during training. Evaluation therefore cannot rely solely on aggregate success rates across a small set of fixed tasks. This requires broad yet controlled coverage across relevant skills and generalization dimensions (60, 2, 61).

Real-robot evaluation provides direct evidence of policy performance in physical systems, naturally incorporating sensor noise, contact dynamics, control delays, and hardware-specific effects. However, statistically reliable evaluation requires repeated trials across different tasks and initial conditions. Each trial requires access to a

robot and often involves manual scene setup, environment reset, safety monitoring, and success assessment, making large-scale evaluation costly and time-consuming. Differences in robot hardware, evaluation environments, and evaluation protocols across institutions further hinder reproducibility. Recent work has begun to address these challenges through automated environment reset and success detection (62), distributed evaluation across institutions (7), and more rigorous statistical evaluation protocols (63). Nevertheless, the scale of real-robot evaluation remains limited by the number of available robots and evaluation environments. This limited scale poses a particular challenge during generalist policy evaluation, where many training configurations, checkpoints, and hyperparameter settings must be compared. Simulation-based evaluation offers a scalable complement to the real-robot evaluation and can partially address these limitations.

5.2 Simulation-based Evaluation of Robot Manipulation Policies

Simulation benchmarks provide standardized, reproducible, and scalable environments for evaluating robot policies. Meta-World targets multi-task and meta-reinforcement learning (64), RLBench supports vision-based manipulation under several learning settings (65), and robosuite provides modular and standardized environments for robot learning (66). ManiSkill and ManiSkill2 progressively expand the coverage of objects, tasks, robot embodiments, and observation modalities (67, 68), while ManiSkill3 further introduces GPU-parallelized physics simulation and rendering for more efficient data collection and policy evaluation (69). CALVIN evaluates policies on long-horizon manipulation tasks specified by natural-language instructions (70), whereas LIBERO evaluates knowledge transfer across sequentially learned manipulation tasks (24). More recent benchmarks, including BEHAVIOR-1K, RoboCasa, VLABench, RoboTwin 2.0, RoboVerse, MolmoSpaces, RoboCasa365, and RoboDojo, further expand evaluation to richer household environments, more diverse language-conditioned and long-horizon tasks, bimanual and mobile manipulation, and a wider range of robot embodiments (12, 25, 71, 14, 72, 26, 13, 15).

Most existing simulation benchmarks rely on physics-based simulation. Action-conditioned world models have also been studied as learned environments for robot policy evaluation by predicting future visual observations conditioned on robot actions (73, 74, 75, 76, 77, 78, 79). Learned and physics-based simulators provide complementary forms of environment modeling. Because our framework requires explicit state access, state-based success detection, and systematic control over assets, tasks, and evaluation conditions, we adopt physics-based simulation.

When simulation is used as a proxy for real-robot evaluation, its validity depends on whether performance measured in simulation reflects real-robot performance. Simulators allow explicit configuration of visual conditions, dynamics, controllers, and hardware parameters, but these configurations must be calibrated against the target physical system (49). SIMPLER studies sim–real correspondence through paired simulation and real-robot evaluations (9). PolaRiS and REALM extend this line of research through reconstructed real-world environments and controlled evaluation conditions, respectively (10, 11). VISER further studies how visual realism affects the agreement between simulation and real-world evaluation (80), while related work extends sim–real policy evaluation to deformable-object interactions (81, 82). SureSim instead combines a large number of simulation trials with a small number of paired real-robot trials to estimate real-robot performance and its confidence interval (83).

Beyond sim–real correspondence, evaluation diversity depends not only on the number of tasks, but also on broad yet controlled coverage of manipulation skills and generalization across objects, environments, language instructions, task configurations, and robot embodiments (60). Colosseum, Colosseum V2, REALM, and RoboLab use controlled variations to analyze policy robustness and generalization (61, 84, 11, 85). EBench provides capability- and generalization-level diagnosis for mobile manipulation (86), while RoboDojo evaluates generalist policies across generalization, memory, precision, long-horizon execution, and open-vocabulary instruction following in both simulation and real-world settings (15).

Another concern is benchmark integrity, including the separation between training and evaluation and the complete specification of evaluation protocols. RoboLab highlights domain overlap between training and evaluation environments (85). Recent benchmark audits further identify shortcut solvability, insufficient statistical evidence, gradual overfitting to established benchmarks, and dependence on training-data sources as factors that can limit the interpretation of benchmark results (87). Reproduction experiments conducted with

vla-eval also show that undocumented termination conditions, data normalization, and other implementation details can affect evaluation results (17). Together, these works motivate simulation benchmarks that provide evidence of sim–real correspondence, broad yet controlled coverage of relevant skills and generalization dimensions, and clearly specified training–evaluation splits and evaluation protocols.

5.3 Simulation Data Engines for Robot Learning

Simulation can serve not only as an environment for predefined training and evaluation tasks, but also as a system for continually producing robot learning data. In this work, we use the term *simulation data engine* to denote an integrated pipeline for constructing and validating simulation-ready assets, generating executable scenes and tasks, collecting and validating demonstrations, organizing the resulting data, and maintaining a clear separation between training and evaluation distributions. A simulation benchmark primarily specifies what is evaluated and under which conditions, whereas a simulation data engine provides the underlying processes for producing and managing the assets, tasks, and data used for training and evaluation.

Simulation-ready assets form the foundation of a data engine and must support the physical interactions involved in its target tasks. SAPIEN, PartNet-Mobility, Objaverse, and GAPartNet provide important foundations for 3D objects, articulated structures, and manipulation-relevant part semantics (88, 27, 89). Building on these foundations, assets intended for robot simulation may require additional task-dependent information, such as metric scale, canonical orientation, part and task semantics, physically based rendering (PBR) materials, collision geometry, mass and inertia, contact properties, and executable kinematic structures. Recent generative models extend high-quality 3D generation toward simulation-ready and physically grounded asset creation (90, 91, 92). Articulate-Anything automates articulated-object modeling, whereas URDF-Anything+ generates executable articulated models directly from visual observations (93, 94). ManiTwin provides a scalable pipeline for generating simulation-ready assets enriched with physical properties, functional annotations, and language descriptions (32). AnnotateAnything generates manipulation-relevant annotations grounded in asset geometry and physical constraints (95). VISER further develops material-aware asset preparation for visually realistic policy evaluation (80). For reliable use at scale, automatically constructed assets generally require geometric, physical, and interaction-level validation. More generally, simulation readiness is task-relative: it depends on whether an asset provides the geometric, physical, semantic, and, where applicable, kinematic information required by its intended role, such as support surfaces, containment volumes, articulation parameters, and contact regions.

Given suitable assets, a data engine can expand its coverage by generating scenes, tasks, and demonstrations across diverse conditions and behaviors. GenSim uses large language models to generate simulation tasks, environments, and expert demonstrations (96). RoboGen organizes task proposal, scene generation, supervision generation, and skill learning into a generative pipeline (97). RoboTwin 2.0 combines multimodal language models with simulation-in-the-loop refinement to generate executable task-level programs (14). Demonstrations can be collected through teleoperation, scripted experts, motion planning, trajectory optimization, and reinforcement learning. MimicGen and its subsequent extensions generate additional demonstrations from limited source data by adapting object-centric motion segments and extending this paradigm to skill composition, bimanual dexterous manipulation, dynamic tasks, deformable objects, and humanoid loco-manipulation (98, 99, 100, 101, 102, 103). Such methods can reduce the need to collect demonstrations manually for every task condition, while the resulting coverage remains shaped by factors such as the available assets, task specifications, source demonstrations, and validation procedures.

A data engine operating at scale requires coordinated support for asset and scene construction, task and demonstration generation, simulation, and evaluation, together with unified interfaces and mechanisms for validating, organizing, and versioning the resulting data. Recent systems such as RoboTwin 2.0, SimFoundry, and Genie Sim 3.0 integrate multiple components spanning asset and scene construction, task generation, data generation, and policy evaluation (14, 104, 105). RoboVerse provides simulator-agnostic interfaces across multiple physics and rendering backends (72). IsaacIPC couples GPU-accelerated penetration-free contact simulation with Isaac Sim/Lab to support contact-rich rigid–deformable interactions (106). MagicSim supports particle-based fluids, granular materials, and other physical processes, and integrates their simulation with world construction, robot execution, task evaluation, and data generation within a deterministic batched runtime (107). Complementary learned approaches, including DreamGen, Qwen-RobotWorld, and GigaWorld-

0, use video or hybrid world models to generate synthetic visual trajectories and interaction data for policy learning (108, 109, 110). RLDS, robomimic, LeRobot, and Robo-DM support standardized sequential-decision datasets, offline robot learning, end-to-end data collection and learning workflows, and large-scale robot data management, respectively (111, 112, 19, 113). Integrating these components, including asset production, scene and task generation, demonstration collection, simulation infrastructure, quality control, version management, and training–evaluation separation, can support the continued expansion of training and evaluation distributions as policy capabilities evolve. Evaluation results can in turn identify underrepresented conditions and inform subsequent task and data generation.

6 Discussion and Limitations

We present X2Real simulation benchmark, which addresses three core limitations inherent to contemporary robot manipulation simulation benchmarks, namely the simulation-to-reality gap, insufficient task diversity, and evaluative bias stemming from benchmark exploitation. To remedy these deficiencies, X2Real incorporates hardware-aligned physical calibration, a hierarchically structured multi-dimensional task set, orthogonal domain randomization, and strictly decoupled training and evaluation pipelines, yielding improved cross-domain correlation and robust quantification of generalist manipulation performance. Furthermore, the proposed Mana simulation ecosystem supports iterative benchmark update and closed-loop evaluation, alleviating the performance saturation inherent to static benchmark designs. The following discussion analyzes the key findings and inherent limitations of our framework, and highlights potential avenues for future embodied intelligence benchmarking and generalist policy development.

Despite its improved faithfulness, diversity, and fairness over existing benchmarks, X2Real still entails several limitations that point to promising future directions. First, the current benchmark tasks are restricted to rigid and articulated rigid body manipulation, which cannot fully replicate real-world scenarios involving deformable objects, fluids, and soft materials. Extending the physical simulation to support diverse material dynamics is essential for evaluating generalist policies with universal manipulation capabilities. Second, the construction of existing task suites relies on a combination of manual design and agent-assisted refinement, which limits the scalability of task quantity and scenario richness. Fully automated agent-driven task generation will further expand task diversity and continuously enrich the benchmark distribution. Third, our simulation framework currently adopts standard gripper embodiments without tactile sensing modules. Integrating dexterous hands, high-precision tactile feedback, and diverse robot hardware setups can better align with real robotic manipulation research and enhance evaluation generality. Fourth, the evaluation pipeline remains relatively static in paradigm. Future work can introduce intelligent agent-based result analysis to enable automated experimental iteration and comparative study, as well as adversarial and competitive evaluation paradigms similar to RoboArena, to achieve more dynamic, in-depth, and robust policy assessment.

7 Contributors

X2Real is a collaborative effort of the X Square robot team and outside collaborators. * denotes core contributors, † denotes the project leader, ‡ denotes the corresponding author.

Lian Ruan*†, Jade Yang*, Sherphylan Gao*, Felix Gao*, Kyson Liang*, Galen Liu*, Ligo Wu*, Lane Jin*, Guu Gu*, Bevan Xie*, Cloud Yan*, Zongzi Yuan*, Kino Luo, Emma Chen, Shuwen Chen, Yang Ping, Miles Guo, Rain Sun, Kayden Zhang, Alex Du, Ruihai Wu, Liang Hao, Zhaoshuo Li, Roy Gan, Hao Wang‡, Qian Wang.

References

[1] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

[2] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

[3] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026. URL https://arxiv.org/abs/2602.15922.

[4] Ryan Yu, Pushi Zhang, Starrick Liu, Brae Liu, Miracle Kang, Shalfun Li, Lights Shi, Ellie Ma, Ping Yang, Chris Pan, Jerry Chen, Dongxiu Liu, Rain Sun, Miles Guo, Byron Zhang, et al. Wall-OSS-0.5 technical report. *arXiv preprint arXiv:2605.30877*, 2026. URL https://arxiv.org/abs/2605.30877.

[5] Shalfun Li, Victor Yao, Charles Yang, Truth Qu, Regis Cheng, Ryan Yu, Howard Lu, Newton Von, Vincent Chen, Yohann Tang, Maeve Zhang, Ellie Ma, Gody Li, Sage Yang, Lorien Shu, et al. WALL-WM: Carving world action modeling at the event joints. *arXiv preprint arXiv:2606.01955*, 2026. URL https://arxiv.org/abs/2606.01955.

[6] NVIDIA. Cosmos 3: Omnimodal world models for physical ai. *arXiv preprint arXiv:2606.02800*, 2026. URL https://research.nvidia.com/labs/cosmos-lab/cosmos3/technical-report.pdf.

[7] Pranav Atreya, Karl Pertsch, Tony Lee, Moo Jin Kim, Arhan Jain, Artur Kuramshin, Cyrus Neary, Edward S. Hu, Kanav Arora, Kirsty Ellis, et al. RoboArena: Distributed real-world evaluation of generalist robot policies. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 336–364. PMLR, 2025. URL https://proceedings.mlr.press/v305/atreya25a.html.

[8] Yu Sun, Meng Cao, Yang Ping, Kaidong Zhang, Qingxuan Chen, Rongtao Xu, Liangwang Ruan, Xuecheng Chen, Dongxiu Liu, Yunxiao Yan, Zunnan Xu, Runze Xu, Charles Yang, Peilun Zhang, Xiaofan Li, Ruyi Gan, Liang Ma, Yuehao Yin, Jincheng Yu, Lufang Chen, Yuxin Liang, Peng Zhai, Hao Wang, Ivan Laptev, Ian Reid, Qian Wang, and Xiaodan Liang. Maniparena: Comprehensive real-world evaluation of reasoning-oriented generalist robot manipulation, 2026. URL https://arxiv.org/abs/2603.28545.

[9] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 3705–3728. PMLR, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

[10] Arhan Jain, Mingtong Zhang, Kanav Arora, William Chen, Marcel Torne, Muhammad Zubair Irshad, Sergey Zakharov, Yue Wang, Sergey Levine, Chelsea Finn, Wei-Chiu Ma, Dhruv Shah, Abhishek Gupta, and Karl Pertsch. PolaRiS: Scalable real-to-sim evaluations for generalist robot policies. In *Proceedings of Robotics: Science and Systems*, 2026. URL https://roboticsconference.org/program/papers/62/.

[11] Martin Sedlacek, Pavlo Yefanov, Georgy Ponimatkin, Jai Bardhan, Simon Pilc, Mederic Fourmy, Evangelos Kazakos, Cees G. M. Snoek, Josef Sivic, and Vladimir Petrik. REALM: A real-to-sim validated benchmark for generalization in robotic manipulation. *IEEE Robotics and Automation Letters*, 2026. URL https://ieeexplore.ieee.org/document/11513985/.

[12] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Mart
'in-Mart
'in, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. BEHAVIOR-1K: A benchmark for embodied AI with 1,000 everyday activities and realistic simulation. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 80–93. PMLR, 2023. URL https://proceedings.mlr.press/v205/li23a.html.

[13] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tQJYKwc3n4.

[14] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

[15] Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Weijie Wan, Baijun Chen, Haoran Lu, Haowen Yan,

Honghao Su, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026. URL https://arxiv.org/abs/2607.04434.

[16] StarVLA Community. StarVLA: A lego-like codebase for vision-language-action model developing, 2026. URL https://arxiv.org/abs/2604.05014.

[17] Suhwan Choi, Yunsung Lee, Yubeen Park, Chris Dongjoo Kim, Ranjay Krishna, Dieter Fox, and Youngjae Yu. vla-eval: A unified evaluation harness for vision-language-action models. *arXiv preprint arXiv:2603.13966*, 2026. URL https://arxiv.org/abs/2603.13966.

[18] Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

[19] Remi Cadene, Simon Aliberts, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, Mustafa Shukor, Jess Moss, Alexander Soare, Dana Aubakirova, Quentin Lhoest, Quentin Gallouédec, and Thomas Wolf. LeRobot: An open-source library for end-to-end robot learning. *arXiv preprint arXiv:2602.22818*, 2026. URL https://arxiv.org/abs/2602.22818.

[20] XPolicyLab Community, Tianxing Chen, Yue Chen, Tian Nian, et al. XPolicyLab: A unified standard and open ecosystem for robot policy evaluation and deployment. *arXiv preprint arXiv:2608.09892*, 2026. doi: 10.48550/arXiv.2608.09892. URL https://arxiv.org/abs/2608.09892.

[21] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

[22] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, 2024.

[23] NVIDIA Isaac Lab-Arena Contributors. Isaac lab-arena: Composable environment creation and policy evaluation for robotics, 2025. URL https://github.com/isaac-sim/IsaacLab-Arena.

[24] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 44776–44791, 2023. URL https://arxiv.org/abs/2306.03310.

[25] 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 *Proceedings of Robotics: Science and Systems*, 2024. URL https://arxiv.org/abs/2406.02523.

[26] Yejin Kim, Wilbert Pumacay, Omar Rayyan, Max Argus, Winson Han, Eli VanderBilt, Jordi Salvador, Abhay Deshpande, Rose Hendrix, Snehal Jauhri, et al. MolmoSpaces: A large-scale open ecosystem for robot navigation and manipulation. *arXiv preprint arXiv:2602.11337*, 2026. URL https://arxiv.org/abs/2602.11337.

[27] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023. URL https://arxiv.org/abs/2212.08051.

[28] Li Jin, Yuchen Yang, Weikai Chen, Yujie Wang, Dehao Hao, Tanghui Jia, Yingda Yin, Zeyu Hu, Runze Zhang, Keyang Luo, Li Yuan, Long Quan, Xin Wang, and Xueying Qin. CanoVerse: 3D object scalable canonicalization and dataset for generation and pose. *arXiv preprint arXiv:2603.07144*, 2026. doi: 10.48550/arXiv.2603.07144. URL https://arxiv.org/abs/2603.07144.

[29] Jiyao Zhang, Weiyao Huang, Bo Peng, Mingdong Wu, Fei Hu, Zijian Chen, Bo Zhao, and Hao Dong. Omni6DPose: A benchmark and model for universal 6D object pose estimation and tracking. In *Computer Vision – ECCV 2024*, volume 15133 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 2024. doi: 10.1007/978-3-031-73226-3_12.

[30] Blendkit. Blendkit (formerly BlenderKit): The integrated 3D asset library. https://www.blendkit.com/our-mission/, 2026. Accessed 2026-08-11.

[31] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B. McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3D scanned household items. In *2022 IEEE International Conference on Robotics and Automation*, pages 2553–2560, 2022. doi: 10.1109/ICRA46639.2022.9811809.

[32] Kaixuan Wang, Tianxing Chen, Jiawei Liu, Honghao Su, Shaolong Zhu, Minxuan Wang, Zixuan Li, Yue Chen, Huan-ang Gao, Yusen Qin, Jiawei Wang, Qixuan Zhang, Lan Xu, Jingyi Yu, Yao Mu, and Ping Luo. ManiTwin: Scaling data-generation-ready digital object dataset to 100k. *arXiv preprint arXiv:2603.16866*, 2026. URL https://arxiv.org/abs/2603.16866.

[33] 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 ObjectGoal navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16384–16393, 2024.

[34] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-scale embodied AI using procedural generation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[35] Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F. Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. ABO: Dataset and benchmarks for real-world 3D object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21126–21136, 2022.

[36] Poly Haven. Poly Haven: The public 3D asset library. https://polyhaven.com/, 2026. Accessed 2026-08-11.

[37] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimír Vondruš, 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*, volume 34, 2021.

[38] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Yale-CMU-Berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017. doi: 10.1177/0278364917700714.

[39] Zhe Zhu, Le Wan, Rui Xu, Yiheng Zhang, Honghua Chen, Zhiyang Dou, Cheng Lin, Yuan Liu, and Mingqiang Wei. PartSAM: A scalable promptable part segmentation model trained on native 3D data. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=y8sZUQPYXC.

[40] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3D mesh approximate convex decomposition. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3501–3504. IEEE, 2009. doi: 10.1109/ICIP.2009.5414068.

[41] Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3D meshes with collision-aware concavity and tree search, 2022. URL https://arxiv.org/abs/2205.02961.

[42] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3D embodied AI environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024.

[43] Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, Shuran Song, and Fangyin Wei. SAGE: Scalable agentic 3D scene generation for embodied AI. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22358–22368, June 2026.

[44] Zhe Liu, Huanbo Jin, Zhaohui Du, Zhe Wang, Dongzhan Zhou, Minting Pan, He Xu, Peijia Li, Jiaming Gu, Quan Lu, Qi Wang, Bin Ji, and Ting Xiao. Pipette: An embodied simulation platform, benchmark, and data-efficient augmentation framework for wet-lab robotics. *arXiv preprint arXiv:2606.12936*, 2026. URL https://arxiv.org/abs/2606.12936.

[45] Jie Tan, Karen Liu, and Greg Turk. Stable proportional-derivative controllers. *IEEE Computer Graphics and Applications*, 31(4):34–44, 2011. URL https://ieeexplore.ieee.org/document/5719567.

[46] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. URL https://direct.mit.edu/evco/article/9/2/159/892/Completely-Derandomized-Self-Adaptation-in.

[47] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. doi: 10.1017/9781316671528.

[48] Yash Jangir, Yidi Zhang, Pang-Chi Lo, Kashu Yamazaki, Chenyu Zhang, Kuan-Hsun Tu, Tsung-Wei Ke, Lei Ke, Yonatan Bisk, and Katerina Fragkiadaki. RobotArena ∞: Scalable robot benchmarking via real-to-sim translation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=OutljIofvS.

[49] Bhairav Mehta, Ankur Handa, Dieter Fox, and Fabio Ramos. A user's guide to calibrating robotics simulators. In *Proceedings of the Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 1326–1340. PMLR, 2021. URL https://proceedings.mlr.press/v155/mehta21a.html.

[50] Antonia Bronars, Younghyo Park, and Pulkit Agrawal. Tune to learn: How controller gains shape robot policy learning. *arXiv preprint arXiv:2604.02523*, 2026. URL https://arxiv.org/abs/2604.02523.

[51] Filip Bjelonic, Fabian Tischhauser, and Marco Hutter. Towards bridging the gap: Systematic sim-to-real transfer for diverse legged robots. *The International Journal of Robotics Research*, 2026. doi: 10.1177/02783649261459628. URL https://journals.sagepub.com/doi/10.1177/02783649261459628.

[52] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of*

the 8th Conference on Robot Learning, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

[53] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

[54] Abbas Abdolmaleki, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Ashwin Balakrishna, et al. Gemini Robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025. URL https://arxiv.org/abs/2510.03342.

[55] NVIDIA. NVIDIA Isaac GR00T N1.7. Hugging Face model card, 2026. URL https://huggingface.co/nvidia/GR00T-N1.7-3B. Accessed: 2026-08-14.

[56] Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.074. URL https://www.roboticsproceedings.org/rss21/p074.html.

[57] Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.015. URL https://www.roboticsproceedings.org/rss21/p015.html.

[58] Jonas Pai, Liam Achenbach, Oliver Sanchez, Stefanos Charalambous, Victoriano Montesinos, Benedek Forrai, Oier Mees, and Elvis Nava. mimic-video: Video-action models for generalizable robot control beyond VLAs. *arXiv preprint arXiv:2512.15692*, 2025. URL https://arxiv.org/abs/2512.15692.

[59] Qihang Zhang, Lin Li, Luyao Zhang, Shuai Yang, Yiming Luo, Shuaiting Li, Ruilin Wang, Junke Wang, Jiahao Shao, Yinghao Xu, et al. Native video-action pretraining for generalizable robot control. *arXiv preprint arXiv:2607.08639*, 2026. URL https://arxiv.org/abs/2607.08639.

[60] Jensen Gao, Suneel Belkhale, Sudeep Dasari, Ashwin Balakrishna, Dhruv Shah, and Dorsa Sadigh. A taxonomy for evaluating generalist robot manipulation policies. *IEEE Robotics and Automation Letters*, 11(3):3182–3189, 2026. doi: 10.1109/LRA.2026.3656785. URL https://ieeexplore.ieee.org/document/11361080.

[61] Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation. In *Proceedings of Robotics: Science and Systems*, 2024. URL https://www.roboticsproceedings.org/rss20/p133.html.

[62] Zhiyuan Zhou, Pranav Atreya, You Liang Tan, Karl Pertsch, and Sergey Levine. AutoEval: Autonomous evaluation of generalist robot manipulation policies in the real world. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 1997–2017. PMLR, 2025. URL https://proceedings.mlr.press/v305/zhou25a.html.

[63] Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, et al. A careful examination of large behavior models for multitask dexterous manipulation. *Science Robotics*, 11(113):eaea6201, 2026. doi: 10.1126/scirobotics.aea6201. URL https://www.science.org/doi/10.1126/scirobotics.aea6201.

[64] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR, 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

[65] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark and learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. doi: 10.1109/LRA.2020.2974707. URL https://ieeexplore.ieee.org/document/9001253.

[66] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Mart
'in-Mart
'in, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020. URL https://arxiv.org/abs/2009.12293.

[67] 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 *Advances in Neural Information Processing Systems: Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2107.14483.

[68] 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 *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2302.04659.

[69] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. ManiSkill3: GPU parallelized robotics simulation and rendering for generalizable embodied AI. In *Proceedings of Robotics: Science and Systems*, 2025. URL https://arxiv.org/abs/2410.00425.

[70] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3): 7327–7334, 2022. URL https://arxiv.org/abs/2112.03227.

[71] Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Zhang_VLABench_A_Large-Scale_Benchmark_for_Language-Conditioned_Robotics_Manipulation_with_Long-Horizon_ICCV_2025_paper.html.

[72] Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, et al. RoboVerse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. In *Proceedings of Robotics: Science and Systems*, 2025. URL https://arxiv.org/abs/2504.18904.

[73] Yaxuan Li, Yichen Zhu, Junjie Wen, Chaomin Shen, and Yi Xu. WorldEval: World model as real-world robot policies evaluator. *arXiv preprint arXiv:2505.19017*, 2025. URL https://arxiv.org/abs/2505.19017.

[74] Julian Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. WorldGym: World model as an environment for policy evaluation. *arXiv preprint arXiv:2506.00613*, 2025. URL https://arxiv.org/abs/2506.00613.

[75] Wei-Cheng Tseng, Jinwei Gu, Qinsheng Zhang, Hanzi Mao, Ming-Yu Liu, Florian Shkurti, and Yen-Chen Lin. Scalable policy evaluation with video world models. *arXiv preprint arXiv:2511.11520*, 2025. URL https://arxiv.org/abs/2511.11520.

[76] Gemini Robotics Team, Krzysztof Choromanski, Coline Devin, Yilun Du, Debidatta Dwibedi, Ruiqi Gao, Abhishek Jindal, Thomas Kipf, Sean Kirmani, Isabel Leal, Fangchen Liu, Anirudha Majumdar, Andrew Marmon, Carolina Parada, Yulia Rubanova, Dhruv Shah, Vikas Sindhwani, Jie Tan, Fei Xia, Ted Xiao, Sherry Yang, Wenhao Yu, and Allan Zhou. Evaluating Gemini Robotics policies in a Veo world simulator. *arXiv preprint arXiv:2512.10675*, 2025. URL https://arxiv.org/abs/2512.10675.

[77] Yixuan Wang, Rhythm Syed, Fangyu Wu, Mengchao Zhang, Aykut Onol, Jose Barreiros, Hooshang Nayyeri, Tony Dear, Huan Zhang, and Yunzhu Li. Interactive world simulator for robot policy training and evaluation. *arXiv preprint arXiv:2603.08546*, 2026. URL https://arxiv.org/abs/2603.08546.

[78] Yaxuan Li, Zhongyi Zhou, Yefei Chen, Yaokai Xue, and Yichen Zhu. dWorldEval: Scalable robotic policy evaluation via discrete diffusion world model. *arXiv preprint arXiv:2604.22152*, 2026. URL https://arxiv.org/abs/2604.22152.

[79] Wei-Cheng Tseng, Gashon Hussein, Yuzhu Dong, Allen Z. Ren, Lucy X. Shi, Xudong Wang, Sergey Levine, Zhaoshuo Li, Jinwei Gu, Florian Shkurti, Ming-Yu Liu, and Quan Vuong. SC3-Eval: Evaluating robot foundation models via self-consistent video generation. *arXiv preprint arXiv:2606.18610*, 2026. URL https://arxiv.org/abs/2606.18610.

[80] Yixin Zhu, Zixiong Wang, Jian Yang, Jin Xie, Jingyi Yu, Jiayuan Gu, and Beibei Wang. Toward visually realistic simulation: A benchmark for evaluating robot manipulation in simulation. *arXiv preprint arXiv:2605.06311*, 2026. URL https://arxiv.org/abs/2605.06311.

[81] Kaifeng Zhang, Shuo Sha, Hanxiao Jiang, Matthew Loper, Hyunjong Song, Guangyan Cai, Zhuo Xu, Xiaochen Hu, Changxi Zheng, and Yunzhu Li. Real-to-sim robot policy evaluation with gaussian splatting simulation of soft-body interactions. In *2026 IEEE International Conference on Robotics and Automation*, 2026. URL https://arxiv.org/abs/2511.04665.

[82] Zeyi Li, Yushi Yang, Shawn Xie, Kyle Xu, et al. LeHome: A simulation environment for deformable object manipulation in household scenarios. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

[83] Apurva Badithela, David Snyder, Lihan Zha, Joseph Mikhail, Matthew O'Kelly, Anushri Dixit, and Anirudha Majumdar. Reliable and scalable robot policy evaluation with imperfect simulators. In *2026 IEEE International Conference on Robotics and Automation*, 2026. URL https://arxiv.org/abs/2510.04354.

[84] Jeremy Morgan, Prajwal Vijay, Hyeonho Oh, Jincen Song, Ashvin Arora, Alina Du, Gaurav Sukhatme, Jesse Thomason, and Ishika Singh. Colosseum V2: Benchmarking generalization for vision language action models. *arXiv preprint arXiv:2605.27759*, 2026. URL https://arxiv.org/abs/2605.27759.

[85] Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A high-fidelity simulation benchmark for analysis of task generalist policies. In *Proceedings of Robotics: Science and Systems*, 2026. URL https://roboticsconference.org/program/papers/96/.

[86] Ning Gao, Jinliang Zheng, Xing Gao, Haoxiang Ma, Hanqing Wang, Yukai Wang, Jiantong Chen, Zanxin Chen, Shujie Zhang, Mingda Jia, et al. EBench: Elemental diagnosis of generalist mobile manipulation policies. *arXiv preprint arXiv:2606.18239*, 2026. URL https://arxiv.org/abs/2606.18239.

[87] Tianchong Jiang, Xiangshan Tan, Samuel Wheeler, Luzhe Sun, Tewodros W. Ayalew, and Matthew Walter. What are we actually benchmarking in robot manipulation? *arXiv preprint arXiv:2606.04233*, 2026. URL https://arxiv.org/abs/2606.04233.

[88] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020. URL https://arxiv.org/abs/2003.08515.

[89] Haoran Geng, Helin Xu, Chengyang Zhao, Chao Xu, Li Yi, Siyuan Huang, and He Wang. GAPartNet: Cross-category domain-generalizable object perception and manipulation via generalizable and actionable parts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7081–7091, 2023. URL https://arxiv.org/abs/2211.05272.

[90] Zibo Zhao, Zeqiang Lai, Qingxiang Lin, et al. Hunyuan3D 2.0: Scaling diffusion models for high resolution textured 3d assets generation. *arXiv preprint arXiv:2501.12202*, 2025. doi: 10.48550/arXiv.2501.12202. URL https://arxiv.org/abs/2501.12202.

[91] Jiashi Feng, Xiu Li, Jing Lin, Jiahang Liu, Gaohong Liu, Weiqiang Lou, Su Ma, Guang Shi, Qinlong Wang, Jun Wang, Zhongcong Xu, Xuanyu Yi, Zihao Yu, Jianfeng Zhang, Yifan Zhu, Rui Chen, Jinxin Chi, Zixian Du, Li Han, Lixin Huang, Kaihua Jiang, Yuhan Li, Guan Luo, Shuguang Wang, Qianyi Wu, Fan Yang, Junyang Zhang, and Xuanmeng Zhang. Seed3D 1.0: From images to high-fidelity simulation-ready 3d assets. *arXiv preprint arXiv:2510.19944*, 2025. doi: 10.48550/arXiv.2510.19944. URL https://arxiv.org/abs/2510.19944.

[92] Ziang Cao, Zhaoxi Chen, Liang Pan, and Ziwei Liu. PhysX-3D: Physical-grounded 3d asset generation. *arXiv preprint arXiv:2507.12465*, 2025. doi: 10.48550/arXiv.2507.12465. URL https://arxiv.org/abs/2507.12465.

[93] Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=s3FTX4Ay55.

[94] Zhuangzhe Wu, Yue Xin, Chengkai Hou, Minghao Chen, Yaoxu Lyu, Jieyu Zhang, and Shanghang Zhang. URDF-Anything+: Autoregressive articulated 3d models generation for physical simulation. *arXiv preprint arXiv:2603.14010*, 2026. URL https://arxiv.org/abs/2603.14010.

[95] Haoran Lu, Mutian Shen, Shuyang Yu, Yu Xiao, Songling Liu, Jianshu Zhang, Shang Wu, Yue Chen, Guo Ye, Jiayi Wang, Zhaoran Wang, and Han Liu. AnnotateAnything: Automatic annotation of 3d assets for robot manipulation. *arXiv preprint arXiv:2606.17446*, 2026. URL https://arxiv.org/abs/2606.17446.

[96] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01361.

[97] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 51936–51983. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24cc.html.

[98] 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 *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1820–1864. PMLR, 2023. URL https://proceedings.mlr.press/v229/mandlekar23a.html.

[99] Caelan Reed Garrett, Ajay Mandlekar, Bowen Wen, and Dieter Fox. SkillMimicGen: Automated demonstration generation for efficient skill learning and deployment. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 2750–2790. PMLR, 2025. URL https://proceedings.mlr.press/v270/garrett25a.html.

[100] Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. DexMimicGen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *2025 IEEE International Conference on Robotics and Automation*, 2025. URL https://arxiv.org/abs/2410.24185.

[101] Vincenzo Pomponi, Paolo Franceschi, Stefano Baraldo, Oliver Avram, Loris Roveda, Luca Maria Gambardella, and Anna Valente. DynaMimicGen: A data generation framework for robot learning of dynamic tasks. *IEEE Robotics and Automation Letters*, 2026. doi: 10.1109/LRA.2026.3703978. URL https://doi.org/10.1109/LRA.2026.3703978.

[102] Masoud Moghani, Mahdi Azizian, Animesh Garg, Yuke Zhu, Sean Huver, and Ajay Mandlekar. SoftMimicGen: A data generation system for scalable robot learning in deformable object manipulation. *arXiv preprint arXiv:2603.25725*, 2026. doi: 10.48550/arXiv.2603.25725. URL https://arxiv.org/abs/2603.25725.

[103] Kevin Lin, Ajay Mandlekar, Caelan Reed Garrett, Nikita Chernyadev, Yu Fang, Runyu Ding, Yuqi Xie, Justin Tran, Linxi Fan, and Yuke Zhu. HumanoidMimicGen: Data generation for loco-manipulation via whole-body planning. *arXiv preprint arXiv:2605.27724*, 2026. doi: 10.48550/arXiv.2605.27724. URL https://arxiv.org/abs/2605.27724.

[104] Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, Yu Fang, Linxi Fan, Danfei Xu, Ruohan Zhang, Li Fei-Fei, Bowen Wen, Ajay Mandlekar, and Yuke Zhu. SimFoundry: Modular and automated scene generation for policy learning and evaluation. *arXiv preprint arXiv:2606.28276*, 2026. URL https://arxiv.org/abs/2606.28276.

[105] Chenghao Yin, Da Huang, Di Yang, Jichao Wang, Nanshu Zhao, Chen Xu, Wenjun Sun, Linjie Hou, Zhijun Li, Junhui Wu, Zhaobo Liu, Zhen Xiao, Sheng Zhang, Lei Bao, Rui Feng, Zhenquan Pang, Jiayu Li, Qian Wang, and Maoqing Yao. Genie Sim 3.0: A high-fidelity comprehensive simulation platform for humanoid robot. *arXiv preprint arXiv:2601.02078*, 2026. URL https://arxiv.org/abs/2601.02078.

[106] Qixin Liang and Zhongqing Han. IsaacIPC: Coupling high-fidelity simulation and realistic rendering for contact-rich robotic systems. *arXiv preprint arXiv:2605.24339*, 2026. URL https://arxiv.org/abs/2605.24339.

[107] Haoran Lu, Songling Liu, Yue Chen, Guo Ye, Mutian Shen, Shuyang Yu, Yu Xiao, Jihai Zhao, Shang Wu, Jianshu Zhang, Xiangtian Gui, Chuye Hong, Yuran Wang, Maojiang Su, Jiayi Wang, Ruihai Wu, Zhaoran Wang, and Han Liu. MagicSim: A unified infrastructure for executable embodied interaction. *arXiv preprint arXiv:2606.17511*, 2026. URL https://arxiv.org/abs/2606.17511.

[108] Joel Jang, Seonghyeon Ye, Zongyu Lin, Jiannan Xiang, Johan Bjorck, Yu Fang, Fengyuan Hu, Spencer Huang, Kaushil Kundalia, Yen-Chen Lin, Loic Magne, Ajay Mandlekar, Avnish Narayan, You Liang Tan, Guanzhi Wang, Jing Wang, Qi Wang, Yinzhen Xu, Xiaohui Zeng, Kaiyuan Zheng, Ruijie Zheng, Ming-Yu Liu, Luke Zettlemoyer, Dieter Fox, Jan Kautz, Scott Reed, Yuke Zhu, and Linxi Fan. DreamGen: Unlocking generalization in robot learning through video world models. *arXiv preprint arXiv:2505.12705*, 2025. doi: 10.48550/arXiv.2505.12705. URL https://arxiv.org/abs/2505.12705.

[109] Jie Zhang, Xiaoyue Chen, Anzhe Chen, Dayiheng Liu, Deqing Li, Gengze Zhou, Hale Yin, Haoqi Yuan, Haoyang Li, Jiahao Li, Jiazhao Zhang, Jingren Zhou, Kaiyuan Gao, Kun Yan, Lihan Jiang, Ningyuan Tang, Pei Lin, Qihang Peng, Shengming Yin, Tianhe Wu, Tianyi Yan, Xiao Xu, Yan Shu, Yanran Zhang, Ye Wang, Yi Wang, Yilei Chen, Yixian Xu, Yiyang Huang, Yuxiang Chen, Zekai Zhang, Zhendong Wang, Zixing Lei, Zhixuan Liang, Zihao Liu, Zikai Zhou, Chenxu Lv, Xiong-Hui Chen, and Chenfei Wu. Qwen-RobotWorld technical report: Unifying embodied world modeling through language-conditioned video generation. *arXiv preprint arXiv:2606.17030*, 2026. doi: 10.48550/arXiv.2606.17030. URL https://arxiv.org/abs/2606.17030.

[110] GigaWorld Team, Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Haoyun Li, Jiagang Zhu, Kerui Li, Mengyuan Xu, Qiuping Deng, Siting Wang, Wenkang Qin, Xinze Chen, Xiaofeng Wang, Yankai Wang, Yu Cao, Yifan Chang, Yuan Xu, Yun Ye, Yang Wang, Yukun Zhou, Zhengyuan Zhang, Zhehao Dong, and Zheng Zhu. GigaWorld-0: World models as data engine to empower embodied AI. *arXiv preprint arXiv:2511.19861*, 2025. doi: 10.48550/arXiv.2511.19861. URL https://arxiv.org/abs/2511.19861.

[111] Sabela Ramos, Sertan Girgin, Léonard Hussenot, Damien Vincent, Hanna Yakubovich, Daniel Toyama, Anita Gergely, Piotr Stanczyk, Raphael Marinier, Jeremiah Harmsen, Olivier Pietquin, and Nikola Momchev. RLDS: An ecosystem to generate, share and use datasets in reinforcement learning. *arXiv preprint arXiv:2111.02767*, 2021. URL https://arxiv.org/abs/2111.02767.

[112] 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 *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1678–1690. PMLR, 2022. URL https://proceedings.mlr.press/v164/mandlekar22a.html.

[113] Kaiyuan Chen, Letian Fu, David Huang, Yanxiang Zhang, Lawrence Yunliang Chen, Huang Huang, Kush Hari, Ashwin Balakrishna, Ted Xiao, Pannag R. Sanketi, John Kubiatowicz, and Ken Goldberg. Robo-DM: Data management for large robot datasets. In *2025 IEEE International Conference on Robotics and Automation*, 2025. doi: 10.1109/ICRA55743.2025.11128693. URL https://arxiv.org/abs/2505.15558.

[114] Linxuan Xin, Zheng Zhang, Zhiyi Pan, Jinfu Wei, Duan Gao, and Wei Gao. DreamPBR: Text-driven high-resolution SVBRDF generation with multimodal guidance. In *2025 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2025. doi: 10.1109/ICME59968.2025.11210202.

[115] Tencent Hunyuan3D Team. Hunyuan3D 2.1: From images to high-fidelity 3D assets with production-ready PBR material, 2025. URL https://arxiv.org/abs/2506.15442.

[116] Zibo Zhao, Zeqiang Lai, Qingxiang Lin, Yunfei Zhao, Haolin Liu, Shuhui Yang, Yifei Feng, et al. Hunyuan3D 2.0:

Scaling diffusion models for high resolution textured 3D assets generation. *arXiv preprint arXiv:2501.12202*, 2025. doi: 10.48550/arXiv.2501.12202.

[117] Google DeepMind. Gemini 3.1 Pro: Model card. Google DeepMind model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro.

[118] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3D shape collections. *ACM Transactions on Graphics*, 35(6):210:1–210:12, 2016. doi: 10.1145/2980179.2980238.

[119] Yuchen Li, Ujjwal Upadhyay, Habib Slim, Ahmed Abdelreheem, Arpit Prajapati, Suhail Pothigara, Peter Wonka, and Mohamed Elhoseiny. 3D CoMPaT: Composition of materials on parts of 3D things. In *Computer Vision–ECCV 2022*, pages 110–127, 2022. doi: 10.1007/978-3-031-20074-8_7.

[120] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019.

[121] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973. doi: 10.3138/FM57-6770-U75U-7727.

A Appendix

A.1 Task Details

Reasoning

Generalization

Open Drawer

Open the drawer, place the pill bottle inside, close the drawer, and return both arms to their home configuration.

ID uses seen drawers and pill bottles. OOD introduces unseen pill bottles, two distractors, and visual and embodiment changes.

Embodiment: ArtiXon Arm-6A, ARX R5 · **Data:** Auto · 300 demos

Fruit in Basket

Pick up the fruit, place it inside the basket, and return the arm to its home configuration.

ID uses seen fruits and baskets. OOD introduces unseen fruits, two distractors, and visual and embodiment changes; the basket must remain free of distractors.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

Bottle on Plate

Pick up the bottle, place it upright on the plate, and return the arm to its home configuration.

ID uses seen bottles and plates. OOD introduces unseen bottles and plates, two distractors, and visual and embodiment changes; the bottle must remain upright and the plate free of distractors.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

Object Beside Block

Pick up the green object, place it on the left/right/below/upper side of the reference block, and return the arm to its home configuration.

ID uses seen objects and blocks. OOD introduces unseen objects and blocks, three non-green distractors, and visual and embodiment changes; the instructed placement region must remain clear.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

Fruit in Microwave

Open the microwave, place the fruit inside, close the door, and return the arm to its home configuration.

ID uses a fixed microwave and seen fruits. OOD introduces unseen fruits, two distractors, and visual and embodiment changes; the microwave must remain free of distractors.

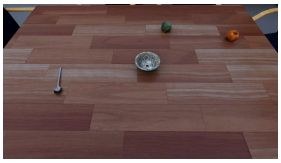
Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Cup on Plate

Orient the cup upright, move the plate to the table center, place the cup upright on the plate, and return both arms to their home configurations.

ID and OOD use the same cups and plates. OOD adds two distractors and introduces visual and embodiment changes; the cup must remain upright and the plate free of distractors.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Spoon in Bowl

Pick up the spoon, place it inside the bowl, and return the arm to its home configuration.

ID uses a fixed spoon and seen bowls. OOD introduces unseen bowls, two distractors, and visual and embodiment changes; the bowl must remain free of distractors.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

Visual Understanding

Classify by Shape

Sort the cube, cylinder, and sphere into the baskets with matching shapes, then return the arm to its home configuration.

ID uses seen instances of the three shapes. OOD replaces all three objects with unseen instances and introduces appearance randomization; the basket remains unchanged.

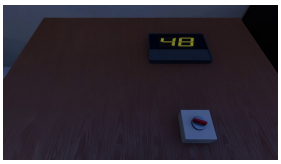
Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Classify by Color

Place each of the four objects on the visible region with the matching color, then return the arm to its home configuration.

ID uses seen objects in blue, green, red, and yellow. OOD replaces them with unseen objects of the same colors and introduces appearance randomization.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Adjust Temperature

Rotate the knob from the displayed initial temperature to the target temperature, then return the arm to its home configuration.

ID samples temperatures from 0–60. OOD expands the range to 0–99 and introduces appearance randomization while retaining the same adjustment rule.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

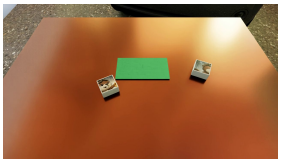
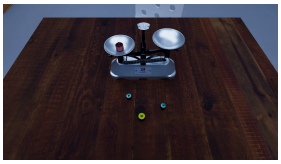


Image Puzzle

Place and align the two puzzle pieces on the green region to form a complete image, then return the arm to its home configuration.

ID uses seen image pairs. OOD introduces unseen image pairs together with visual and embodiment changes; the two-piece assembly rule remains unchanged.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Balance Scale

Select and place the required weights from 3-4 weights onto the lighter pan until the scale is balanced, then return the arm to its home configuration.

ID contains seen cases requiring one or two added weights. OOD introduces unseen weight configurations, including cases requiring three weights, together with visual and embodiment changes.

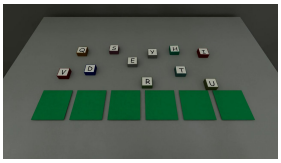
Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Rotate Book

Rotate the book on the table into the reader-facing orientation, then return both arms to their home configurations.

ID uses seen books. OOD replaces them with unseen books and introduces appearance randomization while retaining the same target orientation.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



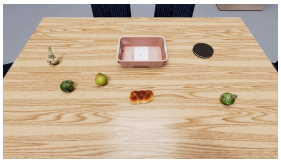
Spell Word

Select the required letter blocks and place them in order to spell the displayed word, then return the arm to its home configuration.

ID uses seen three-letter or four-letter words. OOD introduces unseen words and extends the task to five-letter and six-letter words, together with appearance randomization.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Language Understanding



Fruit to Basket

Place all three fruits in the pink basket, keep the non-fruit objects outside, and return the arm to its home configuration.

ID uses seen fruits and a fixed distractor pool. OOD replaces the fruits with unseen instances and introduces appearance randomization; the distractors and basket remain unchanged.

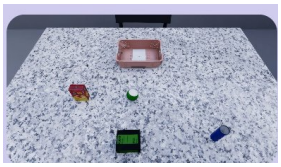
Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Teleop, Auto · 300 demos

AND Expression

Place all and only the objects satisfying both attributes in the instruction into the basket, then return the arm to its home configuration.

The ID training instructions specify a single color or shape. This setting evaluates unseen conjunctive color–shape expressions, together with appearance randomization, using the same object pool.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Confusing Instruction

Follow the operative placement instruction while ignoring the surrounding distractor sentences, then return the arm to its home configuration.

The ID training instructions contain only a direct single-attribute command. This setting adds irrelevant sentences before or after that command and introduces appearance randomization.

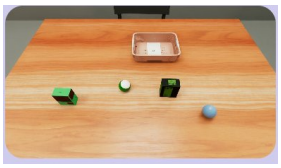
Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

OR Expression

Place all and only the objects satisfying either attribute in the instruction into the basket, then return the arm to its home configuration.

The ID training instructions specify a single color or shape. This setting evaluates unseen disjunctive expressions over colors and shapes, together with appearance randomization, using the same object pool.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



NOT Expression

Place every object except those excluded by the instruction into the basket, then return the arm to its home configuration.

The ID training instructions specify a positive single attribute. This setting evaluates unseen negated or exclusion expressions, together with appearance randomization, using the same object pool.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

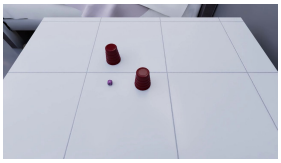
Specific Position

Pick the object at the instructed row and column, place it in the basket, and return the arm to its home configuration.

ID uses 2×4 and 3×3 object grids. OOD changes the layouts to 2×3 and 3×4 grids and introduces visual and embodiment changes while retaining the row–column instruction format.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Memory



Track Under Cup

Cover the die with the target cup, swap the two cups once, lift the cup that still covers the die, and return the arms home.

ID uses one seen cup type. OOD replaces both cups with an unseen matched pair and introduces visual and embodiment changes while retaining the cover–swap–reveal sequence.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Find Drawer

Place the block in the instructed drawer, close it, wait, reopen the same drawer, retrieve the block, close it again, and return the arm home.

ID uses a three-level drawer. OOD replaces it with a four-level drawer, updates the level names in the instruction, and introduces appearance randomization.

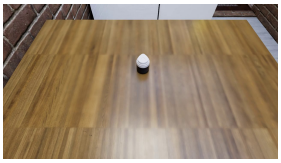
Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Press in Order

Press the colored buttons in the instructed order, then return the arm to its home configuration.

ID uses seen three-button and four-button sequences. OOD introduces unseen orders over the same button colors together with appearance randomization.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Press N Times

Stabilize the pump bottle, press the pump head the instructed number of times, and return both arms to their home configurations.

ID requests three to six presses. OOD requests two, seven, or eight presses and introduces appearance randomization while using the same bottle.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Teleop, Auto · 300 demos

Put Block Back

Move the instructed colored block to the central plate, keep it there briefly, return it to its original plate, and return the arm home.

ID uses three candidate source plates. OOD increases this set to four plates and introduces appearance randomization while retaining the round-trip requirement.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Manipulation

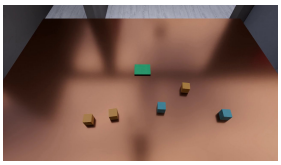
Precision Operation

Ring onto Rod

Pick up the ring, place it over the vertical rod, release it securely, and return the arm to its home configuration.

ID and OOD use the same rings and rod. OOD changes the visual environment while retaining the same asset.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

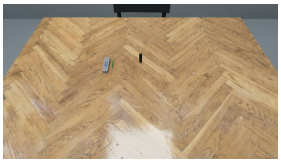


Stack Blocks

Place one block on the green region, stack two more blocks to form a three-layer tower, and return the arm to its home configuration.

ID and OOD use the same blocks, target region, and tower height. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Teleop, Auto · 300 demos

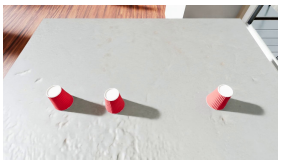


Plug in Charger

Pick up the USB stick, align it with the instructed left, middle, or right port, insert it fully, and return the arm home.

ID and OOD use the same USB stick, hub, ports, and instructions. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A, ARX R5, Franka · **Data:** Auto · 300 demos

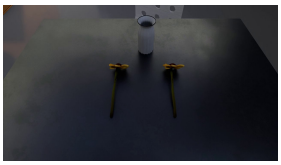


Cup Tower

Place two inverted cups side by side, stack the third cup across them to form a triangular tower, and return the arm home.

ID and OOD use the same three cups and assembly order. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Flowers in Vase

Pick up both flowers by their stems, insert them upright into the vase, and return the arm to its home configuration.

ID and OOD use the same flowers and vase pool. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

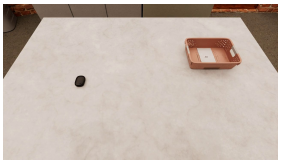
Bimanual Coordination

Object to Cup

Hold the cup with the right hand, place two objects into it with the left hand, set the cup back on the table, and return both arms home.

ID uses seen small objects and cups. OOD replaces the inserted objects with unseen instances and introduces visual changes while retaining the same cups.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

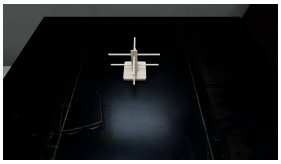


Hand Over

Pick up the object with the left hand, transfer it to the right hand, place it in the right-side basket, and return both arms home.

ID and OOD use the same object and basket pools. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



Glasses on Shelf

Pick up and reorient the glasses, hang them securely on the rack using both arms, and return both arms to their home configurations.

ID and OOD use the same glasses and rack. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

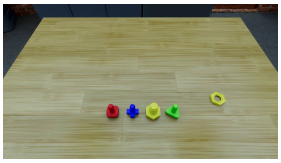
Hand Over Coin

Pick up the coin with one hand, transfer it to the other hand, place it on the coin rack, and return both arms home.

ID and OOD use the same coin and rack, whose initial position may be left, right, or center. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Tool Usage



Nut on Screw

Pick up the instructed nut, place it on the screw with matching color and shape, twist it into place, and return the arm home.

ID and OOD use the same four nut–screw pairs and matching rule. OOD changes only the visual environment.

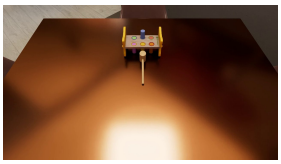
Embodiment: ArtiXon Arm-6A, ARX R5 · **Data:** Teleop · 300 demos

Sweep Trash

Hold the dustpan with one hand and the broom with the other, sweep the trash into the dustpan, and return both arms home.

ID and OOD use the same broom, dustpan, and trash pool. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos



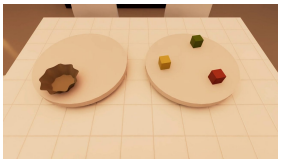
Whack-a-Mole

Pick up the hammer, strike the instructed colored peg below the panel, return the hammer to the table, and return the arm home.

ID and OOD use the same hammer, panel, and six colored pegs. OOD changes only the visual environment.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Dynamic Operation



Pick on Turntable

Pick the instructed colored block from the rotating turntable, place it on the rotating plate, and return the arm to its home configuration.

ID uses moderate turntable speeds. OOD introduces unseen slower or faster speeds together with visual changes while retaining the same blocks and plate.

Embodiment: ArtiXon Arm-6A · **Data:** Teleop · 300 demos

Mobile Manipulation Tasks

Move Bottle to Bin

Navigate to the table, grasp the bottle, carry it to the bin, place it inside, and return both arms to their home configuration.

ID and OOD use the same bottle pool and bin. OOD changes only the visual environment.

Embodiment: Quanta X1 · **Data:** Auto · 1,021 demos

Move Fruit to Basket

Navigate to the fruit table, grasp the fruit, carry it to the second table, place it in the basket, and return both arms home.

ID and OOD use the same fruit and basket pools. OOD changes only the visual environment.

Embodiment: Quanta X1 · **Data:** Auto · 1,021 demos

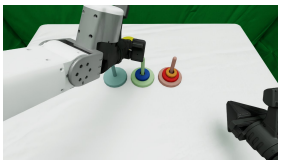
Bottle on Shelf

Navigate to the table, grasp the bottle, carry it to the wooden shelf, place it upright on the shelf, and return both arms home.

ID and OOD use the same bottle and shelf. OOD changes only the visual environment.

Embodiment: Quanta X1 · **Data:** Auto · 1,027 demos

Challenge



Move two to five rings from the left peg to the right peg, one ring at a time, without placing a larger ring on a smaller one.

The task is evaluated separately with two, three, four, or five rings and requires the prescribed legal intermediate states. It currently has no OOD evaluation.

Embodiment: ArtiXon Arm-6A · **Data:** Auto · 45 demos

Hanoi Tower

Press the colored buttons in the instructed sequence, releasing between repeated presses, then return the arm to its home configuration.

Sequence lengths of 4, 8, 12, 16, and 20 are evaluated separately over the same four buttons. This challenge currently has no OOD evaluation.

Embodiment: ArtiXon Arm-6A · **Data:** Auto · 4,000 demos

Press Buttons (Hard)

Build a 12-layer block tower on the green target region and return both arms to their home configuration.

Fifteen colored blocks are randomized around the target region. The task records stable completion of each layer and currently has no OOD evaluation.

Embodiment: ArtiXon Arm-6A · **Data:** Auto + Teleop · 14,600 stages + 1,398 demos

Stack Blocks (Hard)

A.2 Asset Preparation and AIG3D Implementation

Table 4 Asset collection scope and provenance. Counts use package-level units, so geometrically similar content from different sources may remain distinct packages. AIG3D additions are included in the tabletop rigid row, while deformable assets form a separate collection outside the rigid and articulated totals.

Collection	Count	Collection role	Coverage and major sources
Indexed rigid catalog	**96,480**	Source records indexed for candidate retrieval	Tabletop objects, furniture, fixtures, beds, and major appliances.
Tabletop rigid release	**12,485**	Packages prepared and accepted for tabletop task roles	44 leaf categories and 10 semantic families. Major sources: Objaverse/CanoVerse (7,946), Omni6DPose (2,090), and BlenderKit (786).
Articulated release	**670**	Packages prepared and accepted for articulated task roles	9 interaction groups. HSSD-derived (607), task-specific additions (61), and ReplicaCAD (2).

A.2.1 Source, Geometry, and Appearance Implementation

Format adapters convert GLB, Blender, USDZ, URDF, MJCF, and existing USD into a common OpenUSD representation while preserving recoverable hierarchy, transforms, geometry, texture and material bindings, source unit and axis evidence, and link–joint topology when present.

Geometry processing applies observe–canonicalize–re-observe at per-mesh and merged-object levels. It records hierarchy and transforms, bakes child-node transforms into a stable object frame, maps axes to canonical z-up, and converts reliable source units to meters. Checks cover metric extent, native and triangulated complexity, degenerate elements, open boundaries, non-manifold connectivity, thin structures, disconnected fragments, normals, and winding. Transform, orientation, topology, or resolution repairs are followed by world-space rechecks of bounds, origin, orientation, extent, and component structure. Temporary bounding-box centering and scale normalization affect only the annotation frame, not the metric asset or its size estimate.

Source material networks are normalized by optical behavior: standard opaque materials use OmniPBR's metallic–roughness representation, transmissive materials use OmniGlass, and coated materials use a clearcoat

model. The mapping reconciles diffuse–specular and base-color parameterizations, roughness/metallic ranges, OpenGL/DirectX normal conventions, packed occlusion–roughness–metallic channels, color spaces, UV transforms, alpha masking, and transmission. Trustworthy textures and scalar parameters are preserved, and material bindings are checked together with rendered whole-object appearance.

Generative PBR recovery uses available image, text, and geometry cues when curated candidates are insufficient (114, 115), and rendered candidates are evaluated as complete objects. A Hunyuan3D-family model supports the single- and multiview reconstruction branch (116); its conversion path covers simulation-oriented remeshing, retopology, and PBR regeneration on the final topology.

A.2.2 Semantic, Physical, and Acceptance Implementation

Object-level semantic and metric grounding. Four standardized RGB views span azimuth and modest elevation variation in the normalized annotation frame under neutral background and lighting. We use a Gemini 3.1-family multimodal model to produce the instance description and retrieval record (117); controlled vocabularies support role matching, while free-form text preserves long-tail distinctions.

Semantic orientation follows CanoVerse's normalized-pose formulation (28). Metadata, category context, and normalized views define discrete rotations relative to a category reference pose; reliable source orientation narrows the hypothesis set. When source scale is missing or untrustworthy, category- and function-matched anchors from GSO, ABO, and YCB (31, 35, 38) condition relative-scale estimates. Category ranges and anchor–target co-renderings validate relative-scale estimates; inconsistent cases are re-estimated or reviewed.

Part and task-conditioned interaction grounding. PartSAM generates candidate 3D regions from surface prompts (39). Confidence filtering and non-maximum suppression remove weak or redundant proposals; ShapeNet-Part, 3DCoMPaT, and PartNet provide reference part taxonomies and examples (118, 119, 120). Whole-object views, projected masks, local crops, the instance description, and part priors condition cross-view labeling. Equivalent parts are merged and reconciled using 3D containment and relative order along front–back and vertical axes, with selective review for residual ambiguity. Candidate tasks condition final interaction regions, and unsupported task–part assignments are left unassigned.

Physical, kinematic, and contact-material authoring. Collision proxies progress from analytic boxes, spheres, and capsules to a convex hull and then V-HACD/CoACD decomposition (40, 41). Candidate proxies are compared with render geometry and checked for valid collider composition. Mass and inertia use watertight-mesh volume when supported, convex-hull volume next, and oriented or axis-aligned bounds as conservative fallbacks; reliable source inertials are preserved.

For articulated assets, link–joint topology retained from URDF, MJCF, and USD is normalized into one object frame. Joint frames, signed axes, angular and linear limits, link collision shapes, inertial properties, and drives are authored together.

Multiview appearance, PBR evidence, category, and part semantics determine a coarse physical-material class for each relevant part. Texture-region statistics are used when maps exist, with authored constants as fallback. The class selects simulator-specific material-pair priors, including static and dynamic friction relative to the contact counterpart. Records are attached to corresponding collision parts, and low-confidence parameters remain bounded for downstream physical randomization.

Role-specific simulator acceptance. Common prechecks cover metric size, render–collision alignment, collider validity, and settling behavior. Manipulable rigid objects are grasped, lifted and held, transported, released, and observed through landing and post-placement stabilization; checks include residual penetration, pose and velocity stability, grasp retention, release separation, and support contact. Articulated objects and fixtures additionally exercise authored joints against declared axes and limits and inspect collision behavior and task-relevant contacts. Failed cases return to targeted geometry, collision, material, scale, or kinematic repair and repeat the applicable suite; unresolved cases are excluded.

A.3 Runtime Sampling and Replay Implementation Details

A.3.1 Candidate Generation, Coupling, and Selection

Candidate generation and geometric signatures. For each shared support region, the compiler constructs candidates with four complementary heuristic families: grid-family enumeration, free-rectangle search, shelf packing, and guillotine partitioning. Strip and point placements serve as fallbacks for domains not covered by the primary families. Every proposed cell is clipped against the footprint-aware feasible domain and checked against support boundaries, polygon holes, obstacles, and pairwise spacing. Exact geometric duplicates are removed with a deterministic signature comprising the sorted object names and each cell's four planar bounds, rounded to 10^{-4} m.

Quality and diversity selection. Candidate quality combines total and minimum cell area, minimum short-edge length, mean aspect score, minimum pairwise and obstacle gaps, and penalties for fallback placements. Selection begins with the highest-quality case, removes candidates below a quality floor, and fills the remaining pool by balancing quality against geometric distance from already selected cases. The distance normalizes corresponding cell-center displacements by the support diagonal and also accounts for layout-partition topology, preventing near-identical layouts from dominating the retained pool.

Overlap coupling. At runtime-plan construction, placement scopes are grouped by support surface and coordinate frame. Two scopes with disjoint object sets are connected when their footprint-expanded placement bounds overlap. Each nontrivial connected component is compiled into one joint case pool, so a reset selects a compatible assignment for all coupled objects rather than sampling them independently. Scopes outside these components retain their existing local domains.

Reset-time plan interpretation. The `RuntimeSamplingPlan` contains coupled cases, object-local domains, owner-first dependencies, and linked asset, paired-material, and lighting variants. At reset, the runtime selects a coupled case, samples dependent local poses, applies compatible variants, and instantiates the episode context.

Paired material variants. Each material-class variant couples a rendering-material template with an authored physical-material record and the bodies, collision parts, or links to which it applies. Only complete, compatible rendering–physical pairs are available for reset-time sampling. The runtime applies each pair as one update: dynamic bodies receive density-consistent mass and inertia where applicable, affected contact parts receive the associated material-pair friction parameters, and static or kinematic supports update their applicable contact properties. Identity-preserving appearance recipes remain visual and leave the physical-material record unchanged.

A.3.2 Replay Envelope and Stable-placement Details

Motion proxies and temporal refinement. Replay reconstructs robot-link motion by forward kinematics and follows task objects with their recorded rigid transforms. Dynamic bodies are approximated by sphere proxies. Each trajectory interval is probed at its quarter, midpoint, and three-quarter times and recursively subdivided until proxy deviation, joint motion, root translation, and root rotation remain within tolerance. The retained proxy paths are compressed with an error-bounded Douglas–Peucker procedure (121). Each line segment is then converted to a capsule whose radius includes the proxy radius, interpolation and simplification tolerances, and a clearance margin.

Stable-placement priors. Candidate orientations and support offsets are estimated offline with isolated PhysX drop–settle trials under small initial-orientation perturbations. A pose is retained when repeated trials show stable orientation, acceptable ground gap and penetration, limited drift, and consistent outcomes. Stable-orientation and support-offset priors are recomputed when the referenced asset geometry changes; assets without a valid prior use the aligned default pose.