RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation

Tianxing Chen [1 2 *] **Zanxin Chen** [3 4 *] **Baijun Chen** [5 *] **Zijian Cai** [3 4 *] **Yibin Liu** [6 *] **Zixuan Li** [4 *]
Qiwei Liang [4] **Xianliang Lin** [4] **Yiheng Ge** [7] **Zhenyu Gu** [8 9] **Weiliang Deng** [3 10] **Yubin Guo** [8 11] **Tian Nian** [3 4]
Xuanbing Xie [12] **Qiangyu Chen** [4] **Kailun Su** [4] **Tianling Xu** [13]
Guodong Liu [14 8] **Mengkang Hu** [1] **Huan-ang Gao** [14 2] **Kaixuan Wang** [1 2]
Zhixuan Liang [1 3] **Yusen Qin** [15 14] **Xiaokang Yang** [7] **Ping Luo** [1 16] **Yao Mu** [7 3]

Abstract

Simulation-based data synthesis has emerged as a powerful paradigm for enhancing real-world robotic manipulation. However, existing synthetic datasets remain insufficient for robust bimanual manipulation due to two key challenges: (1) the lack of an autonomous self-correcting mechanism to resolve execution failures in complex coordination tasks, and (2) the scarcity of diverse visual and spatial variations required to bridge the sim-to-real gap. To this end, we present **RoboTwin 2.0**, a scalable simulation framework that enables closed-loop, automated, large-scale generation of diverse and realistic data, along with unified evaluation protocols for dual-arm manipulation. Built upon RoboTwin-OD, a foundational library of 731 instances across 147 categories with rich semantic annotations, our framework integrates Multimodal Large Language Models (MLLMs) with simulation-in-the-loop verification. This integration forms an automated feedback mechanism that significantly boosts the success rate of expert task program generation. To enhance robust sim-to-real transfer, RoboTwin 2.0 incorporates structured domain randomization along five axes: clutter, lighting, background, tabletop height and language instructions, thereby maximizing data diversity. We instantiate this framework across 50 dual-arm tasks spanning five robot embodiments. Empirical evaluations demonstrate that Vision-Language-Action (VLA) models pre-trained on our synthetic data achieve a 3.6× improvement in few-shot real-world transfer (over a 10-demo baseline) and a 2.2× gain in zero-shot generalization. We release the data generator, benchmark, pre-collected dataset, and code to support scalable research in robust bimanual manipulation.

[1]HKU MMLab [2]Lumina EAI [3]Shanghai AI Lab [4]SZU [5]NJU [6]NEU [7]MoE key Lab of Artificial Intelligence, AI Institute, SJTU [8]TeleAI [9]FDU [10]SYSU [11]USTC [12]CSU [13]SUSTech [14]THU [15]D-Robotics [16]HKU-SH ICRC. Correspondence to: Ping Luo <pingluo@hku.hk>, Yao Mu <yao.mu@sjtu.edu.cn>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Bimanual robotic manipulation is essential for complex tasks such as collaborative assembly, tool use, and handovers, yet remains challenging due to tightly coupled inter-arm coordination and high environmental variability. While recent Vision–Language–Action (VLA) (Zheng et al., 2025; Bi et al., 2025; Team, 2025a; Intelligence et al., 2025a; Li et al., 2025; Cai et al., 2026; Wu et al., 2026; Team, 2025b; AgiBot-World-Contributors et al., 2025; Wen et al., 2025a; 2024; Lin et al., 2025; Liu et al., 2025) foundation models have demonstrated promising generalization capabilities, their performance is critically bottlenecked by the scarcity of high-quality, diverse, and physically valid dual-arm demonstration data. Collecting real-world bimanual demonstrations at scale is prohibitively expensive, particularly when broad coverage over tasks, object geometries, scene layouts, and robot embodiments is required. Synthetic data generation provides a principled approach for scaling multimodal supervision and enabling sim-to-real transfer. Among existing approaches, methods such as MimicGen (Mandlekar et al., 2023; Jiang et al., 2025) rely on human-in-the-loop demonstrations, which provide high-quality supervision but are costly and difficult to scale. As a result, expert code generation has emerged as a more scalable paradigm, synthesizing task execution programs without requiring human intervention. However, existing expert code generation pipelines typically operate in an open-loop manner (Wang et al., 2023; Mu et al., 2024), producing trajectories without closed-loop verification or corrective feedback. As a consequence, errors introduced during early stages of execution are not corrected and can propagate throughout the

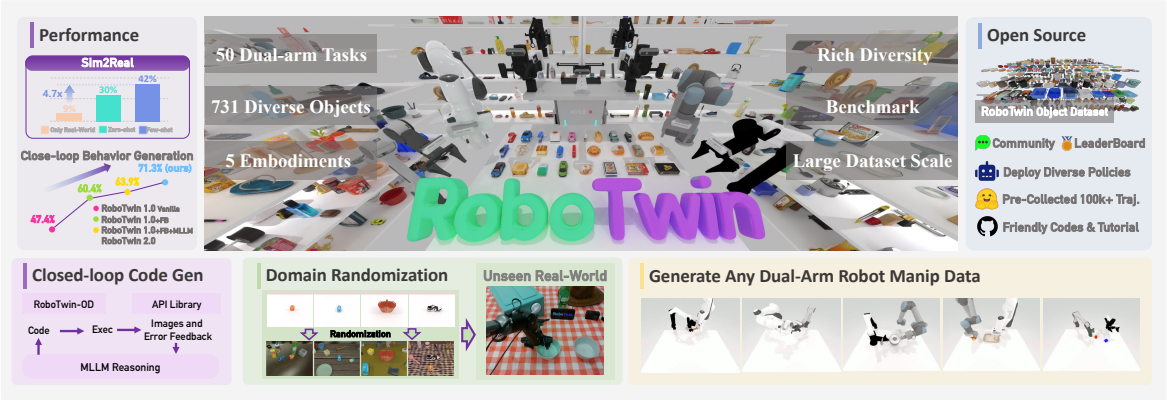

Figure 1. **RoboTwin 2.0** uses an MLLM-driven pipeline for automatic data synthesis and domain randomization to boost policy performance, and provides a 50-task bimanual benchmark with RoboTwin Object Dataset.

trajectory, a limitation that is particularly severe in bimanual manipulation, where small deviations in grasp pose, handover timing, or approach direction can rapidly cascade into task failure.

Beyond error propagation, this open-loop formulation further limits data quality and generalization along several interdependent dimensions. The lack of automated quality control admits execution failures and unstable grasps into training corpora, introducing high variance and misleading supervision signals. Meanwhile, domain randomization is often superficial, yielding visually clean and structurally homogeneous scenes that fail to capture real-world clutter, illumination variation, and instruction-conditioned ambiguity. In addition, most synthetic datasets ignore cross-embodiment diversity, overlooking the substantially different kinematics and grasp affordances of dual-arm platforms. For example, low-degree-of-freedom systems often favor lateral approach and grasping, whereas high-degree-of-freedom manipulators more readily support top-down grasping strategies. Without explicitly modeling such embodiment-specific constraints, synthetic pipelines are prone to generating executions that are infeasible or unstable for a given platform, thereby reducing the success rate and reliability of synthesized data.

In this work, we argue that scalable expert data generation for bimanual manipulation requires a fundamentally different design principle: closed-loop program synthesis under physical verification. Specifically, synthetic expert programs should be iteratively grounded in simulated execution, diagnosed via multimodal feedback, and refined until they satisfy explicit success criteria. We formalize this process as an iterative program refinement loop, where simulation-derived failure signals provide structured supervision to guide multimodal program regeneration. This perspective motivates RoboTwin 2.0, a unified simulation-based framework that couples multimodal large language models (MLLMs) with simulation-in-the-loop verification to synthesize physically valid, semantically grounded dual-arm task programs at scale. The framework comprises: (1) an automated expert pipeline that integrates MLLMs with simulation feedback to validate and iteratively refine task execution code; (2) comprehensive domain randomization over language, clutter, background textures, lighting, and tabletop layouts to improve sim-to-real transfer; and (3) an embodiment-aware adaptation module that conditions action generation on object spatial annotations and kinematic constraints to produce robot-specific manipulation candidates for heterogeneous dual-arm platforms.

Building on these components, we construct RoboTwin-OD, a semantically annotated asset library comprising 731 object instances across 147 categories, and release a large-scale, pre-collected dataset of expert trajectories spanning 50 bimanual tasks and five dual-arm robot embodiments. We further introduce a standardized benchmark for evaluating policy generalization under domain shift, covering variations in scene clutter, visual appearance, embodiment, and open-ended language goals.

Extensive evaluations demonstrate that the proposed paradigm yields consistent and substantial real-world performance gains. VLA models pre-trained on RoboTwin 2.0 data achieve a 3.6× improvement in few-shot real-world transfer and a 2.2× gain in zero-shot generalization compared to baselines trained on real-world data alone.

In summary, our main contributions are threefold:

(1) We formulate a closed-loop expert synthesis approach for bimanual manipulation that integrates MLLM-based program generation with simulation-in-the-loop physical verification, enabling the synthesis of trajectories that are both semantically coherent and physically executable.

(2) We introduce a large-scale and diverse bimanual manipulation dataset comprising 50 tasks, enabled by RoboTwin-OD, a richly annotated asset library, together with a unified data synthesis pipeline and domain randomization along

five complementary dimensions. In parallel, we present the RoboTwin Benchmark, a standardized evaluation suite designed to study policy generalization.

(3) We conduct extensive experiments to systematically evaluate representative manipulation policies on the RoboTwin Benchmark. In addition, real-world robotic experiments demonstrate that the diverse synthetic data generated by RoboTwin effectively supports sim-to-real transfer for bimanual manipulation.

2. Related Works

2.1. Datasets and Benchmarks for Robotic Manipulation

Simulation benchmarks have become essential for manipulation research. SAPIEN (Xiang et al., 2020) and ManiSkill2 (Gu et al., 2023) provide articulated objects with realistic physics and diverse tasks. Meta-World (Yu et al., 2020) established standardized multi-task evaluation, while CALVIN (Mees et al., 2022) and LIBERO (Liu et al., 2023) introduced language-conditioned manipulation with systematic generalization evaluation. Recent works further scale up complexity through generative scene creation (Nasiriany et al., 2024) and cross-platform unification (Geng et al., 2025), with GEMBench (Garcia et al., 2025) and Colosseum (Pumacay et al., 2024) introducing systematic perturbations for OOD evaluation. In the real-world domain, large-scale datasets (O'Neill et al., 2024; Bu et al., 2025; Wu et al., 2024; Ebert et al., 2021) provide diverse multi-embodiment trajectories for training generalizable policies. Despite these advances, most platforms rely on manual demonstrations or open-loop synthesis (Hua et al., 2024; Wang et al., 2023; Chen et al., 2026b; Wang et al., 2026; Chen et al., 2026a), limiting scalability and physically grounded generalization across diverse tasks and scenes. Building upon RoboTwin 1.0 (Mu et al., 2025), RoboTwin 2.0 integrates MLLM-driven closed-loop data generation with comprehensive domain randomization, offering significantly larger scale with over 700 assets, 11,000+ textures, 50 bimanual tasks, and support for five robot embodiments.

2.2. Robot Learning in Manipulation

While task-specific policies such as ACT (Zhao et al., 2023) and Diffusion Policy (Chi et al., 2023; Ke et al., 2024; Ze et al., 2024; Wang et al., 2024a) achieve strong performance on individual tasks, they struggle to generalize across embodiments. Subsequent works extend diffusion-based methods to geometry-aware manipulation, skill-based planning, and dexterous control (Chen et al., 2025a; Liang et al., 2024; 2023; 2025; Wen et al., 2025b). To achieve broader generalization, VLA models leverage large-scale pre-training: RT-1/2 (Brohan et al., 2022; 2023) pioneer this direction by unifying vision, language, and action, with subsequent works exploring generalist policies and heterogeneous pre-training (Team et al., 2024; Kim et al.; Wang et al., 2024b; Li et al., 2024; Liu et al., 2025). For bimanual manipulation, RDT-1B (Liu et al., 2024) trains on over one million episodes, while $\pi_0/\pi_{0.5}$ (Black et al., 2024; Intelligence et al., 2025b) employ flow matching with vision-language reasoning, achieving state-of-the-art open-world generalization. DexVLA (Wen et al., 2025a) and related works (Chen et al., 2025b; Fu et al., 2024) further advance dexterous and bimanual manipulation. Our work addresses the reliance on costly real-world demonstrations by contributing a digital-twin data generation pipeline with systematic domain randomization for training robust bimanual policies.

3. Method

We present RoboTwin 2.0, a unified framework designed to scale high-fidelity bimanual demonstrations via closed-loop synthesis. As shown in Figure 1, we leverage MLLMs to analyze execution videos from physics simulation and iteratively refine generated programs based on observed failure modes, enabling closed-loop trajectory optimization. Our framework comprises three core modules: (1) a closed-loop expert synthesis Pipeline that leverages simulation feedback for automatic program refinement; (2) a structured domain randomization engine for robust sim-to-real transfer; and (3) an embodiment-aware adaptation mechanism for generating trajectories across diverse robot kinematics.

3.1. Expert Code Generation via MLLMs and Simulation-in-the-Loop Feedback

We adopt a closed-loop architecture that couples code generation with multimodal execution feedback (Fig. 2), in contrast to pipelines that depend on manual priors or omit feedback (Hua et al.; Wang et al., 2023). The system comprises two agents: a code-generation agent that translates natural language instructions into executable programs, and a vision–language model observer that monitors execution in simulation, detects failures, and suggests corrections. Iterative integration of these signals proceeds until a predefined success criterion is met or a budget limit is reached, yielding robust, self-improving expert trajectories with minimal human supervision and enabling zero-shot dual-arm manipulation beyond primitive pick and place.

Input Specification. The code-generation agent is conditioned on four inputs: (1) a general API list; (2) example function calls; (3) a hierarchical constraint specification; and (4) task information. Each task is defined by a name (e.g., *Handover Block*) and a natural-language objective description. These components jointly guide the synthesis of Python code for task execution.

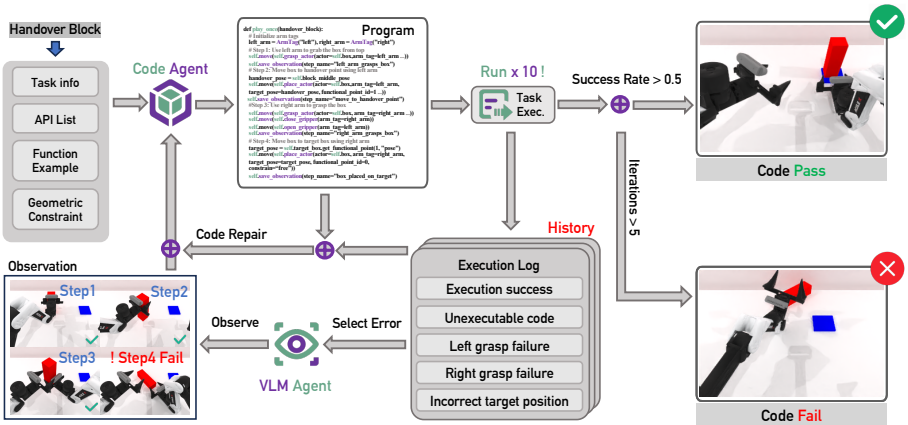

Figure 2. **Expert data generation pipeline.** Task-level code is generated by a Code Agent and iteratively validated in simulation, where execution feedback and visual observations are used for MLLM-based refinement until a target success rate is achieved.

Initial Code Generation. The code-generation agent synthesizes an initial Python program conditioned on the provided task inputs. It models the program synthesis process as a structured prediction problem over the space of available API calls, leveraging natural language understanding and few-shot prompting from task-specific examples. The generated code specifies a stepwise sequence of robot actions designed to accomplish the target manipulation objective.

Simulated Execution and Logging. Each iteration executes the program ten times to account for stochasticity in dynamics, control, and scene layout. After each batch, the system produces a structured log that records trial outcomes and labels failure cases by cause, such as unexecutable code, left/right grasp failure, or incorrect object placement.

Multimodal Observation and Error Localization. During execution, a vision–language model (VLM) monitors all ten trials and performs per-frame analysis to assess stepwise success and localize failures. Beyond temporal localization, the VLM attributes failure modes to flawed logic, incorrect API usage, or other systemic causes. This diagnosis enables repairs that target root causes rather than surface symptoms. Details are provided in Appendix A.10.4.

Code Repair and Iterative Refinement. The agent integrates execution logs and VLM diagnostics to edit failure-prone instructions, re-testing the program each iteration. The process stops upon meeting a success-rate threshold over ten runs in one iteration, or after five consecutive failures, producing expert-level code with minimal supervision and avoiding indefinite refinement.

3.2. Domain Randomization for Robust Robotic Manipulation

To enhance robustness to real-world variability, we randomize five dimensions: (1) cluttered distractors, (2) background textures, (3) lighting, (4) tabletop height, and (5) language instructions. This systematic augmentation broadens the training distribution and, critically, equips manipulation policies with stronger generalization to unseen scenes and instructions (Fig. 3a).

Scene Clutter. To improve robustness to environmental variation, we augment tabletop scenes with task-irrelevant distractors sampled from RoboTwin-OD. Object-level placement annotations enable a generic API for semantically valid insertion. Physical plausibility is enforced through collision-aware placement and precomputed volumes. To prevent spurious ambiguity, distractors that are visually or semantically similar to task-relevant objects are excluded during sampling. This procedure yields diverse yet unambiguous cluttered scenes for training.

Diverse Background Textures. We randomize tabletops and backgrounds using a curated texture library. We first collect 1,000 surface descriptions via LLM prompting and web search, then generate 20 images per description with Stable Diffusion v2 (Rombach et al., 2022) (20,000 images in total). Human-in-the-loop filtering yields 11,000 high-quality textures. This library is used in simulation to increase visual diversity and mitigate overfitting to clean synthetic scenes (Fig. 3b). We further show in Appendix A.16 that the distribution of our texture library closely matches that of real-world textures.

Lighting Variation. Real scenes vary in color temperature, source type, count, and placement, altering appearance and reflections and challenging vision-based manipulation. We randomize light color, type, intensity, and position within physically plausible ranges. As shown in Fig. 3a (second row), changes in color temperature markedly affect appearance (e.g., warm vs. cool light on a shoe). Training under these variations improves robustness to real-world illumination shifts.

Tabletop Heights. We uniformly randomize table height

(a) Visualization of Domain Randomization **(b) Texture Library**

Figure 3. **Visualization of domain randomization and our texture library.**

within a plausible range in simulation, strengthening the policy's robustness to variations in table height.

Trajectory-Level Diverse Language Instructions. We employ an MLLM to generate task templates and multiple object descriptions that capture geometry, appearance, and part-level attributes. Each task and object has several alternative phrasings that can be combined; for each trajectory, we sample from these pools to compose the instruction. For *Move Can Pot*, the template "Use a to place A to the left of B" may yield "Use left arm to place sauce can to the left of gray kitchenpot" or "Use left arm to place white plastic lid sauce can to the left of kitchenpot for boiling and cooking." This augmentation produces a large, linguistically varied instruction set and improves generalization to unseen language and scene configurations (Appendix A.22, A.23).

3.3. Embodiment-Aware Grasp Adaptation

Differences in DoF and kinematics result in different reachable workspaces and preferred strategies for a given task. In grasping a can, Franka often adopts an overhead approach, whereas the lower-DoF Piper favors lateral grasps; consequently, required approaches vary across embodiments (Fig. 6). To model this variation, we annotate each object with candidate manipulation poses that span multiple grasp axes and approach directions, capturing both manipulation diversity and robot-specific preferences. We further improve feasibility via angular perturbations oriented to high-reachability directions. For each object, candidate grasps are generated from preferred operation directions, randomized pose perturbations, and parallel motion-planning attempts. In experiment A.2, embodiment-aware augmentation raises automated data-collection success by 8.3% on average, with gains concentrated on low-DoF platforms (Aloha-AgileX +13.7%, Piper +22.7%, ARX-X5 +5.6%), while high-DoF arms (Franka, UR5) exhibit minimal change, consistent with greater kinematic flexibility.

4. Experiment

We design experiments to evaluate RoboTwin 2.0 along four dimensions: (1) automating the generation of high-quality expert code for novel manipulation tasks; (2) establishing RoboTwin 2.0 as a standardized benchmark for policy generalization across tasks, scenes, and embodiments; (3) improving policy robustness to environmental variation via diversified training data; and (4) demonstrating sim-to-real transfer, whereby RoboTwin 2.0 enables deployment on real robots and confers strong policy generalization to variations in scene composition and appearance.

Table 1. **Overall performance on tasks shared by RoboTwin 1.0 and 2.0.** Per-task success rates are in Appendix 11.

Method	ASR	Top5-ASR	CR-Iter	Token
R1.0 Vanilla	47.4%	57.6%	1.00	1236.6
R1.0 + FB	60.4%	71.4%	2.46	1190.4
R1.0 + MM FB	63.9%	74.2%	2.42	1465.0
R2.0 Vanilla	62.1%	68.0%	1.00	**569.4**
R2.0 + FB	66.7%	73.6%	1.89	581.6
R2.0 + MM FB	**71.3%**	**78.6%**	**1.76**	839.7

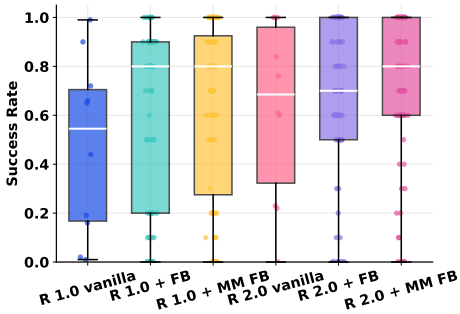

Figure 4. **Success Rate Distribution.**

4.1. Evaluation of Automated Expert Code Generation

To assess whether closed-loop generation improves the quality and efficiency of expert programs, we evaluate the system on 10 manipulation tasks, each specified by a natural-language instruction. For each configuration, the code-generation agent emits multiple candidate programs that are executed in simulation to capture stochasticity in dynamics, control, and perception; task success is defined as the mean success rate over all executions. Performance is measured by **ASR** (average success rate), **Top5-ASR** (average of top-5 success rate), **CR-Iter** (average refinement

Table 2. **Subset of RoboTwin 2.0 benchmark**. Full results are in Appendix A.17. Tasks details are in Appendix A.18.

Simulation Task	RDT		Pi0		ACT		DP		DP3	
	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Adjust Bottle	81%	**75%**	90%	56%	97%	23%	97%	0%	**99%**	3%
Beat Block Hammer	**77%**	**37%**	43%	21%	56%	3%	42%	0%	72%	8%
Blocks Ranking RGB	3%	0%	**19%**	**5%**	1%	0%	0%	0%	3%	0%
Blocks Ranking Size	0%	0%	**7%**	**1%**	0%	0%	1%	0%	2%	0%
Click Alarmclock	61%	12%	63%	11%	32%	4%	61%	5%	**77%**	**14%**
Click Bell	80%	**9%**	44%	3%	58%	3%	54%	0%	**90%**	0%
Dump Bin Bigbin	64%	32%	83%	24%	68%	1%	49%	0%	**85%**	**53%**
Grab Roller	74%	43%	96%	**80%**	94%	25%	**98%**	0%	**98%**	2%
...					...					
Move Pillbottle Pad	8%	0%	21%	**1%**	0%	0%	1%	0%	**41%**	0%
Average (in %)	34.5	13.7	46.4	**16.3**	29.7	1.7	28.0	0.6	**55.2**	5.0

iterations), and **Token** (average tokens in generated code). Results for RoboTwin 1.0 and 2.0 are reported in Table 1 under *Vanilla* (one-shot generation), *FB* (feedback-based repair using execution logs), and *MM FB* (multimodal feedback with vision–language diagnostics). Per-task success rates are provided in Appendix 11.

Across all settings, multimodal feedback improves performance. In RoboTwin 1.0, ASR increases from 47.4% (Vanilla) to 63.9% (MM FB); in RoboTwin 2.0, from 62.1% to 71.3%. Top5-ASR also rises, indicating disproportionate gains for the best candidate programs. RoboTwin 2.0 converges faster than 1.0 (CR-Iter 1.76 vs. 2.42 under MM FB) and reduces token usage, especially in Vanilla (569.4 vs. 1236.6), reflecting more concise initial code. Figure 4 further shows that feedback narrows the success-rate distribution and raises the median; with multimodal feedback, RoboTwin 2.0 exhibits a compact distribution centered above 80%. Overall, three findings emerge: (1) vision–language feedback not only detects failures but also guides precise repairs; (2) architectural improvements in RoboTwin 2.0 accelerate convergence and reduce token usage; and (3) combining symbolic execution logs with perceptual diagnostics yields more reliable, semantically aligned expert data. Together, these results validate the effectiveness of our closed-loop, self-improving code generation architecture. Detailed setups, metric definitions, and additional analyses are provided in Appendix A.10.

4.2. RoboTwin 2.0 Benchmark

We present the RoboTwin 2.0 Benchmark for evaluating policy performance. Results on 50 RoboTwin tasks are reported in Appendix A.17, and Tab. 2 summarizes the average performance of RGB-based policies across evaluation settings. To assess generalization, we evaluate all 50 tasks on the Aloha–AgileX dual-arm platform. For each task, we train on 50 clean expert demonstrations and test with 100 rollouts under two conditions: *Easy* (no domain randomization) and *Hard* (domain randomization with clutter, lighting, texture, and height variation). We report the success rate as the metric of few-shot adaptability and robustness. Appendix A.11 visualizes the benchmark setup, and Appendix A.5 details all training protocols. As shown in Tab. 2, under the Easy condition, ACT and DP perform substantially worse than the pretrained models RDT and Pi0 (29.7%, 28.0% vs. 34.5%, 46.4%), indicating that vision–language–action pretraining supplies strong priors that enable rapid policy learning from 50 demonstrations. Compared with RGB-based policies, DP3 attains the best few-shot performance in Easy (55.2%), highlighting the contribution of 3D information; however, its high success rate is partly attributable to idealized simulated depth and clean background segmentation. From the clean to the randomized Hard setting, all methods degrade: the non-pretrained models ACT, DP, and DP3 drop to 1.7%, 0.6%, and 5.0%, respectively, whereas RDT and Pi0 remain higher at 13.7% and 16.3%. These results indicate that vision–language–action pretraining provides useful priors for scene generalization and improves robustness to environmental variation, yet robustness under domain shift remains a key challenge. Together with Secs. 4.3 and 4.4, these findings highlight RoboTwin 2.0 as a complementary dataset and benchmark for systematic evaluation.

4.3. Assessing the Impact of RoboTwin 2.0 on Policy Robustness

We evaluate whether domain-randomized data in RoboTwin 2.0 enhances robustness to environmental perturbations. RDT and Pi0 are pre-trained on 9,600 expert trajectories drawn from 32 tasks (300 per task) under clean and domain-randomized settings. Off-the-shelf pretrained RDT and Pi0 are included as reference models without further fine-tuning. Generalization is examined on five unseen tasks using 50 clean demonstrations per task for single-task training and subsequent fine-tuning. ACT, DP, RDT, and Pi0 are then evaluated under domain-randomized conditions in previously unseen environments to quantify robustness. Detailed

Table 3. **Evaluating the Impact of RoboTwin 2.0 on Policy Robustness.**

Simulation Tasks	ACT	DP	RDT	RDT +Clean	RDT +Rand.	Pi0	Pi0 +Clean	Pi0 +Rand.
Stack Bowls Two	0.0%	0.0%	30.0%	8.0%	**49.0%**	41.0%	55.0%	**62.0%**
Pick Dual Bottles	0.0%	0.0%	13.0%	12.0%	**17.0%**	12.0%	**15.0%**	7.0%
Move Can Pot	4.0%	0.0%	12.0%	13.0%	**18.0%**	21.0%	**35.0%**	22.0%
Place Object Basket	0.0%	0.0%	**17.0%**	9.0%	6.0%	2.0%	8.0%	**22.0%**
Place Shoe	0.0%	0.0%	7.0%	9.0%	**30.0%**	6.0%	6.0%	**18.0%**
Open Laptop	0.0%	0.0%	32.0%	21.0%	**35.0%**	46.0%	33.0%	**50.0%**
Press Stapler	6.0%	0.0%	24.0%	21.0%	**27.0%**	29.0%	26.0%	**31.0%**
Turn Switch	2.0%	1.0%	15.0%	**24.0%**	16.0%	**23.0%**	21.0%	21.0%
Average	2.0%	0.1%	18.8%	14.6%	24.8%	22.5%	24.9%	29.1%

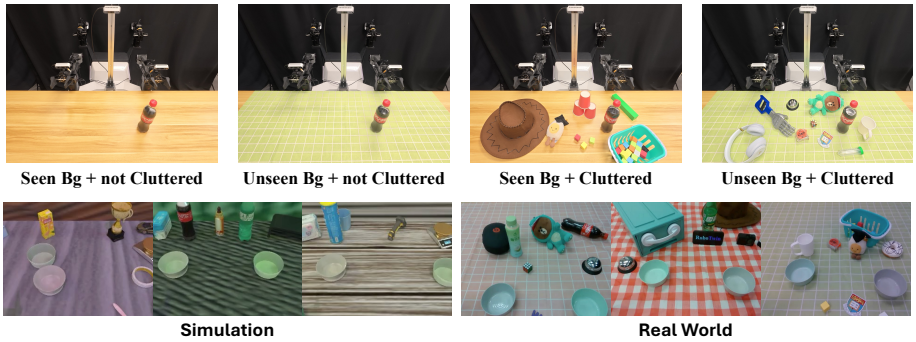

Figure 5. **Real-World Evaluation Configurations and Sim-Real Comparison.**

configurations are provided in Appendix A.4 and A.5.

As shown in Table 3, fine-tuning on clean data yields negligible gains in average success rate relative to pretrained baselines, indicating that non-randomized data do not improve robustness to environmental variation. This further suggests that the low simulated performance of pretrained VLA models is not attributable to a real-to-sim gap, since adding clean simulated data produces no clear benefit. In contrast, pretraining with RoboTwin 2.0 data substantially improves generalization: RDT and Pi0 attain relative gains of 31.9% and 29.3%, respectively. Notably, these gains persist even when downstream training uses only clean, non-randomized data, demonstrating that domain-randomized pretraining with RoboTwin 2.0 confers robustness to visual and spatial variation. Consequently, models pretrained with RoboTwin 2.0 adapt to new tasks without additional augmentation or complex scene variation.

4.4. Evaluation on Sim-to-Real Performance

To assess RoboTwin 2.0's impact on real-world robustness, we evaluate four bimanual tasks: *Stack Bowls*, *Handover Block*, *Pick Bottle*, and *Click Bell*. All experiments use RDT as the policy backbone on the COBOT-Magic dual-arm platform. We compare three training regimes: (1) 10 real-world demonstrations collected in clean tabletop environments; (2) the same demonstrations augmented with 1,000 domain-randomized synthetic trajectories generated under clutter, varied lighting, and diverse backgrounds; and (3) a synthetic-only model trained on the 1,000 synthetic trajectories. To improve robustness to camera jitter and calibration error, we apply random 3D perturbations to simulated camera poses (position and orientation), with translation magnitude bounded by 1 cm. We evaluate under four configurations: clean vs. cluttered tabletops crossed with seen vs. unseen backgrounds (Fig. 5). The synthetic-only model excludes seen backgrounds during training, so the corresponding entries in Table 4 are omitted. This setup tests whether RoboTwin 2.0 supports robust generalization without additional real-world data from visually complex scenes.

RoboTwin 2.0 augmentation yields substantial robustness improvements in real-world bimanual policies. In the few-shot setting, which combines 1,000 domain-randomized synthetic trajectories with 10 real demonstrations, the average success rate across all evaluation configurations increases by 24.4%, with per-configuration gains of 13.5%, 27.5%, 23.5%, and 33.0%. In the zero-shot setting trained solely on synthetic data, the two unseen-background configurations improve by 21.0% and 20.5%. These gains are larger in visually complex scenes, indicating particular effectiveness under challenging conditions. We attribute the improvements to three factors: (1) the high visual and physical fidelity of RoboTwin 2.0, which enables direct

Table 4. **Real-World Experiment Results.** We conduct controlled experiments on 4 dual-arm tasks: *Stack Bowls*, *Handover Block*, *Pick Bottle*, and *Click Bell*, each evaluated under 4 different settings.

Real World Task	Background Type	Cluttered or Not	10 Clean Real	10 Clean Real + 1k RoboTwin 2.0	1k RoboTwin 2.0 (Zero-shot)
Stack Bowls	Seen	False	22.0%	**64.0%**	/
		True	12.0%	**58.0%**	/
	Unseen	False	10.0%	50.0%	**60.0%**
		True	12.0%	**56.0%**	52.0%
Handover Block	Seen	False	40.0%	**48.0%**	/
		True	**16.0%**	12.0%	/
	Unseen	False	36.0%	**56.0%**	**56.0%**
		True	0.0%	**36.0%**	20.0%
Pick Bottle	Seen	False	20.0%	**36.0%**	/
		True	8.0%	**40.0%**	/
	Unseen	False	4.0%	**26.0%**	10.0%
		True	8.0%	28.0%	**32.0%**
Click Bell	Seen	False	**36.0%**	24.0%	/
		True	20.0%	**56.0%**	/
	Unseen	False	12.0%	**24.0%**	20.0%
		True	16.0%	**48.0%**	14.0%
Average	Seen	False	29.5%	**43.0%** $_{+13.5\%}$	/
		True	14.0%	**41.5%** $_{+27.5\%}$	/
	Unseen	False	15.5%	**39.0%** $_{+23.5\%}$	36.5% $_{+21.0\%}$
		True	9.0%	**42.0%** $_{+33.0\%}$	29.5% $_{+20.5\%}$

Table 5. **Real robot performance with different sim–real mixtures.** Tasks: CB = Click Bell, PEC = Place Empty Cup, SB2 = Stack Bowls Two.

	CB	PEC	SB2	***Avg.***
50 real	15.0%	10.0%	0.0%	8.3%
300 sim + 0 real	35.0%	10.0%	0.0%	15.0%
300 sim + 10 real	40.0%	25.0%	10.0%	25.0%
300 sim + 30 real	55.0%	35.0%	20.0%	36.7%
300 sim + 50 real	65.0%	50.0%	25.0%	46.7%

sim-to-real transfer; (2) domain-randomized synthetic data that conditions policies on environmental variations absent from clean real-world demonstrations; and (3) large-scale simulation-based randomization that increases scene diversity and strengthens cross-scene transfer. Overall, these results suggest that a small amount of real-world data, when combined with rich RoboTwin 2.0 simulation, is sufficient to substantially narrow the sim-to-real gap.

Beyond the above RDT-based dual-arm experiments, we further study how RoboTwin 2.0 interacts with larger amounts of real data and a different policy backbone. Specifically, we vary the ratio of real-world and simulated demonstrations and evaluate the resulting pi0 policies on real robots. On three tasks (*Click Bell*, *Place Empty Cup*, *Stack Bowls Two*), we compare: (i) 50 real demonstrations collected in a fixed scene; (ii) 300 domain-randomized simulated demonstrations; and (iii) mixtures of 300 simulated demonstrations with 0, 10, 30, or 50 real demonstrations. Each model is evaluated over 20 trials per task in unseen real scenes. As shown in Table 5, simulation-only training already outperforms real-only training, and adding modest amounts of real data on top of RoboTwin 2.0 simulation leads to consistent gains, reaching 46.7% average success with 300 sim + 50 real demonstrations. Taken together with the RDT results, these findings indicate that RoboTwin 2.0 not only enables strong zero-shot transfer, but also continues to provide sizeable benefits in more data-rich real-world regimes and across different policy architectures.

5. Conclusion

This paper presents RoboTwin 2.0, a scalable simulation framework for data-driven bimanual manipulation that unifies closed-loop expert synthesis, structured domain randomization, and standardized evaluation. By formulating expert data generation as an iterative program refinement process grounded in physical verification, RoboTwin 2.0 enables the automatic synthesis of physically executable and semantically grounded dual-arm trajectories at scale. The framework is supported by RoboTwin-OD, a richly annotated object library with 731 assets across 147 categories, and a benchmark of 50 dual-arm tasks spanning five robot embodiments. Together with systematic randomization over scene layout, visual appearance, geometry, and language. Extensive experiments show that vision–language–action models pre-trained on RoboTwin 2.0 achieve substantial improvements in both few-shot and zero-shot real-world transfer, demonstrating that closed-loop synthetic data can effectively substitute for large-scale real-world demonstrations in bimanual manipulation. Future work includes supporting more diverse object categories and open-world mobile manipulation settings, enabling scalable data synthesis and evaluation beyond tabletop scenarios.

Acknowledgements

This paper is partially supported by the General Research Fund of Hong Kong No.17208825.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

AgiBot-World-Contributors, Bu, Q., Cai, J., Chen, L., Cui, X., Ding, Y., Feng, S., Gao, S., He, X., Hu, X., Huang, X., Jiang, S., Jiang, Y., Jing, C., Li, H., Li, J., Liu, C., Liu, Y., Lu, Y., Luo, J., Luo, P., Mu, Y., Niu, Y., Pan, Y., Pang, J., Qiao, Y., Ren, G., Ruan, C., Shan, J., Shen, Y., Shi, C., Shi, M., Shi, M., Sima, C., Song, J., Wang, H., Wang, W., Wei, D., Xie, C., Xu, G., Yan, J., Yang, C., Yang, L., Yang, S., Yao, M., Zeng, J., Zhang, C., Zhang, Q., Zhao, B., Zhao, C., Zhao, J., and Zhu, J. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems, 2025. URL https://arxiv.org/abs/2503.06669.

Bi, H., Tan, H., Xie, S., Wang, Z., Huang, S., Liu, H., Zhao, R., Feng, Y., Xiang, C., Rong, Y., et al. Motus: A unified latent action world model. *arXiv preprint arXiv:2512.13030*, 2025.

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Bu, Q., Cai, J., Chen, L., Cui, X., Ding, Y., Feng, S., Gao, S., He, X., Huang, X., Jiang, S., et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025.

Cai, J., Cai, Z., Cao, J., Chen, Y., He, Z., Jiang, L., Li, H., Li, H., Li, Y., Liu, Y., et al. Internvla-a1: Unifying understanding, generation and action for robotic manipulation. *arXiv preprint arXiv:2601.02456*, 2026.

Chen, B., Wan, W., Chen, T., Guo, X., Xu, C., Qi, Y., Zhang, H., Wu, L., Xu, T., Li, Z., et al. Univtac: A unified simulation platform for visuo-tactile manipulation data generation, learning, and benchmarking. *arXiv preprint arXiv:2602.10093*, 2026a.

Chen, T., Mu, Y., Liang, Z., Chen, Z., Peng, S., Chen, Q., Xu, M., Hu, R., Zhang, H., Li, X., et al. G3flow: Generative 3d semantic flow for pose-aware and generalizable object manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1735–1744, 2025a.

Chen, T., Wang, K., Yang, Z., Zhang, Y., Chen, Z., Chen, B., Dong, W., Liu, Z., Chen, D., Yang, T., et al. Benchmarking generalizable bimanual manipulation: Robotwin dual-arm collaboration challenge at cvpr 2025 meis workshop. *arXiv preprint arXiv:2506.23351*, 2025b.

Chen, T., Wang, Y., Li, M., Qin, Y., Shi, H., Li, Z., Hu, Y., Zhang, Y., Wang, K., Chen, Y., et al. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design. *arXiv preprint arXiv:2603.01229*, 2026b.

Chi, C., Xu, Z., Feng, S., Cousineau, E., Du, Y., Burchfiel, B., Tedrake, R., and Song, S. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, pp. 02783649241273668, 2023.

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , and Vedaldi, A. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Ebert, F., Yang, Y., Schmeckpeper, K., Bucher, B., Georgakis, G., Daniilidis, K., Finn, C., and Levine, S. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Fu, Z., Zhao, T. Z., and Finn, C. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

Garcia, R., Chen, S., and Schmid, C. Towards generalizable vision-language robotic manipulation: A benchmark and llm-guided 3d policy. In *International Conference on Robotics and Automation (ICRA)*, 2025.

Geng, H., Wang, F., Wei, S., Li, Y., Wang, B., An, B., Cheng, C. T., Lou, H., Li, P., Wang, Y.-J., et al. Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025.

Gu, J., Xiang, F., Li, X., Ling, Z., Liu, X., Mu, T., Tang, Y., Tao, S., Wei, X., Yao, Y., et al. Maniskill2: A unified benchmark for generalizable manipulation skills. In *The Eleventh International Conference on Learning Representations*, 2023.

Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., and Yin, J. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*, 2022.

Hua, P., Liu, M., Macaluso, A., Lin, Y., Zhang, W., Xu, H., and Wang, L. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. In *8th Annual Conference on Robot Learning*.

Hua, P., Liu, M., Macaluso, A., Lin, Y., Zhang, W., Xu, H., and Wang, L. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. *arXiv preprint arXiv:2410.03645*, 2024.

Intelligence, P., Amin, A., Aniceto, R., Balakrishna, A., Black, K., Conley, K., Connors, G., Darpinian, J., Dhabalia, K., DiCarlo, J., et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025a.

Intelligence, P., Black, K., Brown, N., Darpinian, J., Dhabalia, K., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Galliker, M. Y., Ghosh, D., Groom, L., Hausman, K., Ichter, B., Jakubczak, S., Jones, T., Ke, L., LeBlanc, D., Levine, S., Li-Bell, A., Mothukuri, M., Nair, S., Pertsch, K., Ren, A. Z., Shi, L. X., Smith, L., Springenberg, J. T., Stachowicz, K., Tanner, J., Vuong, Q., Walke, H., Walling, A., Wang, H., Yu, L., and Zhilinsky, U. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025b. URL https://arxiv.org/abs/2504.16054.

Jiang, Z., Xie, Y., Lin, K., Xu, Z., Wan, W., Mandlekar, A., Fan, L., and Zhu, Y. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

Ke, T.-W., Gkanatsios, N., and Fragkiadaki, K. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

Kim, M. J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E. P., Sanketi, P. R., Vuong, Q., et al. Openvla: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*.

Lan, Z., Jiang, Y., Wang, R., Xie, X., Zhang, R., Zhu, Y., Li, P., Yang, T., Chen, T., Gao, H., et al. Autobio: A simulation and benchmark for robotic automation in digital biology laboratory. *arXiv preprint arXiv:2505.14030*, 2025.

Li, H., Yang, S., Chen, Y., Tian, Y., Yang, X., Chen, X., Wang, H., Wang, T., Zhao, F., Lin, D., et al. Cronusvla: Transferring latent motion across time for multi-frame prediction in manipulation. *arXiv preprint arXiv:2506.19816*, 2025.

Li, Q., Liang, Y., Wang, Z., Luo, L., Chen, X., Liao, M., Wei, F., Deng, Y., Xu, S., Zhang, Y., et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

Liang, Z., Mu, Y., Ding, M., Ni, F., Tomizuka, M., and Luo, P. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. In *International Conference on Machine Learning*, pp. 20725–20745. PMLR, 2023.

Liang, Z., Mu, Y., Ma, H., Tomizuka, M., Ding, M., and Luo, P. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16467–16476, 2024.

Liang, Z., Mu, Y., Wang, Y., Chen, T., Shao, W., Zhan, W., Tomizuka, M., Luo, P., and Ding, M. Dexhanddiff: Interaction-aware diffusion planning for adaptive dexterous manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1745–1755, 2025.

Lin, M., Ding, P., Wang, S., Zhuang, Z., Liu, Y., Tong, X., Song, W., Lyu, S., Huang, S., and Wang, D. Hif-vla: Hindsight, insight and foresight through motion representation for vision-language-action models, 2025. URL https://arxiv.org/abs/2512.09928.

Liu, B., Zhu, Y., Gao, C., Feng, Y., Liu, Q., Zhu, Y., and Stone, P. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Liu, J., Chen, H., An, P., Liu, Z., Zhang, R., Gu, C., Li, X., Guo, Z., Chen, S., Liu, M., Hou, C., Zhao, M., alex Zhou, K., Heng, P.-A., and Zhang, S. Hybridvla: Collaborative diffusion and autoregression in a unified vision-language-action model, 2025. URL https://arxiv.org/abs/2503.10631.

Liu, S., Wu, L., Li, B., Tan, H., Chen, H., Wang, Z., Xu, K., Su, H., and Zhu, J. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

Mandlekar, A., Nasiriany, S., Wen, B., Akinola, I., Narang, Y., Fan, L., Zhu, Y., and Fox, D. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*, 2023.

Mees, O., Hermann, L., Rosete-Beas, E., and Burgard, W. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

Mu, Y., Chen, T., Peng, S., Chen, Z., Gao, Z., Zou, Y., Lin, L., Xie, Z., and Luo, P. Robotwin: Dual-arm robot benchmark with generative digital twins (early version). *arXiv preprint arXiv:2409.02920*, 2024.

Mu, Y., Chen, T., Chen, Z., Peng, S., Lan, Z., Gao, Z., Liang, Z., Yu, Q., Zou, Y., Xu, M., et al. Robotwin: Dual-arm robot benchmark with generative digital twins. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025.

Nasiriany, S., Maddukuri, A., Zhang, L., Parikh, A., Lo, A., Joshi, A., Mandlekar, A., and Zhu, Y. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024.

O'Neill, A., Rehman, A., Maddukuri, A., Gupta, A., Padalkar, A., Lee, A., Pooley, A., Gupta, A., Mandlekar, A., Jain, A., et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Pumacay, W., Singh, I., Duan, J., Krishna, R., Thomason, J., and Fox, D. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*, 2024.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., and Ma, S. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, June 2022.

Team, O. M., Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Kreiman, T., Xu, C., et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Team, R. Rdt2: Enabling zero-shot cross-embodiment generalization by scaling up umi data, September 2025a. URL https://github.com/thu-ml/RDT2.

Team, X. E. I. Mimo-embodied: X-embodied foundation model technical report, 2025b. URL https://arxiv.org/abs/2511.16518.

Wang, C., Fang, H., Fang, H.-S., and Lu, C. Rise: 3d perception makes real-world robot imitation simple and effective. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2870–2877. IEEE, 2024a.

Wang, K., Chen, T., Liu, J., Su, H., Zhu, S., Wang, M., Li, Z., Chen, Y., Gao, H.-a., Qin, Y., et al. Manitwin: Scaling data-generation-ready digital object dataset to 100k. *arXiv preprint arXiv:2603.16866*, 2026.

Wang, L., Chen, X., Zhao, J., and He, K. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers. In *Neurips*, 2024b.

Wang, Y., Xian, Z., Chen, F., Wang, T.-H., Wang, Y., Fragkiadaki, K., Erickson, Z., Held, D., and Gan, C. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation, 2023.

Wen, J., Zhu, M., Zhu, Y., Tang, Z., Li, J., Zhou, Z., Li, C., Liu, X., Peng, Y., Shen, C., and Feng, F. Diffusionvla: Scaling robot foundation models via unified diffusion and autoregression. *arXiv preprint arXiv:None*, 2024.

Wen, J., Zhu, Y., Li, J., Tang, Z., Shen, C., and Feng, F. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*, 2025a.

Wen, J., Zhu, Y., Li, J., Zhu, M., Tang, Z., Wu, K., Xu, Z., Liu, N., Cheng, R., Shen, C., Peng, Y., Feng, F., and Tang, J. Tinyvla: Toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025b. doi: 10.1109/LRA.2025.3544909.

Wen, W., Xue, X., Li, Y., Gu, P., and Xu, J. Code similarity detection using ast and textual information. *International Journal of Performability Engineering*, 15(10): 2683, 2019.

Wu, K., Hou, C., Liu, J., Che, Z., Ju, X., Yang, Z., Li, M., Zhao, Y., Xu, Z., Yang, G., et al. Robomind: Benchmark on multi-embodiment intelligence normative data

for robot manipulation. *arXiv preprint arXiv:2412.13877*, 2024.

Wu, W., Lu, F., Wang, Y., Yang, S., Liu, S., Wang, F., Ma, S., Sun, H., Wang, Y., Qiu, Z., Xiong, H., Wang, Z., Zhou, S., Ren, Y., Zhang, K., Yu, H., Zhao, J., Zhu, Q., Cheng, R., Li, Y.-L., Huang, Y., Zhu, X., Shen, Y., and Zheng, K. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692v1*, 2026.

Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11097–11107, 2020.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

Ze, Y., Zhang, G., Zhang, K., Hu, C., Wang, M., and Xu, H. 3d diffusion policy. *arXiv e-prints*, pp. arXiv–2403, 2024.

Zhao, T. Z., Kumar, V., Levine, S., and Finn, C. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Zheng, J., Li, J., Wang, Z., Liu, D., Kang, X., Feng, Y., Zheng, Y., Zou, J., Chen, Y., Zeng, J., et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

Zhu, Y., Wong, J., Mandlekar, A., Martín-Martín, R., Joshi, A., Nasiriany, S., and Zhu, Y. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

A. Appendix

A.1. The Use of Large Language Models

No large language models or AI-assisted tools were used at any stage of this work, including writing, coding, data processing, analysis, figure generation, or conclusions. All text and code were authored, reviewed, verified, and tested solely by the authors, who take full responsibility for the content and any errors.

A.2. Evaluating Efficiency with and without Adaptive Grasping

We evaluate embodiment-aware grasp augmentation by measuring automated data-collection success on 50 RoboTwin 2.0 tasks across five robot embodiments. As shown in Table 6, our pipeline outperforms the RoboTwin 1.0 baseline, which lacks diverse grasping and candidate augmentation, with an average gain of 8.3%. Benefits are concentrated on lower-DoF platforms: Aloha-AgileX, Piper, and ARX-X5 improve by 13.7%, 22.7%, and 5.6%, respectively. High-DoF arms with large reachable workspaces, such as Franka and UR5 (7-DoF), show little change, consistent with sufficient kinematic flexibility. These results indicate that augmentation supplies additional feasible grasps that mitigate planning constraints on low-DoF manipulators. Full results are reported in Appendix A.19.

A.3. Benchmarking RoboTwin 2.0 Against Existing Datasets

We compare RoboTwin 2.0 against existing benchmarks and datasets across several key dimensions, including the number of supported tasks, the presence of domain randomization, support for automatic data generation, and compatibility with vision-language-action (VLA) model training and evaluation. The comparison is summarized in Table 7.

A.4. Domain Randomization Setting

Domain randomization in all experiments includes cluttered scenes, random lighting, table height variation (up to 3 cm), unseen language instructions, and randomized background textures.

A.5. Policies Training Details

We adopt joint angles as the model's prediction target, formulating action prediction as joint-angle regression.

RDT in experiment 4.3 was finetuned for 100,000 steps with a batch size of 16 per GPU on 8 GPUs, and all single-task fine-tuning was conducted for 10,000 steps with a batch size of 16 per GPU on 4 GPUs. In all cases, we initialize from the publicly released RDT pretrained weights.

Pi0 in experiment 4.3 was pretrained for 100,000 steps with a batch size of 32, and all fine-tuning was performed for 30,000 steps using the same batch size with LoRA-based fine-tuning. In all cases, we initialize from the publicly released Pi0 pretrained weights.

ACT was trained under a unified setup with a chunk size of 50, batch size of 8, and single-GPU training for 6,000 epochs. During deployment, we applied `temporal_agg` for temporal aggregation to improve execution stability.

DP was trained for 600 epochs with a batch size of 128 and a planning horizon of 8.

DP3 was trained for 3,000 epochs with a batch size of 256, using a planning horizon of 8 and a point cloud resolution of 1,024, with precise segmentation of the background and tabletop.

A.6. Ablation on Domain Randomization Factors

To understand which domain randomization factors contribute most to policy robustness, we conduct an ablation study and report the results in Table 8. For each factor (background, clutter, table height, lighting), we disable only that factor's randomization, collect 100 trajectories for training, and then evaluate the resulting policy under the full domain randomized setting. This isolates the contribution of each factor while keeping all other conditions fixed. The table reports success rates (in %) for four representative tasks, with ACT and DP shown as "ACT / DP" in each cell. Disabling background or clutter randomization leads to the largest performance drops, while turning off height or lighting randomization results in smaller but still noticeable degradation. When all randomization factors are disabled, the success rate becomes very low, indicating that our visual domain randomization, especially in background and clutter, is a primary driver of learning robust behaviors.

A.7. Robustness under Dynamic Scene Changes

In Sec. 4.4, our notion of robustness refers to robustness to substantial variations in scene layout and background appearance, rather than arbitrary external disturbances. To make this explicit, we conduct additional real robot experiments that focus on dynamically changing scenes. On three real-world tasks with pi0, we train two policies: one using 50 real demonstrations collected in a single fixed scene ("50 real"), and one using 300 domain randomized simulated demonstrations from RoboTwin 2.0 combined with the same 50 real demonstrations ("300 sim + 50 real"). During evaluation, we deliberately introduce dynamic perturbations by randomly moving scene objects and changing the tabletop background between episodes. As shown in Table 9, the policy trained only on fixed scene real data completely fails in this setting (0.0% success on all three tasks), whereas the

Table 6. **Overall Performance Comparison between RoboTwin 1.0 and RoboTwin 2.0.**

Method	Aloha-AgileX	Piper	Franka	UR5	ARX-X5	*Average*
RoboTwin 1.0	65.1%	2.4%	**67.3%**	**57.6%**	68.6%	52.2%
RoboTwin 2.0	**78.8%**	**25.1%**	67.2%	57.1%	**74.2%**	**60.5%**
Difference	**+13.7%**	**+22.7%**	**-0.1%**	**-0.5%**	**+5.6%**	**+8.3%**

Table 7. **Comparison of RoboTwin 2.0 with previous manipulation benchmarks and datasets.**

Benchmark & Dataset	#Tasks	Domain Randomization	Auto Data Generation	VLA Model Train & Eval
Meta-world (Yu et al., 2020)	50	×	✓	×
Robosuite (Zhu et al., 2020)	9	×	×	×
RoboCasa (Yu et al., 2020)	25	✓	×	×
Maniskill2 (Gu et al., 2023)	20	×	✓	×
AutoBio (Lan et al., 2025)	16	×	✓	✓
RoboTwin 1.0 (Mu et al., 2025)	14	×	✓	✓
RoboTwin 2.0 (ours)	50	✓	✓	✓

policy trained with RoboTwin 2.0 augmented data maintains a 26.7% average success rate. This large gap under dynamic scene changes supports our claim that RoboTwin 2.0 significantly improves policy robustness to realistic visual and spatial variations.

Our object-centric, embodiment-agnostic data generation framework enables seamless deployment across a wide range of dual-arm robotic systems. The pipeline supports flexible embodiment configurations, allowing arbitrary combinations of heterogeneous manipulators and relative arm placements. This design ensures compatibility with diverse hardware setups and facilitates extensibility to future robotic platforms.

A.8. Support for Flexible Embodiment Combinations

To execute high-success-rate manipulation trajectories across different embodiments (see Section 3.3), we integrate Curobo, a high-performance, GPU-accelerated motion planner that enables efficient and reliable planning under varied kinematic constraints.

Currently, our framework supports five robotic arms—Franka, Piper, UR5, ARX-X5, and Aloha-AgileX—along with multiple gripper types, including the Panda gripper and WSG gripper. As shown in Fig. 8, we demonstrate successful task executions across a variety of dual-arm pairings, highlighting RoboTwin 2.0's ability to scale to heterogeneous robot configurations and its readiness for future real-world deployment.

A.9. Improvements of RoboTwin 2.0 over RoboTwin 1.0 Policy Codebase

We first quantify the architectural impact of RoboTwin 2.0 in a one-shot generation without code repair and iterative refinement. Table 10 shows that RoboTwin 2.0 yields significantly shorter programs (569.4 vs. 1236.6 tokens), with reduced prompt length and higher structural similarity to human-written code. Crucially, it enables dual-arm parallelism via a unified API abstraction, which is absent in RoboTwin 1.0.

These improvements stem from the structured prompting and geometric API modularization designed into RoboTwin 2.0. Higher AST similarity (+21.06%), CodeBERT similarity (+1.08%), and Unixcoder alignment (+5.97%) indicate that RoboTwin 2.0 not only reduces code size but also improves semantic clarity and functional alignment.

In addition, RoboTwin 2.0 integrates a **VLM observer**, a plug-and-play module triggered only when execution fails. To quantify its overhead, we estimated VLM usage via the Kimi API (assuming each image = 1,024 tokens) over three representative tasks: the average cost was 6,295 input tokens and 599 output tokens, totaling **6,894 tokens**. While this introduces moderate overhead, the VLM enables RoboTwin 2.0 to catch and correct errors invisible to execution logging, significantly enhancing robustness and overall task success. Importantly, the observer remains optional and can be disabled when prioritizing token efficiency.

A.10. Experimental Details and Metric Definitions for Code Generation

We use the *DeepSeek-V3* model for program synthesis and the *moonshot-v1-32k-vision-preview* model for multimodal error localization and verification. These models were selected for their strong performance in language reasoning and visual understanding while maintaining efficiency suitable for large-scale iterative refinement. The success rate of the i-th program is computed as $R_i = \frac{1}{M}\sum_{j=1}^{M} s_{i,j}$, and

Figure 6. **Embodiment-aware Adaption**.

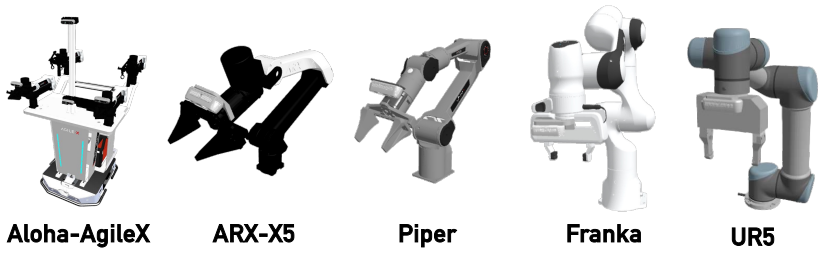

Figure 7. **Five RoboTwin 2.0 Embodiments**.

Table 8. **Ablation on domain randomization factors.** Each entry shows success rate (in %) for ACT / DP. "BG" denotes background randomization.

Task	BG↓	Clutter↓	Height↓	Light↓	All Rand.↓
Adjust Bottle	50 / 49	64 / 91	94 / 89	95 / 98	23 / 0
Beat Block Hammer	3 / 2	4 / 39	3 / 23	7 / 65	3 / 0
Move Can Pot	31 / 4	28 / 29	53 / 37	41 / 34	4 / 0
Stack Bowls Two	14 / 3	35 / 64	29 / 60	36 / 81	0 / 0
Average	24.5 / 14.5	32.8 / 55.8	44.8 / 52.3	44.8 / 69.5	7.5 / 0

the final success rate for a given task under a specific system variant is then defined as $R_{\text{task}} = \frac{1}{N}\sum_{i=1}^{N} R_i$.

Figure 8. **Heterogeneous Dual-Arm Control via Object-Centric Manipulation.**

A.10.1. Metric Definitions

We report the following metrics across all tasks:

ASR is the average of R_task across all 10 tasks. It reflects overall task performance across all generated programs.

Top5-ASR is the mean success rate computed using only the top 5 highest-performing programs per task. This metric estimates system potential under a best-of-selection strategy.

CR-Iter indicates the average number of feedback iterations required per task before reaching a success rate above 50% or exhausting the iteration budget.

Token denotes the average number of tokens of policy code generated by the language model per task. It serves as a proxy for computational cost and LLM inference budget.

These metrics jointly evaluate both the reliability and efficiency of the expert data generation pipeline under varying conditions of feedback, model capability, and refinement strategy.

A.10.2. Task-Specific Performance Comparison on Code Generation

We compare the code generation success rates of RoboTwin 2.0 and RoboTwin 1.0 across all tasks. As shown, RoboTwin 2.0 consistently matches or outperforms the baseline on the majority of tasks, demonstrating the effectiveness of our multimodal feedback and refinement pipeline.

A.10.3. Per-task Success Rates of Code Generation

We report the success rates of all tasks in Tab. 12.

A.10.4. Multimodal Observation and Error Localization

To further investigate the capability of the VLM observer, we manually curated a dataset of 130 execution sequences, including 101 failed trials and 29 successful trials. Each sequence consists of the natural language task instruction, a series of visual observations, and policy code. This dataset enables us to evaluate both binary error detection and fine-grained error localization.

Error Detection. The VLM observer was first tasked with evaluating whether a robotic execution successfully completed the instructed task. The confusion matrix is as follows: TP = 16, FP = 61, TN = 40, FN = 13. The derived performance metrics are: Accuracy = 0.431, Precision = 0.208, Recall = 0.552, and F1-score = 0.302. These results suggest that while the VLM observer is reasonably sensitive in identifying execution failures (recall), its tendency to over-predict errors leads to a high false positive rate.

Table 9. **Real robot performance under dynamic scene changes.** Success rate in %.

	Click Bell	Place Empty Cup	Stack Bowls Two	Average
50 real	0.0%	0.0%	0.0%	0.0%
300 sim + 50 real	40.0%	30.0%	10.0%	26.7%

Table 10. **Code Generation Efficiency and Quality Comparison.** Evaluation of prompt and generated code characteristics, along with code similarity metrics (AST Structural Similarity, CodeBERT, Unixcoder cosine similarity) against expert-written code, for RoboTwin 1.0 and RoboTwin 2.0 in zero-shot generation. The VLM observer cost is also reported for RoboTwin 2.0.

Metric	**RoboTwin 1.0**	**RoboTwin 2.0**
Prompt Token Length ↓	5901.0	**4719.1**
Code Token Length ↓	1236.6	**569.4**
Parallelism Control ↑	✗	✓
AST Similarity (Wen et al., 2019) ↑	23.72%	**44.78%**
CodeBLEU Similarity (Ren et al., 2020) ↑	17.18%	**18.53%**
CodeBERT Similarity (Feng et al., 2020) ↑	97.72%	**98.80%**
Unixcoder Similarity (Guo et al., 2022) ↑	76.24%	**82.21%**
Avg. VLM Token Cost (per observation)	–	**6894**

Error Localization. For failed executions, we further tested whether the VLM observer could correctly localize the program step or code snippet that led to failure. Among the 40 true negative samples (correctly identified failed trials), the observer successfully pinpointed the failure step in 12 cases, yielding a localization accuracy of 30%. This indicates that, while the observer shows promising diagnostic ability, there remains significant room for improvement in reliable error attribution.

Case Study. We illustrate one successful case of VLM observer reasoning. Given a task instruction and five visual observations, the observer produced a step-by-step analysis:

Step 1: The left arm successfully picks up the mug from the left side.
Step 2: The left arm successfully places the mug at the middle position.
Step 3: The right arm successfully picks up the mug from the middle position.
Step 4: The right arm attempts to hang the mug on the rack but fails.
Step 5: The right arm is in the process of moving away after attempting to hang the mug.
The overall task was not successfully completed. The failure occurred in Step 4, where the right arm attempted to hang the mug on the rack but did not succeed. The mug is not properly positioned on the rack. The error message in the code indicates an AttributeError: 'list' object has no attribute 'p'. This error occurs in the line where the target_point is being set to middle_target_pose.p. The issue is likely due to the fact that middle_target_pose is a list, and lists do not have an attribute 'p'. The code should be modified to correctly access the position data from the middle_target_pose list.
To fix the error, the code should be updated to access the position data correctly. For example, if middle_target_pose is a list containing the position and orientation, the position data should be accessed using middle_target_pose[0] instead of middle_target_pose.p. Additionally, the failure in Step 4 could be due to the incorrect position or orientation of the mug when attempting to hang it on the rack. The code should be reviewed to ensure that the mug is positioned and oriented correctly before executing the hanging motion.

Failure Modes. Despite these successes, we also observed cases where the VLM observer fails. Misclassifications primarily occur in failed executions: the observer may hallucinate correct outcomes or overlook subtle orientation errors. For instance, in a stacking task where boxes are misaligned, the observer sometimes marks the execution as successful if all boxes are placed on top of each other. Similarly, in tasks requiring fine-grained pose accuracy (e.g., placing shoes in the correct orientation), small but critical errors may be ignored. Furthermore, failures stemming from invisible factors—such as incorrect grasp axis parameters—remain challenging for purely vision-based observers to diagnose.

A.11. Visualization of RoboTwin 2.0 Benchmark Setting

We visualize the simulation settings of the RoboTwin 2.0 benchmark in Fig. 9. All models are trained on 50 clean (non-randomized) demonstrations per task (blue). For evaluation, the *Easy* setting also uses clean environments, while the *Hard* setting employs domain-randomized environments (green).

Table 11. Task-Specific Performance Comparison between RoboTwin 2.0 and RoboTwin 1.0. R1.0/R2.0: RoboTwin 1.0 / 2.0. Bold numbers indicate the best result for each task.

Task	R1.0 Vanilla	R1.0 + FB	R1.0 + MM FB	R2.0 Vanilla	R2.0 + FB	R2.0 + MMFB
beat_block_hammer	16%	48%	**56%**	23%	34%	53%
handover_block	2%	41%	45%	17%	**50%**	27%
pick_diverse_bottles	**65%**	**65%**	64%	60%	60%	62%
pick_dual_bottles	99%	99%	**100%**	**100%**	**100%**	**100%**
place_container_plate	66%	79%	**91%**	84%	84%	82%
place_dual_shoes	19%	22%	**25%**	0%	2%	22%
place_empty_cup	90%	90%	**100%**	61%	61%	85%
place_shoe	72%	90%	90%	**100%**	**100%**	**100%**
stack_blocks_three	1%	2%	4%	76%	76%	**82%**
stack_blocks_two	44%	68%	64%	**100%**	**100%**	**100%**

Table 12. **Per-task success rates of our proposed R2.0 + MM FB algorithm on all RoboTwin 2.0-supported tasks.**

Task	Rate	Task	Rate	Task	Rate	Task	Rate
Adjust Bottle	100%	Beat Block Hammer	53%	Blocks Ranking Rgb	80%	Blocks Ranking Size	80%
Click Alarmclock	0%	Click Bell	10%	Dump Bin Bigbin	0%	Grab Roller	74%
Handover Block	27%	Handover Mic	0%	Hanging Mug	0%	Lift Pot	40%
Move Can Pot	30%	Move Pillbottle Pad	50%	Move Playingcard Away	90%	Move Stapler Pad	100%
Open Laptop	0%	Open Microwave	0%	Pick Diverse Bottles	62%	Pick Dual Bottles	100%
Place A2B Left	50%	Place A2B Right	60%	Place Bread Basket	0%	Place Bread Skillet	0%
Place Can Basket	0%	Place Cans Plasticbox	100%	Place Container Plate	82%	Place Dual Shoes	22%
Place Empty Cup	85%	Place Fan	70%	Place Burger Fries	100%	Place Mouse Pad	100%
Place Object Basket	0%	Place Object Scale	80%	Place Object Stand	90%	Place Phone Stand	0%
Place Shoe	100%	Press Stapler	0%	Put Bottles Dustbin	0%	Put Object Cabinet	0%
Rotate Qrcode	80%	Scan Object	0%	Shake Bottle	0%	Shake Bottle Horizontally	0%
Stack Blocks Three	82%	Stack Blocks Two	100%	Stack Bowls Three	20%	Stack Bowls Two	30%
Stamp Seal	20%	Turn Switch	0%	***Avg Success Rate***	***43.34%***		

A.12. Visualization of Rendered Meshes and Collision Shapes of Objects in RoboTwin-OD

To illustrate how complex geometries are handled, we visualize both the rendered meshes and collision shapes for five representative objects in Fig. 11, including those with internal cavities such as mugs. Their holes and fine structural details are preserved in the final assets. Specifically, after generating each object via AIGC, we perform convex decomposition in Blender and then merge the resulting parts to obtain smooth, physically stable collision bodies that are compatible with the SAPIEN simulator.

A.13. SAPIEN-IPC–Based Tactile Data Acquisition Setup

We build our simulated tactile data acquisition pipeline on top of the SAPIEN-IPC framework, which provides a flexible and scalable environment for generating high-fidelity contact interactions.

A.14. User Study on Instruction Naturalness

To further evaluate the naturalness of our LLM-generated instructions, we conduct a user study on the 567 objects with textual descriptions in RoboTwin-OD. For each object, we first generate 15 candidate instructions using an LLM, as described in Sec. 3.1. We then ask 5 volunteers to write additional natural-language descriptions for each object, yielding a total of 20 descriptions per object (15 LLM-generated and 5 human-written).

A separate group of 5 volunteers is then presented with the 20 descriptions for each object and asked to select the one that is *most likely* written by a human. Since 5 out of 20 descriptions are human-authored, the ground-truth proportion of human-written descriptions in the pool is 25%. Table 13 reports the success rate of each volunteer in correctly identifying a human-written description.

The average success rate is 23.8%, which is very close to the 25% ground-truth proportion. This near-chance performance indicates that human annotators have difficulty distinguishing between human-written and LLM-generated instructions, suggesting that our instruction pool is linguistically close to natural human descriptions rather than being obviously synthetic or overly templated.

A.15. Control frequency and controller implementation

RoboTwin 2.0 is built on the SAPIEN simulator, where each simulation step corresponds to 0.004 seconds of real time. In our default data collection setting, we record one sample every 15 simulation steps, which gives an effective sampling rate of approximately 16.67 Hz (this rate can be adjusted via configuration parameters). The temporal spacing of

Figure 9. **Visualization of RoboTwin 2.0 Benchmark Settings.**

Table 13. Success rate of volunteers in identifying human-written instructions among mixed pools of human- and LLM-generated descriptions. Chance level corresponds to 25%.

	Vol. 1	Vol. 2	Vol. 3	Vol. 4	Vol. 5	Average
Success rate (%)	21.0	19.9	26.1	27.2	22.9	23.8

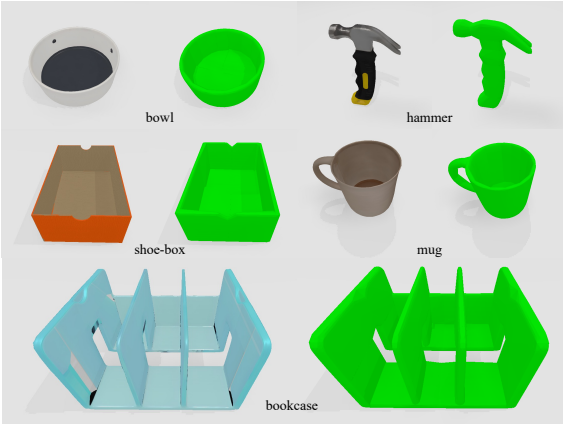

Figure 10. **Visualization of RoboTwin-OD Objects.**

policy inputs and outputs therefore, naturally aligns with this sampling interval.

For policy execution in both simulation and on real robots, we interpolate the predicted actions to match the underlying low-level controller frequencies. When the policy outputs joint space actions, we use TOPP-based interpolation to generate time parameterized joint trajectories; when the policy outputs end effector poses, we use trajectory planning to construct smooth Cartesian motion. In our setup, the controller runs at 250 Hz in simulation and at 30 Hz on the real robot, so the interpolated trajectories bridge the gap between the policy output rate ($\sim$ 16.67 Hz) and the higher frequency control loops.

A.16. Texture distribution vs. real-world backgrounds

To evaluate whether our 11k background texture library reasonably approximates real-world background statistics, we perform an analysis in a pretrained CLIP feature space. Specifically, we extract ViT-B/32 CLIP (Radford et al., 2021) embeddings (OpenAI weights) for two sets of backgrounds: (i) our 11k textures from the background library and (ii) 5,640 real textures from the Describable Textures Dataset (DTD) (Cimpoi et al., 2014), a widely used real-world texture benchmark containing in-the-wild images annotated with human-describable attributes (e.g., "banded", "wrinkled", "cracked"). We then (1) approximate each set by a Gaussian in CLIP space and compute a CLIP-FID between our textures and DTD, and (2) evaluate coverage by, for each real (DTD) texture, finding its nearest neighbor in our pool and recording the cosine similarity between their CLIP embeddings.

As summarized in Tab. 14, the CLIP-FID between our texture library and DTD is essentially zero (-0.36, within numerical noise), indicating that the first- and second-order statistics of our textures are almost identical to those of real textures in this feature space. Moreover, the nearest-neighbor cosine similarity from DTD to our library exhibits a high mean (0.839) and median (0.848), with the 90th percentile reaching 0.914, meaning that at least 90% of real textures have a very close counterpart in our pool. Even the worst-case real sample still attains a cosine similarity of 0.544 to its closest neighbor, so we do not observe any real textures that are nearly orthogonal to our manifold in CLIP space. Overall, these results indicate that our curated texture library is not an artificial or overly narrow distribution, but instead provides a high-coverage, globally well-matched approximation to real-world background statistics.

A.17. Full RoboTwin 2.0 Benchmark

We report the evaluation results of five policies on the RoboTwin 2.0 benchmark under the *Easy* and *Hard* settings, as shown in Tab. 15. Note that these two settings differ only in evaluation conditions, while the training setup remains identical.

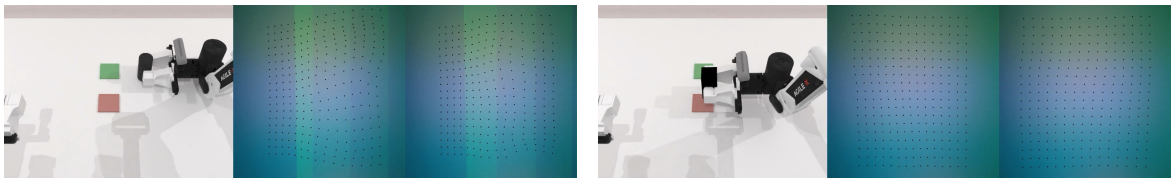

Figure 11. **Visualization of Tactile Data Acquisition.**

Table 14. **CLIP-based global distribution matching and coverage between our 11k background texture library and real-world textures from DTD.** We extract ViT-B/32 CLIP features for 11k textures from our library and 5,640 real textures from DTD. Coverage is measured as nearest-neighbor cosine similarity from each real texture to our texture pool.

Metric	Ours vs. DTD
# Ours	11,000
# Real	5,640
FID ↓	-0.36
Mean ↑	0.839
Median ↑	0.848
p10 ↑	0.749
p90 ↑	0.914
Min / Max	0.544 / 0.979

A.18. 50 RoboTwin 2.0 Tasks Descriptions

We list detailed 50 RoboTwin 2.0 tasks descriptions in Tab. 17.

A.19. Success Rates of Different Embodiments on RoboTwin 2.0 Tasks

Table 16 reports the success rates of five robot embodiments across the 50 RoboTwin 2.0 tasks, using the same set of expert programs for data generation.

A.20. Example: expert code generation pipeline for `handover_block`.

To make the pipeline in Fig. 3 more concrete, we use the `handover_block` task as a running example.

Inputs and assumptions (first round). In the first round, the inputs to the Code Agent include:

1. **Task description.** A natural language specification of the task, for example:

   > "Use the left arm to pick up the block, move it to the handover position between the two arms, then use the right arm to grasp the block and place it at the target location."

2. **API list.** A fixed set of commonly used, high-level, and strongly encapsulated APIs (listed in Appendix A.21), which hide low-level motion planning details from the agent.

3. **API usage examples.** A small collection of curated examples that demonstrate how to use these APIs for typical scenarios and tasks, serving as in-context guidance for the Code Agent.

4. **Object calibration and functional points.** Structured information about calibrated points and axes on the objects, including:

   - several grasp points on the block (for left arm and right arm grasps),
   - functional points on the block (for alignment and placement),
   - the functional point of the target placement location.

Given these inputs, the Code Agent produces executable expert control code that calls the high-level APIs to complete the task.

Outputs and iterative refinement. We then execute the generated code in simulation to test data generation, as illustrated in Fig. 2 (10 test rollouts in our default setting). For each rollout, we log:

- whether the rollout succeeds or fails,
- the failure type (for example, `Left grasp failure`, `Right grasp failure`, `Incorrect target position`),
- any runtime errors,
- snapshot images of the scene after key steps.

If the data generation success rate is above a predefined threshold (50% in Fig. 2), we accept the code as a valid expert policy for data collection. If the success rate is below the threshold, we feed all interaction logs (failures, error messages, and snapshot descriptions) back to the Code Agent, which then iteratively refines the code to fix the identified issues. This loop continues until the generated code reaches the required success rate.

Table 15. **RoboTwin 2.0 Simulation Benchmark (clean vs randomized, 50+ tasks).**

Simulation Task	RDT		Pi0		ACT		DP		DP3	
	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Adjust Bottle	81%	**75%**	90%	56%	97%	23%	97%	0%	**99%**	3%
Beat Block Hammer	**77%**	**37%**	43%	21%	56%	3%	42%	0%	72%	8%
Blocks Ranking RGB	3%	0%	**19%**	**5%**	1%	0%	0%	0%	3%	0%
Blocks Ranking Size	0%	0%	**7%**	**1%**	0%	0%	1%	0%	2%	0%
Click Alarmclock	61%	12%	63%	11%	32%	4%	61%	5%	**77%**	**14%**
Click Bell	80%	**9%**	44%	3%	58%	3%	54%	0%	**90%**	0%
Dump Bin Bigbin	64%	32%	83%	24%	68%	1%	49%	0%	**85%**	**53%**
Grab Roller	74%	43%	96%	**80%**	94%	25%	**98%**	0%	**98%**	2%
Handover Block	45%	**14%**	45%	8%	42%	0%	10%	0%	**70%**	0%
Handover Mic	90%	**31%**	98%	13%	85%	0%	53%	0%	**100%**	3%
Hanging Mug	**23%**	**16%**	11%	3%	7%	0%	8%	0%	17%	1%
Lift Pot	72%	9%	84%	**36%**	88%	0%	39%	0%	**97%**	0%
Move Can Pot	25%	12%	58%	**21%**	22%	4%	39%	0%	**70%**	6%
Move Pillbottle Pad	8%	0%	21%	**1%**	0%	0%	1%	0%	**41%**	0%
Move Playingcard Away	43%	11%	53%	**22%**	36%	0%	47%	0%	**68%**	3%
Move Stapler Pad	2%	0%	0%	**2%**	0%	0%	1%	0%	**12%**	0%
Open Laptop	59%	32%	**85%**	**46%**	56%	0%	49%	0%	82%	7%
Open Microwave	37%	20%	80%	**50%**	**86%**	0%	5%	0%	61%	22%
Pick Diverse Bottles	2%	0%	27%	**6%**	7%	0%	6%	0%	**52%**	1%
Pick Dual Bottles	42%	**13%**	57%	12%	31%	0%	24%	0%	**60%**	1%
Place A2B Left	3%	1%	31%	1%	1%	0%	2%	0%	**46%**	**2%**
Place A2B Right	1%	1%	27%	**6%**	0%	0%	13%	0%	**49%**	0%
Place Bread Basket	10%	2%	17%	**4%**	6%	0%	14%	0%	**26%**	1%
Place Bread Skillet	5%	**1%**	**23%**	**1%**	7%	0%	11%	0%	19%	0%
Place Burger Fries	50%	**27%**	**80%**	4%	49%	0%	72%	0%	72%	18%
Place Can Basket	19%	**6%**	41%	5%	1%	0%	18%	0%	**67%**	2%
Place Cans Plasticbox	6%	**5%**	34%	2%	16%	0%	40%	0%	**48%**	3%
Place Container Plate	78%	17%	**88%**	**45%**	72%	1%	41%	0%	86%	1%
Place Dual Shoes	4%	**4%**	**15%**	0%	9%	0%	8%	0%	13%	0%
Place Empty Cup	56%	7%	37%	**11%**	61%	0%	37%	0%	**65%**	1%
Place Fan	12%	2%	20%	**10%**	1%	0%	3%	0%	**36%**	1%
Place Mouse Pad	1%	0%	**7%**	**1%**	0%	0%	0%	0%	4%	**1%**
Place Object Basket	33%	**17%**	16%	2%	15%	0%	15%	0%	**65%**	0%
Place Object Scale	1%	**0%**	**10%**	**0%**	0%	**0%**	1%	**0%**	15%	**0%**
Place Object Stand	15%	5%	36%	**11%**	1%	0%	22%	0%	**60%**	0%
Place Phone Stand	15%	6%	35%	**7%**	2%	0%	13%	0%	**44%**	2%
Place Shoe	35%	**7%**	28%	6%	5%	0%	23%	0%	**58%**	2%
Press Stapler	41%	24%	62%	**29%**	31%	6%	6%	0%	**69%**	3%
Put Bottles Dustbin	21%	4%	54%	13%	27%	1%	22%	0%	**60%**	**21%**
Put Object Cabinet	33%	**18%**	68%	**18%**	15%	0%	42%	0%	**72%**	1%
Rotate QRcode	50%	5%	68%	**15%**	1%	0%	13%	0%	**74%**	1%
Scan Object	4%	**1%**	18%	**1%**	2%	0%	9%	0%	**31%**	**1%**
Shake Bottle Horizontally	84%	**51%**	99%	**51%**	63%	4%	59%	18%	**100%**	25%
Shake Bottle	74%	45%	97%	**60%**	74%	10%	65%	8%	**98%**	19%
Stack Blocks Three	2%	**0%**	**17%**	**0%**	0%	**0%**	0%	**0%**	1%	**0%**
Stack Blocks Two	21%	**2%**	**42%**	1%	25%	0%	7%	0%	24%	0%
Stack Bowls Three	51%	17%	**66%**	**24%**	48%	0%	63%	0%	57%	5%
Stack Bowls Two	76%	30%	**91%**	**41%**	82%	0%	61%	0%	83%	6%
Stamp Seal	1%	0%	3%	**4%**	2%	0%	2%	0%	**18%**	0%
Turn Switch	35%	15%	27%	**23%**	5%	2%	36%	1%	**46%**	8%
Average (%)	34.5	13.7	46.4	**16.3**	29.7	1.7	28.0	0.6	**55.2**	5.0

Table 16. **Success Rates of Different Embodiments on RoboTwin 2.0 Tasks.**

	RoboTwin1.0					RoboTwin2.0				
Task Name	**Aloha**	**ARX**	**Franka**	**Piper**	**UR5**	**Aloha**	**ARX**	**Franka**	**Piper**	**UR5**
Adjust Bottle	92%	88%	39%	0%	7%	93%	94%	34%	0%	12%
Beat Block Hammer	68%	86%	95%	0%	86%	64%	93%	98%	15%	90%
Blocks Ranking Rgb	92%	98%	96%	0%	82%	96%	97%	99%	13%	53%
Blocks Ranking Size	90%	95%	92%	0%	60%	96%	97%	89%	7%	38%
Click Alarmclock	89%	99%	100%	0%	95%	92%	99%	100%	0%	95%
Click Bell	100%	100%	100%	9%	100%	100%	100%	100%	91%	100%
Dump Bin Bigbin	85%	98%	90%	0%	82%	84%	100%	84%	9%	80%
Grab Roller	95%	69%	99%	0%	80%	95%	69%	99%	7%	81%
Handover Block	1%	3%	0%	0%	4%	83%	81%	0%	44%	0%
Handover Mic	62%	80%	92%	28%	0%	87%	98%	84%	65%	14%
Hanging Mug	68%	76%	5%	0%	12%	63%	73%	11%	0%	11%
Lift Pot	27%	50%	24%	5%	40%	27%	50%	36%	31%	40%
Move Can Pot	18%	0%	37%	2%	4%	93%	65%	92%	96%	99%
Move Pillbottle Pad	30%	52%	15%	0%	35%	67%	90%	69%	47%	86%
Move Playingcard Away	93%	100%	100%	0%	87%	99%	100%	100%	63%	66%
Move Stapler Pad	94%	92%	88%	0%	95%	92%	96%	89%	13%	75%
Open Laptop	76%	91%	78%	14%	55%	82%	92%	77%	23%	51%
Open Microwave	65%	85%	75%	5%	33%	96%	80%	59%	2%	23%
Pick Diverse Bottles	11%	1%	0%	0%	0%	51%	2%	0%	27%	4%
Pick Dual Bottles	8%	3%	0%	0%	0%	92%	6%	0%	81%	7%
Place A2B Left	65%	75%	70%	0%	72%	80%	88%	64%	29%	76%
Place A2B Right	70%	68%	68%	0%	69%	81%	82%	64%	31%	66%
Place Bread Basket	91%	91%	69%	0%	78%	89%	88%	62%	1%	67%
Place Bread Skillet	31%	28%	42%	0%	42%	34%	26%	42%	0%	37%
Place Can Basket	47%	1%	38%	0%	11%	70%	28%	61%	0%	3%
Place Cans Plasticbox	96%	93%	98%	0%	11%	100%	96%	85%	0%	82%
Place Container Plate	86%	85%	83%	0%	82%	89%	86%	86%	37%	81%
Place Dual Shoes	73%	28%	36%	0%	40%	77%	31%	41%	1%	32%
Place Empty Cup	92%	100%	100%	0%	100%	92%	100%	100%	4%	100%
Place Fan	93%	96%	75%	0%	85%	95%	93%	83%	0%	65%
Place Burger Fries	96%	95%	85%	0%	78%	97%	98%	80%	36%	74%
Place Mouse Pad	100%	80%	99%	2%	96%	99%	89%	100%	23%	73%
Place Object Basket	68%	13%	68%	0%	30%	74%	14%	61%	0%	7%
Place Object Scale	77%	93%	94%	0%	87%	78%	92%	82%	2%	76%
Place Object Stand	90%	92%	81%	0%	90%	97%	99%	81%	9%	92%
Place Phone Stand	66%	78%	52%	22%	44%	66%	78%	45%	53%	49%
Place Shoe	87%	85%	70%	0%	97%	84%	85%	74%	7%	91%
Press Stapler	87%	96%	99%	0%	77%	98%	96%	100%	59%	72%
Put Bottles Dustbin	0%	0%	0%	0%	0%	71%	1%	0%	56%	0%
Put Object Cabinet	13%	56%	43%	0%	0%	14%	24%	55%	0%	0%
Rotate Qrcode	78%	83%	98%	0%	81%	75%	74%	94%	0%	67%
Scan Object	8%	13%	21%	0%	8%	4%	45%	26%	0%	19%
Shake Bottle	62%	95%	82%	1%	98%	89%	94%	85%	74%	97%
Shake Bottle Horizontally	64%	93%	81%	1%	97%	90%	94%	85%	74%	98%
Stack Blocks Three	98%	97%	95%	0%	83%	94%	96%	80%	0%	51%
Stack Blocks Two	99%	99%	100%	0%	94%	98%	99%	96%	2%	68%
Stack Bowls Three	27%	64%	76%	0%	76%	43%	58%	82%	0%	81%
Stack Bowls Two	63%	84%	88%	0%	94%	78%	82%	88%	4%	94%
Stamp Seal	46%	91%	95%	0%	100%	56%	91%	4%	37%	100%
Turn Switch	27%	3%	51%	28%	10%	74%	3%	36%	81%	10%
Average	**65.3%**	**68.8%**	**67.6%**	**2.3%**	**57.7%**	**78.8%**	**74.2%**	**67.2%**	**25.1%**	**57.1%**
Difference	/	/	/	/	/	**+13.5%**	**+5.4%**	**-0.4%**	**+22.8%**	**-0.6%**

Table 17. Task descriptions.

Task Name	Description
Adjust Bottle	Pick up the bottle on the table upright with the correct arm.
Beat Block Hammer	Grab the hammer and hit the block.
Blocks Ranking RGB	Order red, green, blue blocks from left to right.
Blocks Ranking Size	Order three blocks by size, largest to smallest, left to right.
Click Alarmclock	Click the center of the top button of the alarm clock.
Click Bell	Click the bell's top center.
Dump Bin Bigbin	Pour balls from the small bin into the big bin.
Grab Roller	Use both arms to grab the roller on the table.
Handover Block	Take the red block, hand it to the other arm, and place it on the pad.
Handover Mic	One arm passes the microphone to the other arm.
Hanging Mug	Left arm places mug; right arm picks it up and hangs it on the rack.
Lift Pot	Use the arms to lift the pot.
Move Can Pot	Pick up the can and move it next to the pot.
Move Playingcard Away	Move the playing card farther away from the table.
Move Stapler Pad	Move the stapler to a colored mat with the correct arm.
Open Laptop	Open the laptop with one arm.
Open Microwave	Open the microwave with one arm.
Pick Diverse Bottles	Pick up one bottle with each arm.
Pick Dual Bottles	Pick up one bottle with each arm.
Place A2B Left	Place object A to the left of object B.
Place A2B Right	Place object A to the right of object B.
Place Bread Basket	One bread: one arm to basket; two: both arms to basket.
Place Bread Skillet	Put the bread into the skillet with one arm.
Place Burger Fries	Use both arms to place burger and fries on the tray.
Place Can Basket	One arm puts can in basket; other arm lifts basket.
Place Cans Plasticbox	Use both arms to place cans into the plastic box.
Place Container Plate	Place the container on the plate.
Place Dual Shoes	Both arms place two shoes in box with tips facing left.
Place Empty Cup	Place the empty cup on the coaster with one arm.
Place Fan	Place the fan on the colored mat facing the robot.
Place Mouse Pad	Place the mouse on a colored mat.
Place Object Basket	One arm puts object in basket; other arm moves basket slightly away.
Place Object Scale	Place the object on the scale with one arm.
Place Object Stand	Place the object on the stand with the correct arm.
Place Phone Stand	Place the phone on the phone stand.
Place Shoe	Place the shoe from table onto the mat.
Press Stapler	Press the stapler with one arm.
Put Bottles Dustbin	Put bottles into the dustbin left of the table.
Put Object Cabinet	Open drawer with one arm; use other arm to put object inside.
Rotate QRcode	Rotate the QR board so the QR code faces the robot.
Scan Object	One arm holds scanner; other holds object; scan the object.
Shake Bottle Horizontally	Shake the bottle horizontally with the correct arm.
Shake Bottle	Shake the bottle with the correct arm.
Stack Blocks Three	Stack blue on green on red at the center.
Stack Blocks Two	Stack green on red at the center.
Stack Bowls Three	Stack three bowls.
Stack Bowls Two	Stack two bowls.
Stamp Seal	Stamp the specific color mat with the stamp.
Turn Switch	Click the switch with the arm.
Move Pillbottle Pad	Place the pill bottle onto the pad.

A.21. RoboTwin 2.0 Prompt

```
# You can directly use the actors provided in the actor_list.
# For example, if actor_list contains ["self.hammer", "self.block"],
# you can directly write:
object1 = self.hammer
object2 = self.block

# Using ArmTag class to represent arms
arm_tag = ArmTag("left")   # Left arm
arm_tag = ArmTag("right")  # Right arm

# Example of selecting an arm based on conditions:
arm_tag = ArmTag("left" if actor_position[0] < 0 else "right")

# Functional points on actors
# Each actor in the environment may have multiple functional points
# that are useful for different interactions.
# Functional points provide precise locations for interactions like
# grasping, placing, or aligning objects.

# To get a functional point from an actor:
functional_point_pose = actor.get_functional_point(point_id, "pose")
position = functional_point_pose.p   # [x, y, z]
orientation = functional_point_pose.q  # [qw, qx, qy, qz]

# Stacking one object on top of another
# Example: placing current_actor on top of last_actor using a
# functional point as target_pose.
target_pose = self.last_actor.get_functional_point(point_id, "pose")

self.move(
    self.place_actor(
        actor=self.current_actor,   # The object to be placed
        target_pose=target_pose,    # Pose acquired from last_actor
        arm_tag=arm_tag,
        functional_point_id=0,      # Align functional point 0 (or as needed)
        pre_dis=0.1,
        dis=0.02,
        pre_dis_axis="fp",          # Use functional point direction
    )
)

# Actors of type "pose" in actor_list
# For all actors in actor_list that are of type "pose", such as
# middle_pose or actor_target_pose, these are already Pose objects
# (or lists of Pose). You do NOT need to call .get_pose() again.
# You can pass them directly as target_pose.

# Example: place self.box at self.actor_pose (already a Pose)
self.move(
    self.place_actor(
        actor=self.box,
        target_pose=self.actor_pose,      # already a Pose
        arm_tag=grasp_arm_tag,
        functional_point_id=0,           # if the actor has functional points
        pre_dis=0,
        dis=0,                           # dis = 0 if is_open is False
        is_open=False,                   # gripper will not open after placing
        constrain="free",                # "align" only if pose is constrained
        pre_dis_axis='fp',               # use functional point direction
    )
)

# Note:
# For the target_actor, it is an actor, not a Pose, so you need to call
# get_pose() to get its pose, or call get_functional_point() to get
# a functional point.

# Selecting an arm for grasping based on actor position
# Get the actor's pose
actor_pose = self.actor.get_pose()
actor_position = actor_pose.p  # [x, y, z]

# Select arm based on x-position
arm_tag = ArmTag("left" if actor_position[0] < 0 else "right")
```

```
# Grasp actor with selected arm
self.move(
    self.grasp_actor(
        actor=self.actor,
        arm_tag=arm_tag
    )
)

# Basic grasping API examples
# Grasp an actor with specified pre-grasp distance and grasp distance
self.move(
    self.grasp_actor(
        actor=self.actor,
        arm_tag=arm_tag,      # ArmTag("left") or ArmTag("right")
        pre_grasp_dis=0.1,
        grasp_dis=0
    )
)

# Grasp-and-lift example
# Grasp the object
self.move(
    self.grasp_actor(
        actor=self.actor,
        arm_tag=arm_tag,      # ArmTag("left") or ArmTag("right")
        pre_grasp_dis=0.1,
        grasp_dis=0
    )
)

# Lift the object up (always lift after grasping to avoid collision)
self.move(
    self.move_by_displacement(
        arm_tag=arm_tag,
        z=0.07,               # Move 7cm upward
        move_axis='world'     # Move in world coordinates
    )
)

# Gripper control examples
# Open gripper fully
self.move(
    self.open_gripper(
        arm_tag=arm_tag,
        pos=1.0               # fully open
    )
)

# Open gripper halfway
self.move(
    self.open_gripper(
        arm_tag=arm_tag,
        pos=0.5
    )
)

# Close gripper fully
self.move(
    self.close_gripper(
        arm_tag=arm_tag,
        pos=0.0               # fully close
    )
)

# Close gripper halfway
self.move(
    self.close_gripper(
        arm_tag=arm_tag,
        pos=0.5
    )
)

# Placing objects at a target location
# Place an object at a specific target pose
self.move(
    self.place_actor(
        actor=self.actor,
        arm_tag=arm_tag,
```

```
        target_pose=self.target_pose,  # retrieved from the actor list
        functional_point_id=0,         # if the actor has functional points
        pre_dis=0.1,
        dis=0.02,                      # dis = 0 if is_open is False
        is_open=True,                  # True to release object after placing
        pre_dis_axis='fp',             # use functional point direction
    )
)

# Lift the gripper up after placing (only needed if is_open is True)
self.move(
    self.move_by_displacement(
        arm_tag=arm_tag,
        z=0.07,               # Move 7cm upward
        move_axis='world'
    )
)

# Placing with functional point alignment
# Place the object by aligning functional point 0 with the target pose
self.move(
    self.place_actor(
        actor=self.actor,
        arm_tag=arm_tag,
        target_pose=target_pose,
        functional_point_id=0,    # align this functional point
        pre_dis=0.1,
        dis=0.02,
        pre_dis_axis='fp'         # use functional point direction
    )
)

# Dual-arm coordination examples
# Move both arms simultaneously to grasp objects
left_arm_tag = ArmTag("left")
right_arm_tag = ArmTag("right")

self.move(
    self.grasp_actor(actor=self.left_actor, arm_tag=left_arm_tag),
    self.grasp_actor(actor=self.right_actor, arm_tag=right_arm_tag)
)

# Lift both actors up after grasping
self.move(
    self.move_by_displacement(arm_tag=left_arm_tag, z=0.07),
    self.move_by_displacement(arm_tag=right_arm_tag, z=0.07)
)

# Place left object while moving right arm back to origin
move_arm_tag = ArmTag("left")   # arm placing the object
back_arm_tag = ArmTag("right")  # arm returning to origin

self.move(
    self.place_actor(
        actor=self.left_actor,
        arm_tag=move_arm_tag,
        target_pose=target_pose,
        pre_dis_axis="fp",
    ),
    self.back_to_origin(arm_tag=back_arm_tag)
)

# Returning arms to their initial positions
# Return a single arm to origin
self.move(self.back_to_origin(arm_tag=arm_tag))

# Return both arms to origin simultaneously
left_arm_tag = ArmTag("left")
right_arm_tag = ArmTag("right")

self.move(
    self.back_to_origin(arm_tag=left_arm_tag),
    self.back_to_origin(arm_tag=right_arm_tag)
)
```

A.21.1. LLM-Generated Code and Human-Written Code Case Study

LLM-Generated Code (`gpt_place_shoe`)

```
class gpt_place_shoe(place_shoe):
    def play_once(self):
        # Initial observation
        self.save_camera_images(task_name="place_shoe", step_name="step1_initial_scene_state",
    generate_num_id="generate_num_0")

        # Get the shoe's position to determine which arm to use
        shoe_pose = self.shoe.get_pose()
        shoe_position = shoe_pose.p
        arm_tag = ArmTag("left" if shoe_position[0] < 0 else "right")

        # Grasp the shoe
        self.move(self.grasp_actor(actor=self.shoe, arm_tag=arm_tag, pre_grasp_dis=0.1, grasp_dis=0))
        self.save_camera_images(task_name="place_shoe", step_name="step2_shoe_grasped",
    generate_num_id="generate_num_0")

        # Lift the shoe up to avoid collision
        self.move(self.move_by_displacement(arm_tag=arm_tag, z=0.07, move_axis='world'))
        self.save_camera_images(task_name="place_shoe", step_name="step3_shoe_lifted",
    generate_num_id="generate_num_0")

        # Get target pose from the top functional point of the target block
        target_pose = self.target_block.get_functional_point(1, "pose")

        # Place the shoe on the target block with head towards left side
        self.move(self.place_actor(actor=self.shoe, arm_tag=arm_tag, target_pose=target_pose,
    functional_point_id=0, pre_dis=0.1, dis=0.02, is_open=True, constrain="align", pre_dis_axis='fp'))
        self.save_camera_images(task_name="place_shoe", step_name="step4_shoe_placed",
    generate_num_id="generate_num_0")

        # Lift the gripper up after placing
        self.move(self.move_by_displacement(arm_tag=arm_tag, z=0.07, move_axis='world'))

        # Return arm to origin
        self.move(self.back_to_origin(arm_tag=arm_tag))

        # Final observation
        self.move(self.save_camera_images(task_name="place_shoe", step_name="step5_final_scene_state",
    generate_num_id="generate_num_0"))
```

Human-Written Code (`place_shoe`)

```
class place_shoe(base_task):
    def play_once(self):
        # Get the shoe's position to determine which arm to use
        shoe_pose = self.shoe.get_pose().p
        arm_tag = ArmTag("left" if shoe_pose[0] < 0 else "right")

        # Grasp the shoe with specified pre-grasp distance and gripper position
        self.move(self.grasp_actor(self.shoe, arm_tag=arm_tag, pre_grasp_dis=0.1, gripper_pos=0))

        # Lift the shoe up by 0.07 meters in z-direction
        self.move(self.move_by_displacement(arm_tag=arm_tag, z=0.07))

        # Get target_block's functional point as target pose
        target_pose = self.target_block.get_functional_point(0)

        # Place the shoe on the target_block with alignment constraint and specified pre-placement distance
        self.move(self.place_actor(self.shoe, arm_tag=arm_tag, target_pose=target_pose,
    functional_point_id=0, pre_dis=0.12, constrain="align"))

        # Open the gripper to release the shoe
        self.move(self.open_gripper(arm_tag=arm_tag))
```

The LLM-generated code tends to be more verbose, explicitly logging intermediate visual states and detailing parameters (e.g., `pre_dis_axis='fp'`, `is_open=True`), while human-written scripts are more minimal, omitting intermediate steps and favoring compact execution. Despite functional similarity, the structural differences illustrate that **MLLM-**

generated programs are not only executable but emphasize step-by-step clarity, contributing to more robust feedback and repair.

A.22. Task Instruction and Object Description Example

Instruction Templates (task: 'Pick Dual Bottles')

"Use {a} to place {A} left of {B}.", "Set {A} to the left of {B}.", "Move {A} beside {B} using {a}.", "Place {A} on {B}'s left side.", "Using {a}, position {A} next to {B}.", "Stick {A} on the left of {B}.", "Use {a} and place {A} on {B}'s left.", etc

Object Description

object id - '001_bottle/0':
"red bottle", "red soda bottle", "plastic red bottle", "red bottle with yellow label", "red plastic bottle with smooth surface", "yellow text printed on red bottle surface", "red bottle with white label design and markings", "red bottle with white sealing and brown top screw cap", etc
object id - '039_mug/0':
"black mug", "dark coffee mug", "sleek black mug", "black ceramic mug", "single-handle mug", "smooth black surface mug", "medium-sized drinking mug", "round mug with curved side", "dark mug with sturdy handle", "solid black mug with smooth finish", etc

A.23. Prompts for Generating Task Instructions and Object Descriptions

```
# Task Instruction Template
- Goal: Generate task instruction template
- Requirements:
  - Generate 60 items. Vary in sentence length and structure
  - Use natural action verbs (grab, slide, place)
- split
  - 50 items for training
  - 10 items for evaluation

## Schema Requirements
- Goal: Use placeholders for objects in instructions
- Requirements:
  - Format: {X} for objects defined in schema
  - Include all object placeholders ({A-Z}) in every instruction
  - Omit arm references and placeholders ({a-z}) in 50% of instructions
  - Ensure natural flow when placeholders are replaced with actual values

# Object Description
- Goal: Generate natural object descriptions for robotic manipulation
- Requirements:
  - Generate 15 items. Vary in sentence length and structure
  - Use natural oral language
  - Include essential physical properties (color, shape, size, texture)
  - Use noun-focused phrases
  - For multi-part objects, use structures like 'X with Y'
- split
  - 12 items for training
  - 3 items for evaluation

# Episode
An episode is a specified task, in which each task may have different objects to be manipulated,
resulting in the same task template being reused by replacing the placeholders with specific objects.
For example:
  {A} -> 'medium-sized yellow bottle'
  {A} -> 'green drink bottle with bold labels'

General Task -> Specific Episode:
  {A} -> bottle/0.glb
  {A} -> bottle/1.glb

The number of task instructions for an episode can be calculated by:
  Episode_num = TaskInstruction_num * ObjectDescription_num
```