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

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

[1] MoE key Lab of Artificial Intelligence, AI Institute, SJTU‡, [2] HKU MMLab‡,
[3] Shanghai AI Lab, [4]D-Robotics, [5]SZU, [6]THU, [7]TeleAI, [8]FDU, [9]USTC, [10]SUSTech,
[11]SYSU, [12]CSU, [13]NEU,[14]HKU-SH ICRC, [15]NJU, [16]Lumina EAI
[*] Equal contribution [✉] Corresponding authors [†] Co-project leads
[‡] Equally leading organizations

Webpage:https://robotwin-platform.github.io Doc: https://robotwin-platform.github.io/doc/

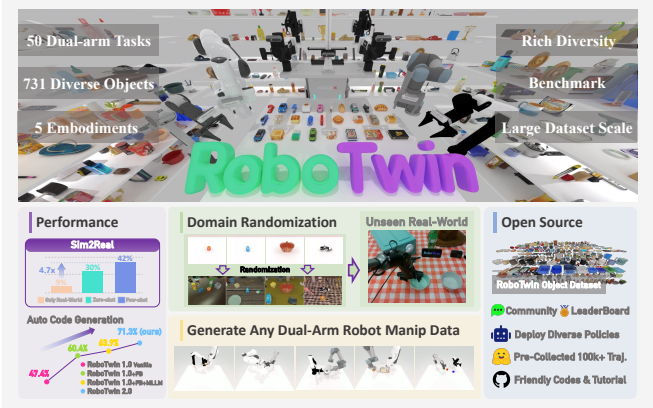

Fig. 1: **Overview of RoboTwin 2.0.** RoboTwin 2.0 is a scalable framework for bimanual manipulation, integrating an expert data generation pipeline with a 50-task benchmark built on the RoboTwin Object Dataset (731 objects, 147 categories). A multimodal language agent automates task program synthesis, while flexible dual-arm configurations enable large-scale, diverse data collection. Policies trained on RoboTwin 2.0 exhibit improved robustness and generalization to unseen environments.

***Abstract*— Synthetic data generation via simulation represents a promising approach for enhancing robotic manipulation. However, current synthetic datasets remain insufficient for robust bimanual control due to limited scalability in novel task generation and oversimplified simulations that inadequately capture real-world complexity. We present RoboTwin 2.0, a scalable framework for automated diverse synthetic data generation and unified evaluation for bimanual manipulation. We construct RoboTwin-OD, an object library of 731 instances across 147 categories with semantic and manipulation labels. Building on this, we design a expert data generation pipeline by utilizing multimodal large language models to systhesize task-execution code with simulation-in-the-loop refinement. To improve sim-to-real transfer, RoboTwin 2.0 applies structured domain randomization over five factors (clutter, lighting, background, tabletop height, language instructions). Using this approach, we instantiate 50 bimanual tasks across five robot embodiments. Experimental results demonstrate a 10.9% improvement in code-generation success rates. For downstream learning, vision-language-action models trained with our synthetic data achieve 367% performance improvements in the few-shot setting and 228% improvements in the zero-shot setting, relative to a 10-demo real-only baseline. We further evaluate multiple policies across 50 tasks with two difficulty settings, establishing a comprehensive benchmark to study policy performance. We release the generator, datasets, and code to support scalable research in robust bimanual manipulation.**

I. Introduction

Bimanual robotic manipulation is essential for complex tasks such as collaborative assembly, tool use, and handovers. Training generalizable bimanual policies, particularly vision-language-action (VLA) foundation models [**?**], requires datasets that are high quality, diverse, and large scale. Without sufficient variation in object geometry, scene clutter, lighting, instruction language, and robot embodiments, learned policies overfit and generalization degrades across environments and hardware. However, collecting real-world demonstrations at scale remains costly, time-intensive, and logistically difficult, especially when targeting broad task, object, and embodiment coverage.

Simulation has become an effective way to scale multimodal data collection and enable sim-to-real transfer [28], [9]. However, prevailing pipelines exhibit three persistent limitations: (i) the absence of automated quality control, which admits execution failures and weak grasps that degrade learning; (ii) shallow domain randomization, producing overly clean, homogeneous scenes that neglect clutter, illumination changes, and instruction ambiguityfactors critical for robust transfer; and (iii) limited cross-embodiment coverage, despite substantial differences in kinematics and grasp strategies

across bimanual platforms. For example, low-DoF systems such as Piper tend to favor lateral grasps, whereas high-DoF arms like Franka support top-down precision grasps. Current synthetic datasets rarely encode these embodiment-specific affordances and task constraints, limiting generality.

To address these challenges, we introduce **RoboTwin 2.0**, a scalable simulation-based framework for generating high-quality, diverse, and realistic datasets for bimanual manipulation. The framework comprises: (1) an automated expert pipeline that uses multimodal large language models (MLLMs) with simulation-in-the-loop feedback to validate and refine task execution code; (2) comprehensive domain randomization over language, clutter, background textures, lighting, and tabletop layouts to improve sim-to-real transfer and policy generalization; and (3) embodiment-aware adaptation that annotates object affordances and generates robot-specific action candidates for heterogeneous dual-arm kinematics. Building on these components, we introduce three new resources to support scalable research in bimanual manipulation: (1) the RoboTwin-OD asset library, comprising 731 annotated object instances across 147 categories; (2) an automated data generation pipeline with comprehensive domain randomization and a pre-collected, open-source dataset of expert trajectories spanning 50 tasks across five dual-arm robot platforms; and (3) a benchmark for evaluating policy generalization to cluttered environments and open-ended language goals. Together, these resources enable the community to train and evaluate robust bimanual manipulation policies under conditions that closely reflect real-world complexity and diversity.

In summary, our main contributions are as follows: (1) We develop an automated expert data generation framework that integrates MLLMs with simulation-in-the-loop feedback to ensure high-quality, expert-level trajectories; (2) We propose a systematic domain randomization strategy that enhances policy robustness by increasing data diversity and sim-to-real generalization; (3) We introduce an embodiment-aware adaptation mechanism that generates robot-specific manipulation candidates based on object affordances; (4) We release the RoboTwin-OD, a large-scale pre-collected multi-embodiment domain-randomized trajectory dataset, a scalable bimanual data generator, and a standardized evaluation benchmark to support scalable training and evaluation of generalizable policies across different robot embodiments, scene configurations, and language instructions.

II. Method

Figure 2 overviews the RoboTwin 2.0 pipeline. A taskcode generation module employs MLLMs with simulation-in-the-loop feedback to synthesize executable plans from natural-language instructions. The module is grounded in a large object asset library (RoboTwin-OD) and a predefined skill library, enabling scalable instantiation across diverse objects and manipulation scenarios. A comprehensive domain-randomization scheme along language, visual, and spatial dimensions further expands coverage, producing diverse, realistic demonstrations and policies robust to real-world variability.

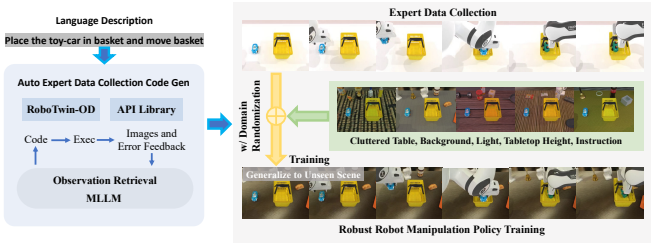

Fig. 2: **Our Pipeline.** Built on RoboTwin-OD and skill APIs, an MLLM guides code generation with simulation feedback to produce expert programs and domain-randomized trajectories.

A. 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. 3), in contrast to pipelines that depend on manual priors or omit feedback [16], [34]. The system comprises two agents: a code-generation agent that translates natural language instructions into executable programs, and a visionlanguage 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.

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 in simulation 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 visionlanguage model (VLM) monitors all ten trials and performs per-frame analysis to assess stepwise success and localize failures. Beyond temporal localization,

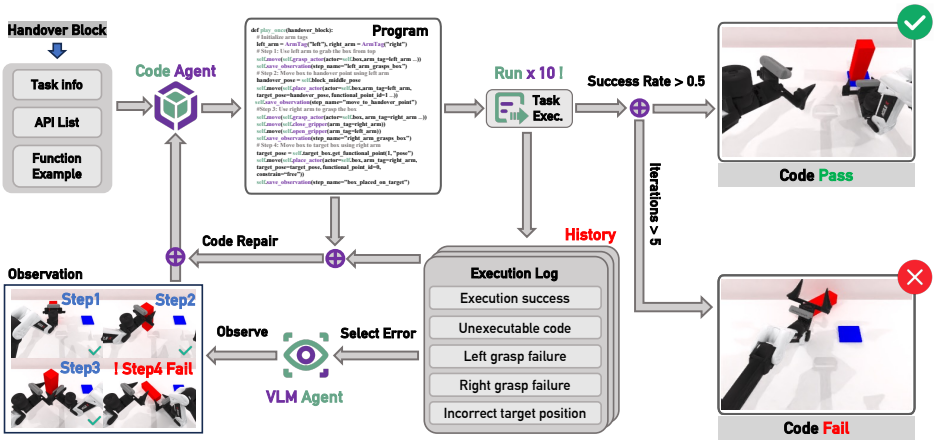

Fig. 3: **Expert Code Generation Pipeline.**

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 H.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.

B. 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. 4a).

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

Fig. 4: **Visualization of domain randomization and our texture library.**

Scene Clutter. To improve robustness to environmental variation, we augment tabletop scenes with task-irrelevant distractors sampled from RoboTwin-OD (Section III-A). 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 collected 1,000 surface descriptions via LLM prompting and web search, then generated 20 images per description with Stable Diffusion v2 [**?**] (20,000 total). Human-in-the-loop filtering yielded 11,000 high-quality textures. The library is used in simulation to increase visual diversity and mitigate overfitting to clean synthetic scenes (Fig. 4b).

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. 4a (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 within a plausible range in simulation, strengthening the policys robustness to variations in table height.

Trajectory-Level Diverse Language Instructions. We employ a 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 combinatorial augmentation produces a large, linguistically varied instruction set and improves generalization to unseen language and scene configurations (Appendix I, J).

C. 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. 5). To model this variation, we annotate each ob-

Fig. 5: **Diverse Grasping Behaviors**.

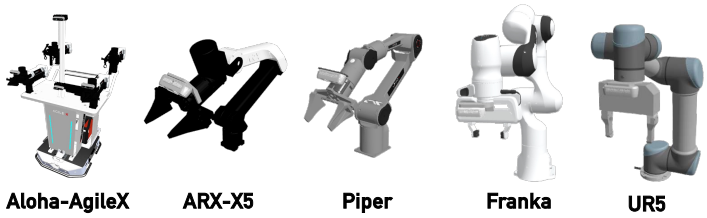

Fig. 6: **Five RoboTwin 2.0 Embodiments**.

ject 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 B, 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.

III. RoboTwin 2.0 Data Generator, Benchmark and rdDataset

A. RoboTwin-OD: RoboTwin Object Dataset

Fig. 7: **RoboTwin-OD.** A large-scale object dataset with rich annotations.

We build RoboTwin-OD, an object dataset with rich semantics covering 147 categories and 731 objects: 534 in-house instances across 111 categories reconstructed from RGB-to-3D via the Rodin platform [**?**], followed by convex decomposition and mesh merging for physically accurate collisions; 153 objects from 27 categories in Objaverse [8]; and 44 articulated instances from 9 categories in SAPIEN PartNet-Mobility [39]. All sources support cluttered scenes, with Objaverse enhancing the visual and semantic diversity of distractors. We also curate a texture library for surfaces and backgrounds using generative models with human-in-the-loop filtering. To support language grounding and robustness across diverse objects, we deploy an automated description generator with human verification, producing 15 annotations per object that vary in shape, texture, function, part structure, and granularity. For object-centric interaction, we annotate key pointaxis information, including placement points, functional points, grasp points, and grasp axes, to encode affordances. Combined with our manipulation API library, these annotations enable generalizable grasp execution in simulation.

B. 50 Tasks for Data Generation and Benchmarking

Building on automated task generation, embodiment-adaptive synthesis, and the RoboTwin-OD asset library, we define 50 dual-arm collaborative manipulation tasks. Data collection and evaluation are supported on five robot embodiments, enabling comprehensive cross-embodiment benchmarking; representative keyframes are shown in Fig. 8. We also release a pre-collected corpus of 100,000+ dual-arm trajectories across these tasks in RoboTwin 2.0.

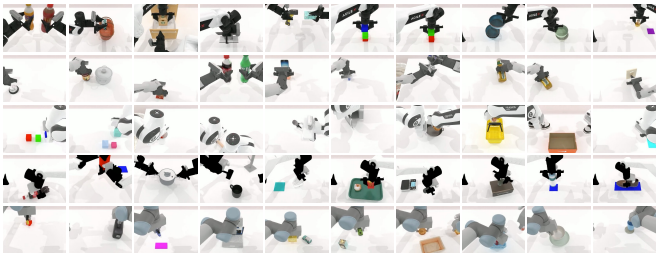

Fig. 8: **50 Bimanual Manipulation Tasks with Multi-Embodiment Support per Task**.

IV. 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.

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

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 9 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) visionlanguage 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 H.

B. RoboTwin 2.0 Benchmark

We present the RoboTwin 2.0 Benchmark for evaluating policy performance. Results on 50 RoboTwin tasks are reported in Appendix L, and Tab. II summarizes the average performance of RGB-based policies across evaluation settings. To assess generalization, we evaluate all 50 tasks on the AlohaAgileX 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 success rate as the metric of few-shot adaptability and robustness. Appendix K visualizes the benchmark setup, and Appendix E details all training protocols.

As shown in Tab. II, 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 visionlanguageaction 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 visionlanguageaction pretraining provides useful priors for scene generalization and improves robustness to environmental variation, yet robustness under domain shift remains a central challenge. In conjunction with Secs. IV-C and IV-D, these findings underscore the value of RoboTwin 2.0 as both a complementary dataset and a benchmark for systematic evaluation.

C. 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 configurations are provided in Appendix D and E.

As shown in Table III, 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.

D. Evaluation on Sim-to-Real Performance

To assess RoboTwin 2.0s 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 settings: (1) 10 real-world demonstrations in clean tabletop environments; (2) the same demonstrations augmented with 1,000 domain-randomized synthetic trajectories generated under clutter, varied lighting, and diverse backgrounds; (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 settings: clean and cluttered tabletops crossed with seen and unseen backgrounds (Fig. 10). The synthetic-only model excludes seen backgrounds during training, so the corresponding cells in

TABLE I: **Overall performance on tasks shared by RoboTwin 1.0 and 2.0.** Per-task success rates are in Appendix VIII.

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

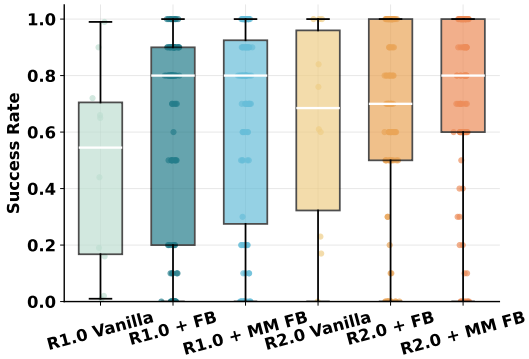

Fig. 9: **Success Rate Distribution.**

TABLE II: **Average Result of RoboTwin 2.0 Benchmark**. Full results are in Appendix L.

Simulation Tasks	RDT		Pi0		ACT		DP		DP3	
	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Average (in %)	34.5	13.7	46.4	**16.3**	29.7	1.7	28.0	0.6	**55.2**	5.0

TABLE III: **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%	0.0%	18.8%	14.6%	24.8%	22.5%	24.9%	29.1%

TABLE IV: **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%

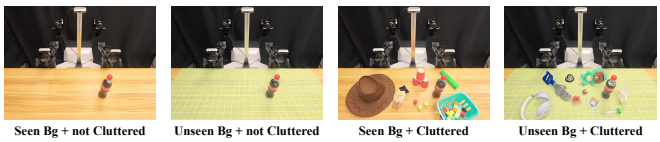

Fig. 10: **Four Real-World Evaluation Configurations.**

Table IV 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 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. Taken together, the few-shot results indicate that only limited real-world data are required to bridge the sim-to-real gap.

V. Related Works

A. Datasets and Benchmarks for Robotic Manipulation

Physics-based simulators underpin manipulation research. SAPIEN [39] supports dynamic interaction with 2,300+ articulated objects; ManiSkill2 [14] provides millions of demonstrations; Meta-World [41], CALVIN [27], LIBERO [25], and RoboVerse [13] target multi-task, language-conditioned, lifelong, and domain-randomized settings; RoboCasa [29] offers large-scale human demonstrations but lacks automation and a dual-arm focus. Large real-world datasetsAgiBot World [4], RoboMIND [38], Open X-Embodiment [30], Bridge [10]bridge sim-to-real with millions of trajectories. Building on RoboTwin-1.0 [28], RoboTwin 2.0 integrates LLM-driven feedback and systematic domain randomization over visual, physical, and task factors, yielding richer corpora and stronger generalization (Appendix C).

B. Robot Learning in Manipulation

Task-specific policies [33], [17], [42], [7], [12], [5], [24], [22], [23], [36], [35], [6] excel on individual tasks yet transfer poorly across embodiments. Foundation models trained on million-scale, multi-robot data generalize better: RT-1 [3] unifies vision, language, and action; RT-2 [2] co-fine-tunes visionlanguage models on web and robot data for semantic planning; RDT-1B [26] and π_0 [1] use $>$ 1M episodes to capture diverse bimanual dynamics. OpenVLA [19] and CogACT [21], with Octo [32], LAPA [40], and OpenVLA-OFT [18], demonstrate efficient adaptation to new robots and sensors. We contribute digital-twin data collection and broad domain randomization to produce realistic datasets that support robust, generalizable bimanual policies.

VI. Conclusion

This paper introduced RoboTwin 2.0, a scalable simulation framework for generating diverse, high-fidelity expert data for robust bimanual manipulation. The system integrates MLLM-based expert code generation, embodiment-adaptive behavior synthesis, and comprehensive domain randomization, addressing key limitations of prior synthetic data generators. Leveraging an annotated object library and automated trajectory synthesis, RoboTwin 2.0 produces visually, linguistically, and physically rich datasets while reducing manual effort. Experiments demonstrate consistent improvements in cluttered scenes, enhanced generalization to unseen tasks, and reliable cross-embodiment transfer; notably, few-shot and zero-shot evaluations indicate measurable sim-to-real improvements, showing that domain-randomized, semantically grounded synthetic data can substantially reduce real-world data requirements. To support the community, we release as open source RoboTwin-OD, a pre-collected trajectory dataset, a standardized benchmark, and a scalable data-collection toolchain. RoboTwin 2.0 provides a principled basis for unified benchmarking and scalable sim-to-real pipelines.

References

[1] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. *pi_0*: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[2] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[3] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[4] Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xu Huang, Shu Jiang, et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025.

[5] Tianxing Chen, Yao Mu, Zhixuan Liang, Zanxin Chen, Shijia Peng, Qiangyu Chen, Mingkun Xu, Ruizhen Hu, Hongyuan Zhang, Xuelong Li, et al. G3flow: Generative 3d semantic flow for pose-aware and generalizable object manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1735–1744, 2025.

[6] Tianxing Chen, Kaixuan Wang, Zhaohui Yang, Yuhao Zhang, Zanxin Chen, Baijun Chen, Wanxi Dong, Ziyuan Liu, Dong Chen, Tianshuo Yang, et al. Benchmarking generalizable bimanual manipulation: Robotwin dual-arm collaboration challenge at cvpr 2025 meis workshop. *arXiv preprint arXiv:2506.23351*, 2025.

[7] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[8] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023.

[9] Shengliang Deng, Mi Yan, Songlin Wei, Haixin Ma, Yuxin Yang, Jiayi Chen, Zhiqi Zhang, Taoyu Yang, Xuheng Zhang, Heming Cui, et al. Graspvla: a grasping foundation model pre-trained on billion-scale synthetic action data. *arXiv preprint arXiv:2505.03233*, 2025.

[10] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021.

[11] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[12] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[13] Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, et al. Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025.

[14] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. In *The Eleventh International Conference on Learning Representations*, 2023.

[15] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*, 2022.

[16] Pu Hua, Minghuan Liu, Annabella Macaluso, Yunfeng Lin, Weinan Zhang, Huazhe Xu, and Lirui Wang. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. In *8th Annual Conference on Robot Learning*.

[17] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

[18] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[19] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*.

[20] Zhiqian Lan, Yuxuan Jiang, Ruiqi Wang, Xuanbing Xie, Rongkui Zhang, Yicheng Zhu, Peihang Li, Tianshuo Yang, Tianxing Chen, Haoyu Gao, et al. Autobio: A simulation and benchmark for robotic automation in digital biology laboratory. *arXiv preprint arXiv:2505.14030*, 2025.

[21] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[22] Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. In *International Conference on Machine Learning*, pages 20725–20745. PMLR, 2023.

[23] Zhixuan Liang, Yao Mu, Hengbo Ma, Masayoshi Tomizuka, Mingyu Ding, and Ping Luo. 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*, pages 16467–16476, 2024.

[24] Zhixuan Liang, Yao Mu, Yixiao Wang, Tianxing Chen, Wenqi Shao, Wei Zhan, Masayoshi Tomizuka, Ping Luo, and Mingyu Ding. Dexhanddiff: Interaction-aware diffusion planning for adaptive dexterous manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1745–1755, 2025.

[25] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

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

[27] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

[28] Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, et al. Robotwin: Dual-arm robot benchmark with generative digital twins. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025.

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

[30] Abby ONeill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, 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)*, pages 6892–6903. IEEE, 2024.

[31] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

[32] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[33] Chenxi Wang, Hongjie Fang, Hao-Shu Fang, and Cewu Lu. Rise: 3d perception makes real-world robot imitation simple and effective. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2870–2877. IEEE, 2024.

[34] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation, 2023.

[35] Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*, 2025.

[36] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. Tinyvla: Toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025.

[37] Wu Wen, Xiaobo Xue, Ya Li, Peng Gu, and Jianfeng Xu. Code similarity detection using ast and textual information. *International Journal of Performability Engineering*, 15(10):2683, 2019.

[38] Kun Wu, Chengkai Hou, Jiaming Liu, Zhengping Che, Xiaozhu Ju, Zhuqin Yang, Meng Li, Yinuo Zhao, Zhiyuan Xu, Guang Yang, et al. Robomind: Benchmark on multi-embodiment intelligence normative data for robot manipulation. *arXiv preprint arXiv:2412.13877*, 2024.

[39] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020.

[40] Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Se June Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, et al. Latent action pretraining from videos. In *CoRL 2024 Workshop on Whole-body Control and Bimanual Manipulation: Applications in Humanoids and Beyond*.

[41] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[42] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy. *arXiv e-prints*, pages arXiv–2403, 2024.

[43] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

APPENDIX

A. 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.

B. 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 V, 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 M.

C. 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 VI.

D. 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.

E. Policies Training Details

We adopt joint angles as the models prediction target, formulating action prediction as joint-angle regression.

RDT in experiment IV-C 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 IV-C 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.

F. Support for Flexible Embodiment Combinations

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.

To execute high-success-rate manipulation trajectories across different embodiments (see Section II-C), 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-AgileXalong with multiple gripper types, including the Panda gripper and WSG gripper. As shown in Fig. 11, we demonstrate successful task executions across a variety of dual-arm pairings, highlighting RoboTwin 2.0s ability to scale to heterogeneous robot configurations and its readiness for future real-world deployment.

G. 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 VII 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.

TABLE V: **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 VI: **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 [41]	50	×	✓	×
Robosuite [43]	9	×	×	×
RoboCasa [41]	25	✓	×	×
Maniskill2 [14]	20	×	✓	×
AutoBio [20]	16	×	✓	✓
RoboTwin 1.0 [28]	14	×	✓	✓
RoboTwin 2.0 (ours)	50	✓	✓	✓

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 [37]	23.72%	**44.78%**
CodeBLEU Similarity [31]	17.18%	**18.53%**
CodeBERT Similarity [11]	97.72%	**98.80%**
Unixcoder Similarity [15]	76.24%	**82.21%**
Avg. VLM Token Cost (per observation)	–	**6894**

TABLE VII: **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.

Fig. 11: **Heterogeneous Dual-Arm Control via Object-Centric Manipulation.**

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

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.

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.

TABLE VIII: 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%**

3) Per-task Success Rates of Code Generation: We report the success rates of all tasks in Tab. IX.

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.

a) 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.

b) 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.

c) 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.

d) 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 factorssuch as incorrect grasp axis parametersremain challenging for purely vision-based observers to diagnose.

5) 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
```

TABLE IX: **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%*		

```
    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"))
```

Listing 1: LLM-Generated Code for `gpt_place_shoe` Task

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))
```

Listing 2: Human-Written Code for `place_shoe` Task

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.

I. 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
```

J. 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
```

Listing 3: Prompts for Generating Task Instructions and Object Descriptions

K. RoboTwin 2.0 Benchmark Setting Visualization

We visualize the simulation settings of the RoboTwin 2.0 benchmark in Fig. 12. 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).

Fig. 12: **Visualization of RoboTwin 2.0 Benchmark Settings.**

L. 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 in Tab. X. Note that these two settings differ only in evaluation conditions, while the training setup remains identical.

M. Success Rates of Different Embodiments on RoboTwin 2.0 Tasks

Table XI 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.

TABLE X: **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 XI: **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%**