LADDER: LANGUAGE AND AFFORDANCE DRIVEN DEMONSTRATION ENGINE FOR ROBOTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Robot learning has moved closer to generalist agents that can act in everyday environments, but efficiently collecting high-quality manipulation data for training remains a challenge. Real-world manipulation data collection often relies on human demonstrations, including robot teleoperation and handheld devices that record human manipulation trajectories. While data collection in simulation mitigates this labor-intensive burden, existing approaches often rely on manually acquired affordances for manipulation or carefully designed reward shaping for reinforcement learning before data collection. To efficiently collect manipulation data, we present LADDER, an automated pipeline for demonstration generation given a natural language instruction. In this pipeline, a manipulation knowledge base is built using parameterized instances on which affordances are formulated to support automatic task execution; a multi-agent workflow decomposes a natural language instruction into verified atomic tasks and automatically constructs a scene; affordances of each instance are then mapped to a corresponding asset and extended to a sequence of key poses; trajectories connecting the key poses are curated and used to guide robots to perform each atomic task. After a successful execution, the trajectory and related data are generated. LADDER collects demonstration data on 100 tasks of RoboTwin 2.0 and RoboGen, demonstrating that it improves generation efficiency for both rigid-body and articulated tasks. Furthermore, Vision-Language-Action (VLA) models fine-tuned on datasets synthesized by LADDER exhibit high performance, validating the high quality of our collected demonstrations. Our website is available at `https://anonymous.4open.science/w/ladder-project-D8E2/`.

1 INTRODUCTION

Robots are edging closer to becoming generalist agents capable of performing a broad range of tasks in the real world with the rapid expansion and diversification of robot manipulation data (Black et al., 2024; Kim et al., 2024; Bjorck et al., 2025). However, collecting manipulation datasets in the real world requires amounts of capital and labor (Nasiriany et al., 2024). Thus, some researchers have shifted their focus to collecting data in simulation (Gong et al., 2023; Li et al., 2023; James et al., 2020). Simulation-based data collection has emerged as a promising approach. Recent studies have demonstrated the effectiveness of simulation-generated data for training robot policies at scale and enabling good performances of policies in real-world manipulation tasks (Tao et al., 2024; Deshpande et al., 2026). In simulation, data can be collected with diverse domain randomization. Utilizing this data for training VLA models promotes the generalization capability of robots (Jin et al., 2026; Chen et al., 2025). Moreover, simulators provide direct access to low-level states of each asset and can control robots to perform effective interaction, which meets the demand for efficiently and automatically collecting data.

Despite these advantages, three challenges remain. First, the pipeline of data collection is often not fully automated. For the knowledge of affordance, some prior works label substantial affordances manually for 3D assets, which will become costly and inefficient as the scale of assets grows (Tian et al., 2026; Chen et al., 2025). Second, the process of demonstration generation usually fails to achieve high efficiency. For RL-driven approaches to collect data, they not only require extensive and careful reward shaping for tasks (Qi et al., 2023; Chen et al., 2022), but also suffer from time-consuming policy training across different robot embodiments and tasks

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(Wang et al., 2023; Zhou et al., 2025). Third, existing task-planning methods occasionally fail to generate valid plans. Utilizing Large Language Models (LLMs) to plan a task in robotic simulation often lacks grounding in physical and geometric details (Hua et al., 2024; Wang et al., 2023), which may produce plans that cannot be executed (Ahn et al., 2022) and require iterative refinement based on feedback (Chen et al., 2025).

To address these limitations, we propose **LADDER** (Language and Affordance Driven Demonstration Engine for Robots), an automated and efficient pipeline for demonstration generation that integrates five steps: Knowledge Construction, Task Decomposition, Scene Construction, Affordance Grounding and Task Execution.

Knowledge Construction: Inspired by recent advances in asset modeling (Sun et al., 2024), LADDER leverages parameterized geometric primitive templates to acquire the knowledge of affordance. These templates can be instantiated with specific parameters to create a parameterized instance that represents a given asset. Crucially, affordances are formulated as functions of the parameters which concretely are geometric parameters (e.g., defining part dimensions) and kinematic parameters (e.g., specifying joint axes and motion limits for articulation). Based on this, LADDER constructs a **manipulation knowledge base**. The knowledge base extends parameterized instances to represent assets across a wide range of categories. Affordances are formulated mathematically based on the parameters of corresponding parameterized instances. The affordances can also be directly accessed via their respective affordance functions (e.g., pick, place and hang), which is helpful for next steps.

Task Decomposition and **Scene Construction**: Given a natural language instruction and knowledge (asset categories and affordance functions) from the manipulation knowledge base as input, LADDER uses a multi-agent workflow to perform task decomposition. This decomposition reduces the complexity of an instruction by breaking down a complex goal into a sequence of sub-tasks (Kannan et al., 2024; Zhang et al., 2025). In the workflow, an asset agent extracts the required assets for the instruction while another planning agent extracts affordance functions available in manipulation knowledge base from the instruction. Through collaborative reasoning, the two agents frequently communicate and exchange information with each other based on the instruction to match each affordance function to its intended asset (e.g., assigning `pick` to a mug rather than to a rack). Then they acquire atomic tasks which specify categories of assets and affordance functions (see Fig. 1 middle). Furthermore, the agents output a structured configuration file that includes all assets and ordered atomic tasks needed to perform a task. With this file, a scene agent imports each asset into a region of the scene, configures their physical properties (e.g., static friction coefficients and restitution coefficients) and sets textures, lighting and background (see Fig. 2).

Affordance Grounding and **Task Execution**: According to the configuration file, LADDER acquires the affordance by the corresponding affordance function of each atomic task. Moreover, by obtaining each target object's 6D pose that is hard to acquire in the real world, LADDER maps affordances of parameterized instances precisely onto the actual poses of the assets and extends these actual poses into a sequence of key poses for task execution (e.g., computing the arc trajectory for opening or closing a door based on the actual pose belonging to a handle and a hinge pose). Based on this, LADDER grounds affordances of instances into a series of key poses. Then, it plans a collision-free trajectory of an end-effector between the key poses for task execution. Following a successful task execution, LADDER outputs the generated trajectory and interaction data for downstream policy training.

We evaluate the demonstration collection success rate across 100 tasks of RoboTwin 2.0 (Chen et al., 2025) and RoboGen (Wang et al., 2023). Rigid-body objects are primarily utilized in the 50 tasks of RoboTwin 2.0, whereas articulated objects are utilized in the 50 tasks of RoboGen. By enhancing the success rate, LADDER can improve the efficiency and reduce the cost of demonstration generation. Furthermore, to validate the data collected by LADDER, we fine-tune Vision-Language-Action (VLA) models and benchmark them on RoboTwin 2.0, demonstrating that data collected by LADDER is high-quality for training policies.

In summary, our main contributions are:

- We introduce **LADDER** (language and affordance driven demonstration engine for robots), a language-conditioned, high-quality demonstration generation pipeline. Given an instruction in natural language, LADDER *automatically* controls the robot to achieve the task in an auto-generated scene at a high efficiency and collect the data.

- We construct a **manipulation knowledge base** that can offer precise affordances of each asset with parameterized geometric primitive templates. These templates can represent each asset and formulate affordances with specific parameters. For any asset in the scene, the knowledge base makes the interaction between the asset and the robot easier and high-quality, therefore enabling LADDER to achieve a task at a high success rate.
- We propose a **multi-agent workflow** that efficiently decomposes an instruction in natural language into executable procedures at a high success rate. This is realized by a fine-grained decomposition and reassembly method: two agents first individually decompose the instruction into assets and affordance functions that specify the required manipulation operations; then they collaborate to match each asset and name, which can form ordered atomic tasks. Executing the simpler atomic tasks can finally achieve a task.

2 Related Work

2.1 Robot Learning in Manipulation

Robot manipulation policies have seen advances from classical imitation-learning methods to large-scale generalist robot policies (Kim et al., 2024; Chen et al., 2025). By learning from expert demonstrations, some policies show strong generalization performance, including π0.7 (Physical Intelligence et al., 2026), RDT-1B (Liu et al., 2025),OpenVLA (Kim et al., 2024), G0.5 (Liu et al., 2026), Octo (Team et al., 2024), Qwen-RobotManip (Team, 2026). Behind these impressive performances, robot manipulation data have played a vital role. To further advance this direction, we propose LADDER to collect robot manipulation data across different embodiments with domain randomization.

2.2 Manipulation Data Collection

In order to train robot manipulation policies and reach high performance, researchers have collected some datasets by teleoperation or scripts (Brohan et al., 2022; Walke et al., 2023; Fang et al., 2024; Wu et al., 2024). Real-world data can also be collected without operating a robot at the demonstration site: UMI uses handheld grippers, while DexCap captures human hand motion (Chi et al., 2024; Wang et al., 2024). These approaches reduce the need to deploy robots during collection, but still require people to perform demonstrations. Simulation offers another route to data collection, although collecting demonstrations through human-operated interfaces remains labor-intensive (Liu et al., 2023; Mees et al., 2022; Nasiriany et al., 2026; Li et al., 2023). In order to collect data automatically, reinforcement learning is used for expert data generation (Wang et al., 2023; Ma et al., 2024; Tao et al., 2024). But most of these works need the time and human efforts which is considered not efficient for data collection. In contrast, LADDER can automatically and efficiently collect manipulation data with a natural language instruction.

3 Method

LADDER consists of five steps: Knowledge Construction, Task Decomposition, Scene Construction, Affordance Grounding, and Task Execution. The first step builds the knowledge base of affordances from geometric primitive templates. The remaining four steps turn a natural-language instruction into a verified atomic plan and robot manipulation data.

3.1 Knowledge Construction

Research on human cognition and brain science demonstrates that humans learn about the physical world by perceiving geometric patterns in objects (Ullman, 1996; Palmeri & Gauthier, 2004). Knowledge construction formalizes the same observation: once an object is abstracted as a composition of parameterized geometric primitive templates, the templates together with geometric parameters and kinematic parameters are sufficient both to represent an asset and to then compute its affordances.

Knowledge of Affordance. For a rigid asset, parameterized geometric primitive templates along with geometric parameters can constitute parameterized instances to represent rigid assets (see Fig.

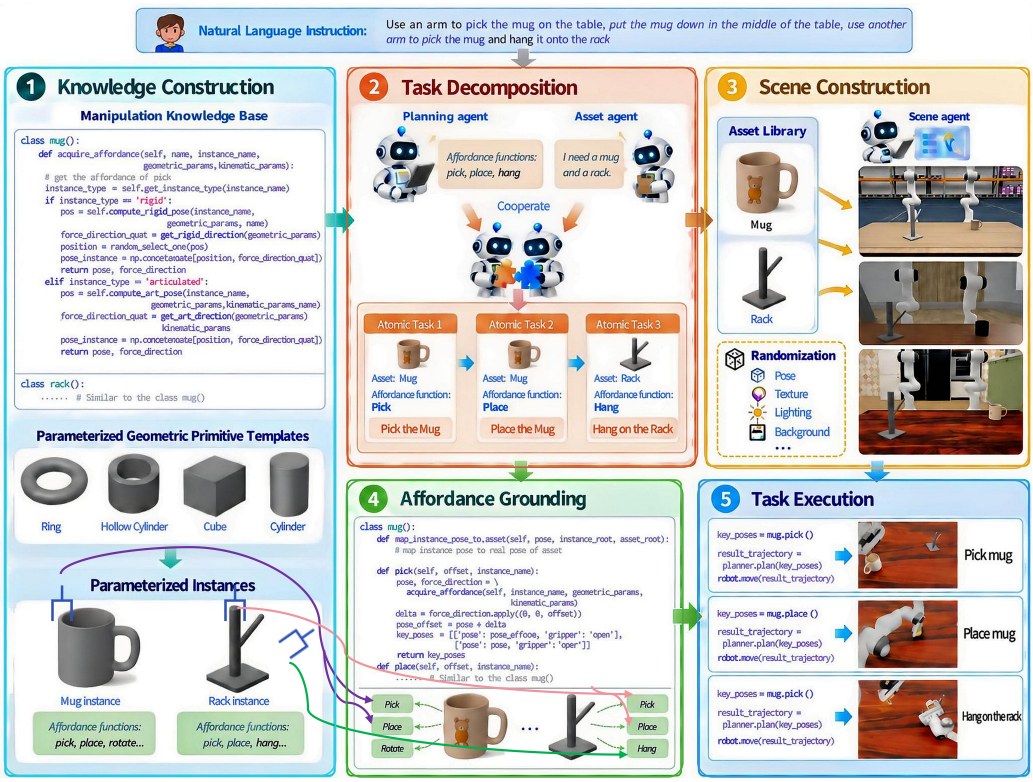


Figure 1: **LADDER Pipeline.** Given a natural-language instruction, LADDER can use a multi-agent workflow to decompose the instruction into atomic tasks and execute these atomic tasks with the help of affordances acquired from manipulation knowledge base.

1 left). For an articulated asset, LADDER also needs kinematic parameters to form parameterized instances and represent articulated assets. The parameters are determined by a web-based platform that automatically adjusts them with a single click. For the acquisition of affordances, they are formulated automatically by the parameters of parameterized instances (see Fig. 1 left bottom). And the feasible 6D poses associated with an affordance form a continuous region, which yields abundant feasible poses for robot interaction in contrast to limited poses labeled manually.

Manipulation Knowledge Base. Building on ConceptFactory (Sun et al., 2024), we construct a Manipulation Knowledge Base. It uses parameterized geometric primitive templates to represent a broader variety of assets, including assets from 68 asset categories of RoboTwin 2.0 (Chen et al., 2025), which are used primarily for rigid-body tasks, and assets from 46 articulated categories in PartNet-Mobility (Xiang et al., 2020). It also computes a richer set of affordances from the parameters, which is essential for task decomposition Sec. 3.2 and affordance grounding Sec. 3.4. Concretely, the knowledge base contains 4,584 assets. Each asset is associated with its own set of affordances and corresponding affordance functions (e.g., `pick`, `place`, `hang`, and `open`), amounting to 23k affordances in total. The Manipulation Knowledge Base supplies Sec. 3.2 with two kinds of information: affordance functions and the categories of assets. This information will help agents in Sec. 3.2 decompose instructions into executable atomic tasks and build scenes using available assets.

3.2 Task Decomposition

Task decomposition uses the fine-grained decomposition and reassembly method in two stages: knowledge extraction identifies the required asset categories and affordance functions, while knowledge-guided task planning assembles them into ordered atomic tasks. The resulting plan is validated and repaired if needed.

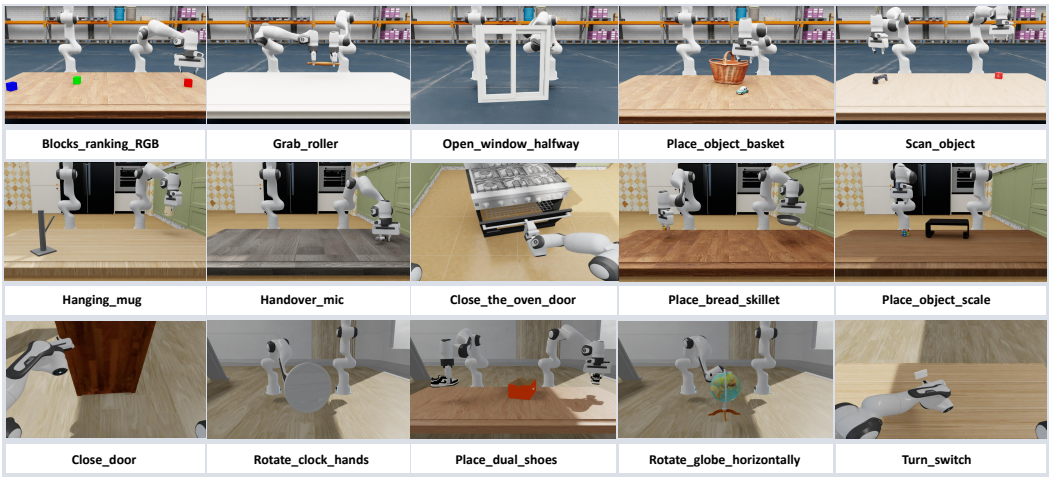


Figure 2: Examples of rigid-body and articulated tasks performed by LADDER under randomized domain settings.

Knowledge Extraction. LADDER uses two agents to extract task knowledge from a natural language instruction. An *asset agent* identifies the required asset categories, while a *planning agent* extracts affordance functions in parallel, both using the knowledge from manipulation knowledge base (Fig. 1, middle). The agents use their language understanding capabilities to map diverse expressions to corresponding knowledge-base terms, allowing the workflow to handle different instructions of a task.

Knowledge-Guided Task Planning. The planning and asset agents iteratively exchange information to match each extracted affordance function to an extracted asset (e.g., `pick` to a mug rather than a rack). For each affordance function offered by the planning agent, the asset agent identifies compatible assets. If none exists, the planning agent revises the function up to three times. If matching still fails, the asset agent re-extracts the assets, and the agents make up to three additional matching attempts. If all additional attempts fail, task decomposition is declared unsuccessful. Each successful match yields an atomic task specifying an asset category and an affordance function (see Fig. 1 middle). The planning agent then orders the atomic tasks. The asset agent checks this plan and reports any errors with reasons, allowing the planning agent to revise it when needed. Finally, a structured configuration is acquired that records the task name, each asset's category and ID (to distinguish between different objects of the same category), and the atomic plan (Fig. 1, middle; Appendix B).

Validation and Repair. To validate the planning of the multi-agent workflow, semantic constraints are checked locally: every asset category and affordance function must belong to the knowledge base; every referenced ID must exist; unused assets are rejected; and simultaneous bimanual atomic tasks must be specified as simultaneous tasks. Invalid plans are returned to the agents with the validator's error message and repaired until the configuration is valid, or the attempt is discarded.

3.3 Scene Construction

Asset Import. A *scene agent* builds a scene in Isaac Lab (Mittal et al., 2025) using the configuration from task decomposition Sec. 3.2. For each required asset, it randomly selects a candidate from the corresponding category in the asset library, sets its physical properties (e.g., static/dynamic friction and restitution), and imports it at a random pose within a region. Articulated assets start with joint configurations consistent with the planned skill (e.g., an open door to be closed, or a closed microwave door to be opened). Robots, the table, lights, ground, and cameras are loaded by default and are outside the asset agent's scope.

Domain Randomization. Recent works have demonstrated that data with domain randomization can enhance policy generalization (Jin et al., 2026; Chen et al., 2025). So when assets are imported

into the simulator, LADDER will randomly and automatically set textures, background, lighting, camera pose, mass, and friction.

3.4 Affordance Grounding

Affordance grounding is driven by the configuration file and the Manipulation Knowledge Base. For each atomic task, LADDER looks up the ID of each asset in Sec. 3.2. Using this ID, rather than the asset category, LADDER retrieves the target asset's 6D pose from the simulator, since multiple assets of the same category may appear in a single task. LADDER aligns the coordinate frame of the parameterized instance with that of the target asset. It then applies the resulting rigid transformation to the instance's affordance region, mapping the region onto the asset in the scene. LADDER samples a pose from that affordance region and expands it into a sequence of key poses with relative parameters. Concretely, the grounding of the affordance differs slightly between rigid and articulated objects. For rigid-body tasks, some affordances with affordance functions such as place, hang, stack, and click follow the same pattern, except that the poses of affordances are taken from the receiving asset (rack hook, plate surface, button) rather than from the grasped asset. LADDER then expands this world pose into a waypoint sequence: an approach pose offset along the gripper local z-axis, a contact pose, a gripper command (open/close), and, when needed, a retreat or handover pose. For articulated tasks, an atomic task involving a revolute joint interpolates an arc around the hinge, clipped by the joint limits and the requested requirements (e.g., open halfway). A prismatic atomic task instead slides the handle along the joint axis. And the end-effector orientation is kept consistent with the handle frame. Besides, LADDER uses the same procedure as for rigid-body tasks to extend poses for motions like grasping a handle or pressing a button.

3.5 Task Execution

Obstacle Construction. With asset information available in the simulator, LADDER directly acquires collision geometries of each asset from Isaac Sim (NVIDIA) and converts these collision geometries into a mesh representation as obstacles for CuRobo V2 (Sundaralingam et al., 2026). When the end-effector of robots reaches a pre-grasp pose, LADDER allows the collision of the end-effector and treats the grasped asset or part as an additional link of the robot after a successful grasp. Then the collision of the end-effector is banned again. But the collision between assets or parts of assets and the end-effector is allowed.

Trajectory Execution and Demonstration Collection. Given the key poses obtained through affordance grounding Sec. 3.4, LADDER leverages CuRobo to plan a collision-free trajectory between the key poses. The robot executes the planned trajectory together with the corresponding gripper commands to complete a task (see Fig. 2). After a successful execution, LADDER records proprioception, end-effector poses, gripper states, and camera observations, and formats the trajectory for downstream VLA training. Failed executions (unreachable IK, dropped objects, or unmet success metrics) are discarded, so the exported dataset contains only completed atomic plans.

4 Experiments

We evaluate LADDER's collection efficiency on rigid-body and articulated tasks, the quality of its demonstrations for policy learning, the manipulation knowledge base, and the multi-agent workflow.

4.1 Demonstration Collection Efficiency

4.1.1 Rigid-body tasks

Overview. We compare LADDER with the official RoboTwin 2.0 demonstration collection pipeline on the full 50-task suite, using the same embodiment (Aloha-AgileX), background configuration, and per-task success checkers. The metric is the proportion of runs in which the robot successfully performs a task, with 100 trials per task.

Main result. On the 50 tasks, LADDER attains 94.2% average collection success versus 78.8% for RoboTwin 2.0, a 15.4-point gain. Improvements are especially large for contact-rich or

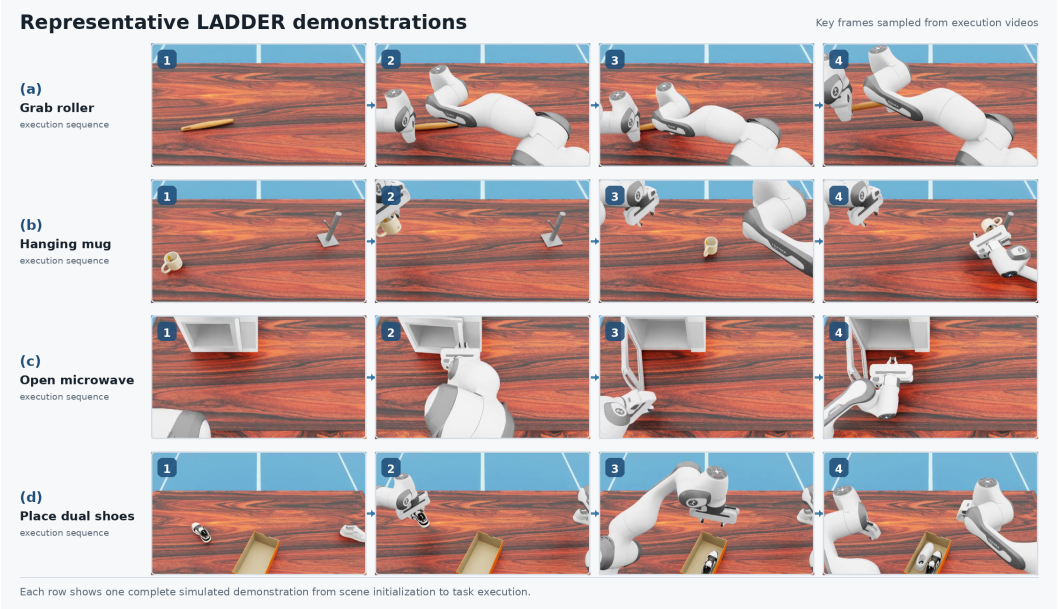


Figure 3: **Representative LADDER demonstrations.** Each row shows key frames sampled from one execution video: (a) `grab roller`, (b) `hanging mug`, (c) `open microwave`, and (d) `place dual shoes`. The examples cover both single-arm and coordinated bimanual interactions, illustrating that LADDER can execute diverse atomic tasks under different asset configurations and record the resulting trajectories.

Table 1: Comparison of task success rates (%) of LADDER and RoboTwin 2.0. *Some representative results* are shown in this table. Full results in Appendix C.2

Task	RoboTwin 2.0	LADDER	Task	RoboTwin 2.0	LADDER
Put object cabinet	14.0	95.0	Adjust bottle	93.0	98.0
Scan object	4.0	85.0	Beat block hammer	64.0	95.0
Place bread skillet	34.0	100.0	Blocks ranking rgb	96.0	98.0
Lift pot	27.0	73.0	Move pillbottle pad	67.0	86.0
Stack bowls three	43.0	98.0	**Average success rate**	78.8	94.2

spatially tight skills, where a closed-form affordance supplies a key pose on the functional part: `put object cabinet` (14%→95%), `scan object` (4%→85%), `place bread skillet` (34%→100%), `lift pot` (27%→73%), and `stack bowls three` (43%→98%). These gains also indicate that the multi-agent workflow maps natural-language instructions to executable procedures in a feasible order, supporting efficient data collection.

4.1.2 Articulated Tasks

Overview. Rigid-body pick-and-place does not test hinge and slider skills. We therefore evaluate the demonstration generation of LADDER on 50 articulated tasks from RoboGen under RoboGen's asset-selection, placement, and scene configuration, using the same success metric; details are in Appendix D.1.

Main result. LADDER reaches 89.5% on the 50 skills versus RoboGen's 74.5%, a 15.0-point gain. The largest improvements occur where reward-driven skill learning is brittle and the manipulation knowledge base provides a useful interaction affordance, including opening a dispenser lid (0%→100%), opening a microwave door (25%→100%), closing a dispenser lid (25%→100%), moving a clock hand (38%→100%), and turning on a lamp (25%→100%). LADDER collects these demonstrations without task-specific skill training, whereas RoboGen requires a substantial skill-learning budget for each task.

Table 2: Comparison of task success rates (%) of LADDER and RoboGen. *Some representative results* are shown in this table. Full results in Appendix D.2

Task	RoboGen	LADDER	Task	RoboGen	LADDER
Open dispenser lid	0.0	100.0	Open door	50.0	100.0
Open microwave door	25.0	100.0	Open dishwasher door	60.0	100.0
Close dispenser lid	25.0	100.0	Flush the toilet	67.0	100.0
Turn on lamp	25.0	100.0	Turn off faucet	67.0	100.0
Close door	50.0	100.0	**Average success rate**	74.5	89.5

Table 3: Success rates (%) on RoboTwin 2.0 for policies trained on RoboTwin 2.0 expert data or LADDER demonstrations. RT: same architecture trained on RoboTwin 2.0 expert data. LADDER: trained on LADDER demonstrations. *Some representative results* are shown in this table. Full results in Appendix E.

	$\pi_{0.5}$				DP3				X-VLA			
	Clean		Hard		Clean		Hard		Clean		Hard	
Task	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER
Click alarmclock	65.0	96.0	50.0	74.0	77.0	100.0	14.0	19.0	62.0	77.0	43.0	56.0
Place dual shoes	70.0	79.0	45.0	65.0	13.0	40.0	0.0	1.0	30.0	47.0	12.0	15.0
Place object basket	70.0	76.0	29.0	40.0	65.0	87.0	0.0	8.0	81.0	95.0	6.0	25.0
Place shoe	82.0	94.0	50.0	72.0	58.0	83.0	2.0	7.0	83.0	99.0	39.0	52.0
Press stapler	79.0	96.0	63.0	78.0	69.0	70.0	3.0	9.0	87.0	88.0	73.0	87.0
Average	70.6	77.1	46.0	54.4	55.24	64.34	4.96	8.46	68.0	73.8	20.9	29.0

4.2 Demonstration Quality for Policy Learning

Overview. To evaluate the quality of data collected by LADDER, we replace the RoboTwin 2.0 expert dataset for training policies with LADDER demonstrations on the same 50 tasks (50 trajectories per task, in the same format), fine-tune $\pi_{0.5}$ (Intelligence et al., 2025) and X-VLA (Zheng et al., 2026), and train DP3 (Ze et al., 2024). We compare with the corresponding models trained on RoboTwin 2.0 data under identical architectures, tasks, and `clean`/`hard` protocols. All models follow RoboTwin 2.0's training configurations and hardware setup, and are evaluated with its official success checkers; details are in Appendix E.1.

Main result. Replacing RoboTwin 2.0 data with LADDER demonstrations improves all three policies: on clean/hard, $\pi_{0.5}$ rises from 70.6/46.0 to 77.1/54.4 (+6.5/+8.4), DP3 from 55.24/4.96 to 64.34/8.46 (+9.10/+3.50), and X-VLA from 68.0/20.9 to 73.8/29.0 (+5.8/+8.1). Consistent gains across architectures support the quality of LADDER demonstrations. Larger gains on precision-demanding skills such as clicking and placing are also consistent with accurate affordances producing better trajectories.

4.3 Evaluation of the Manipulation Knowledge Base

Overview. We evaluate the manipulation knowledge base through its contribution to RoboGen skill learning. Both settings use the same generated articulated-object tasks, URDF assets, RoboGen (Wang et al., 2023) skill-learning pipeline, GPT-5.6 reward generator, and Soft Actor-Critic (SAC) training. We use GPT-5.6 in place of the GPT-4 used in the original RoboGen setup. The settings differ only in the information provided during reward generation: the baseline uses RoboGen's original approach, whereas the augmented setting additionally provides task-relevant knowledge from our manipulation knowledge base. We report average task success.

Main result. Providing knowledge-base information raises average success from 76.4% to 80.9%, a 4.5-point gain. The largest improvements occur when the original prompt does not explicitly identify the target part or required joint motion, including opening a microwave door (25% → 100%), turning on a lamp (25% → 100%), and moving a clock hand (38% → 75%). Success also increases from 50% to 100% for closing a door and turning on a coffee machine, and from 50% to 88% for opening a stapler lid. These improvements are consistent with affordance information helping GPT-

Table 4: Success rates (%) on RoboGen articulated tasks. LADDER reports demonstration collection success. MKB denotes the Manipulation Knowledge Base. *Some representative results* are shown in this table. Full results in Appendix F.

Task	LADDER	RoboGen+MKB	RoboGen (GPT-5.6)
Open microwave door	100.0	100.0	25.0
Turn on lamp	100.0	100.0	25.0
Close door	100.0	100.0	50.0
Open stapler lid	88.0	88.0	50.0
Average success rate	89.5	80.9	76.4

5.6 generate rewards that direct the end-effector toward the target part and corresponding joint. For tasks that RoboGen already performs successfully, the two settings are comparable or the baseline can perform better.

4.4 Evaluation of the multi-agent workflow

Overview. Our multi-agent workflow performs fine-grained decomposition and reassembly. We compare it with a single-LLM baseline using the same knowledge base but without separating asset extraction from planning. Both use GPT-5.6 (OpenAI, 2026) and are evaluated on 25 RoboTwin 2.0 tasks with 10 instructions per task; 17 human experts assess each decomposition. We report planning success rate and tokens per instruction.

Table 5: Overall performance comparison between multi-agent workflow and LLM. ASR is the average planning success rate. Full results in Appendix G

Method	ASR	Token
multi-agent workflow	98.0	1388.4
LLM	81.2	1638.2

Main result. The full workflow achieves 98.0% planning success with 1388.4 tokens, versus 81.2% and 1638.2 tokens for the baseline. Without fine-grained decomposition, multi-step tasks such as blocks ranking RGB and place bread skillet can be compressed into too few steps for the robot to execute. Without reassembly, the plan can also fail to specify which object should be acted on when several same-type assets are present.

4.5 Discussion

LADDER can execute long-horizon instructions by composing atomic tasks and recover from unexpected disturbances. It monitors object and gripper states, such as whether a bowl drops from the gripper or a bottle is knocked over. If a disturbance occurs, LADDER stops the current waypoint sequence, updates the scene state, and attempts recovery before resuming the remaining plan. The long-horizon restaurant-operations task in Fig. 4 illustrates task composition and recovery from interruptions. This enables recovery-data collection that is expensive to obtain in the real world; these demonstrations are not used for VLA fine-tuning in our experiments.

5 Conclusion

We introduced LADDER, a language-conditioned pipeline for automatically collecting manipulation demonstrations in simulation. Its manipulation knowledge base and multi-agent workflow enable task decomposition, affordance grounding, and trajectory execution without task-specific skill training. Higher collection success rates and improved downstream policy performance support the efficiency and quality of the generated data.

AI USE STATEMENT

In this work, generative AI was used in two distinct roles. First, GPT-5.6 was an evaluated component of the research workflow. Within LADDER, the asset and planning agents used GPT-5.6 to parse natural-language instructions, map diverse expressions to terms in the manipulation knowledge base, produce structured task configurations, and revise invalid plans. In the RoboGen-based knowledge-base experiment, GPT-5.6 generated task-specific reward functions both with and without the additional affordance information described in the paper. These uses are therefore part of the evaluated methodology rather than an unreported post-processing step. Second, generative AI tools assisted with manuscript drafting and English-language editing, code implementation and debugging.

The authors reviewed all AI-assisted outputs. AI-assisted code was inspected, executed, and checked against the intended implementation; manuscript text was checked for correctness, consistency, and anonymity; and reported results, numerical comparisons, citations, and figure content were checked against the source materials and experimental records. We did not use generative AI to fabricate experimental data or replace author judgment in the final scientific claims. We take responsibility for the final content of this paper.

REPRODUCIBILITY STATEMENT

We will release the codes of this paper. The release will include the parameterized asset and affordance definitions, the structured configuration format, the retry and semantic-validation rules used by the multi-agent workflow, scene-construction and affordance-grounding code, trajectory-collection scripts, and the training and evaluation configurations for the policy experiments. The main paper describes the pipeline, task suites, baselines, success metrics, and evaluation protocol. Appendix B documents the structured configuration, prompts, packed knowledge-base format, and local semantic checks; Appendices C.3 and D.1 provide collection-time and articulated-task details; and Appendix E.1 specifies policy training and evaluation settings. The remaining experimental appendices report per-task results and the protocols for evaluating the knowledge base and multi-agent workflow. Together with the public RoboTwin 2.0 and RoboGen task definitions, these materials are intended to make the reported collection, policy-learning, and ablation results independently reproducible.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A Vision-Language-Action Flow Model for General Robot Control. *arXiv preprint arXiv:2410.24164*, 2024.

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.

Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on robot learning*, pp. 297–307. PMLR, 2022.

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

Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

Abhay Deshpande, Maya Guru, Rose Hendrix, Snehal Jauhri, Ainaz Eftekhar, Rohun Tripathi, Max Argus, Jordi Salvador, Haoquan Fang, Matthew Wallingford, et al. Molmob0t: Large-scale simulation enables zero-shot manipulation. *arXiv preprint arXiv:2603.16861*, 2026.

Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 653–660. IEEE, 2024.

Ran Gong, Jiangyong Huang, Yizhou Zhao, Haoran Geng, Xiaofeng Gao, Qingyang Wu, Wensi Ai, Ziheng Zhou, Demetri Terzopoulos, Song-Chun Zhu, et al. Arnold: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 20483–20495, 2023.

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. *arXiv preprint arXiv:2410.03645*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

Ruixing Jin, Zicheng Zhu, Ruixiang Ouyang, Sheng Xu, Bo Yue, Zhizheng Wu, and Guiliang Liu. Grounding sim-to-real generalization in dexterous manipulation: An empirical study with vision-language-action models. *arXiv preprint arXiv:2603.22876*, 2026.

Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. Smart-llm: Smart multi-agent robot task planning using large language models. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12140–12147. IEEE, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pp. 80–93. PMLR, 2023.

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.

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. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025.

Yicheng Liu, Zibin Dong, Baijun Ye, Tianyuan Yuan, Tao Jiang, Anqi Yang, Shicheng Cao, Haonan Liu, Yue Sun, Zihan Guo, Xiao Liu, Ke Dong, Changxun Pan, Chenru Wu, Tailai Cheng, Xiaoshu Ren, Xinlei Zhang, Jianning Cui, Zijie Zhao, Haoyu Zhang, Kaiming Xu, Haodong Yang, Bowen Zhang, Jiahui Niu, Shaoting Zhu, Shiduo Zhang, and Hang Zhao. G0.5: One autoregressive stream for robot reasoning and action, 2026. URL `https://arxiv.org/abs/2608.11739`.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pp. 26516–26560, 2024.

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.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen, Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi "Jim" Fan, Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025. URL https://arxiv.org/abs/2511.04831.

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. *arXiv preprint arXiv:2406.02523*, 2024.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026.

NVIDIA. Isaac Sim. URL https://github.com/isaac-sim/IsaacSim.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, July 2026. Published July 9, 2026.

Thomas J Palmeri and Isabel Gauthier. Visual object understanding. *Nature Reviews Neuroscience*, 5(4):291–303, 2004.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: A Steerable Generalist Robotic Foundation Model with Emergent Capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Haozhi Qi, Ashish Kumar, Roberto Calandra, Yi Ma, and Jitendra Malik. In-hand object rotation via rapid motor adaptation. In *Conference on Robot Learning*, pp. 1722–1732. PMLR, 2023.

Jianhua Sun, Yuxuan Li, Longfei Xu, Nange Wang, Jiude Wei, Yining Zhang, and Cewu Lu. Conceptfactory: Facilitate 3d object knowledge annotation with object conceptualization. *Advances in Neural Information Processing Systems*, 37:75454–75467, 2024.

Balakumar Sundaralingam, Adithyavairavan Murali, and Stan Birchfield. curobov2: Dynamics-aware motion generation with depth-fused distance fields for high-dof robots, 2026.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

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.

Qwen Team. Qwen-robotmanip technical report: Alignment unlocks scale for robotic manipulation foundation models, 2026. URL https://arxiv.org/abs/2606.17846.

Yang Tian, Yuyin Yang, Yiman Xie, Zetao Cai, Xu Shi, Ning Gao, Hangxu Liu, Xuekun Jiang, Zherui Qiu, Feng Yuan, et al. Interndata-a1: Pioneering high-fidelity synthetic data for pre-training generalist policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 976–985, 2026.

Shimon Ullman. *High-level vision: Object recognition and visual cognition*. The MIT Press, 1996.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on robot learning*, pp. 1723–1736. PMLR, 2023.

Chen Wang, Haochen Shi, Weizhuo Wang, Ruohan Zhang, Li Fei-Fei, and C Karen Liu. Dexcap: Scalable and portable mocap data collection system for dexterous manipulation. *arXiv preprint arXiv:2403.07788*, 2024.

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. *arXiv preprint arXiv:2311.01455*, 2023.

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.

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

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

Xiaohan Zhang, Yan Ding, Yohei Hayamizu, Zainab Altaweel, Yifeng Zhu, Yuke Zhu, Peter Stone, Chris Paxton, and Shiqi Zhang. Llm-grop: Visually grounded robot task and motion planning with large language models. *The International Journal of Robotics Research*, pp. 02783649251378196, 2025.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.

Zihan Zhou, Animesh Garg, Ajay Mandlekar, and Caelan Garrett. Reinforcegen: Hybrid skill policies with automated data generation and reinforcement learning. *arXiv preprint arXiv:2512.16861*, 2025.

Appendix

TABLE OF CONTENTS

A Long-horizon execution and disturbance recovery **15**

B Additional method details **15**

B.1 Structured configuration 15

B.2 Multi-agent prompts 16

B.3 Packed knowledge base 19

B.4 Local semantic checks 19

C Demonstration collection on rigid-body tasks **19**

C.1 Evaluation protocol 19

C.2 Per-task results 19

C.3 Collection time 21

D Demonstration collection on articulated tasks **21**

D.1 Evaluation protocol 21

D.2 Per-task results 21

E Demonstration quality for policy learning **22**

E.1 Policy training and evaluation 22

E.2 Per-task results 24

F Evaluation of the manipulation knowledge base **25**

F.1 Experimental setup 25

F.2 Per-task results 25

G Evaluation of the multi-agent workflow **26**

G.1 Experimental setup 26

G.2 Per-task results 27

A LONG-HORIZON EXECUTION AND DISTURBANCE RECOVERY

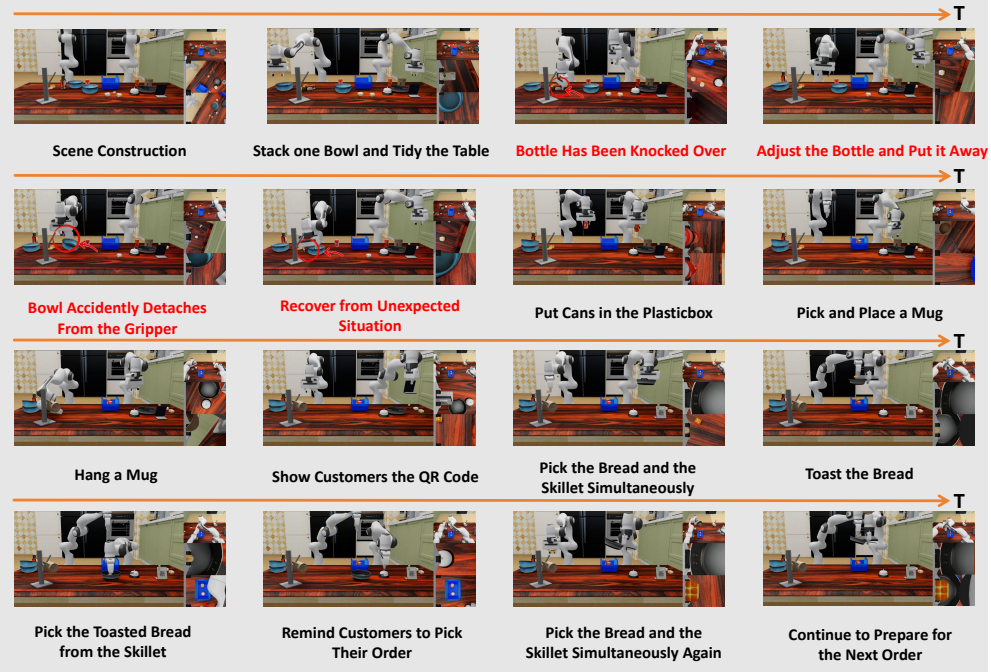


Figure 4: A long-horizon restaurant-operations task used to illustrate composition of atomic tasks and recovery from unexpected disturbances.

As discussed in this paper, a long-horizon instruction is handled by composing more atomic tasks in the configuration of Appendix B.1. If an object is dropped or knocked over, LADDER stops the current waypoint sequence, updates object states, and resumes the remaining plan.

B ADDITIONAL METHOD DETAILS

In this section, we record the structured configuration emitted by the multi-agent workflow, the prompts used by the two agents, the packed knowledge-base format injected at runtime, and the local semantic checks used for repair.

B.1 STRUCTURED CONFIGURATION

After knowledge extraction and matching (Sec. 3.2), the workflow writes a configuration with three fields: `task_name`; `assets`, each with a category and an id that distinguishes instances of the same category; and `atomic_plan`, an ordered list of atomic tasks. Each atomic task names one asset id and one affordance function from the Manipulation Knowledge Base. Table, robot, lights, ground, and cameras are loaded by default and are not listed by the asset agent. Figure 5 shows the configuration for the hanging-mug instruction used in Fig. 1: `pick` and `place` are bound to the mug, and `hang` is bound to the rack.

Atomic task plan (`hanging_mug`)

```
{
  "task_name": "hanging_mug",
  "assets": [
    {"id": "mug_1", "type": "mug", "asset_kind": "auto"},
    {"id": "rack_1", "type": "rack", "asset_kind": "auto"}
  ],
  "atomic_plan": [
    {"asset": "mug_1", "atom": "pick", "params": {}},
    {"asset": "mug_1", "atom": "place", "params": {}},
    {"asset": "rack_1", "atom": "hang", "params": {}}
  ]
}
```

Figure 5: Structured configuration produced from a hanging-mug instruction. This is the file consumed by scene construction (Sec. 3.3) and affordance grounding (Sec. 3.4).

B.2 MULTI-AGENT PROMPTS

The asset agent and the planning agent run in parallel. They do not enumerate every category or affordance in the prompt. Both receive the instruction and a packed object `ATOMS` (Appendix B.3). A third matching step binds each affordance name to one extracted category and emits Fig. 5. If the local validator rejects the file, the error string is appended and the agents repair the JSON.

Multi-agent planner prompts. The asset and planning agents first extract asset categories and affordance functions in parallel. For each proposed affordance, the asset agent checks compatible categories. If matching fails, the planning agent revises the affordance up to three times. The asset agent then re-extracts the categories if necessary, followed by at most three further matching attempts. After the plan is assembled, the asset agent reviews it, and a local validator checks the resulting configuration.

```
[1. Asset agent: initial asset extraction]

You are the asset agent in LADDER. Identify the asset categories required
to execute the instruction using the Manipulation Knowledge Base (ATOMS).

Instruction: {instruction}
ATOMS: {packed_knowledge_base}

Output JSON only:
{"categories": ["mug", "rack"]}

Rules:
- Every category must be a key in ATOMS.types.
- Include manipulated objects and objects used as targets, containers,
  or supports.
- Map alternative names in the instruction to categories in ATOMS.types.
- Do not invent categories.
- Do not include the table, robot, lights, ground, or cameras unless the
  instruction requires manipulating them and they occur in ATOMS.types.
- Do not output affordance names or an execution plan.


[2. Planning agent: initial affordance extraction]

You are the planning agent in LADDER. Identify the affordance functions
needed to execute the instruction using ATOMS. Run this step in parallel
with the asset agent's initial extraction.

Instruction: {instruction}
ATOMS: {packed_knowledge_base}

Output JSON only:
{"affordances": ["pick", "place", "hang"]}

Rules:
- Every name must belong to the expanded affordance set of ATOMS;
  expand "@family" entries before checking membership.
- Map different expressions to the corresponding names in ATOMS.
- Include the functions needed to complete the instruction, in the
  execution order implied by the instruction.
- If simultaneous two-arm execution is required, use the corresponding
  simultaneous affordance names available in ATOMS.
- Do not output asset categories or a complete plan.
```

```
[3. Asset agent: compatibility check]

You are the asset agent in LADDER. The planning agent has proposed one
affordance. Check which extracted asset categories support it in ATOMS.

Instruction: {instruction}
Extracted categories: {categories}
Proposed affordance: {affordance}
ATOMS: {packed_knowledge_base}

Output JSON only:
{"affordance": "pick",
 "compatible_categories": ["mug"],
 "status": "matched"}

Rules:
- Return only categories present in the extracted categories and ATOMS.types.
- A category is compatible only if the proposed affordance belongs to its
  expanded affordance set in ATOMS.
- If none is compatible, return an empty compatible_categories list and
  set status to "no_match".
- Report all compatible extracted categories. The planning agent will
  choose the intended asset using the instruction.
- Do not infer compatibility from general world knowledge.


[4. Planning agent: revise an unmatched affordance]

You are the planning agent in LADDER. The proposed affordance is not
supported by any extracted asset category. Propose another affordance
that preserves the action required by the instruction.

Instruction: {instruction}
Extracted categories: {categories}
Unmatched affordance: {affordance}
Previously rejected affordances: {rejected_affordances}
ATOMS: {packed_knowledge_base}

Output JSON only:
{"affordance": "alternative_name"}

Rules:
- The new name must belong to the expanded affordance set of ATOMS.
- Do not repeat a rejected affordance.
- Preserve the intended action and outcome.
- If no suitable alternative exists, return {"affordance": null}.
- Do not invent asset categories or output a complete plan.


[5. Asset agent: re-extract assets]

You are the asset agent in LADDER. Matching has failed after the allowed
affordance revisions. Re-examine whether the extracted categories were
incomplete or incorrect.

Instruction: {instruction}
Previous categories: {categories}
Unmatched action: {unmatched_action}
Matching history: {matching_history}
ATOMS: {packed_knowledge_base}

Output JSON only:
{"categories": ["mug", "rack"]}

Rules:
- Every category must be a key in ATOMS.types.
- Include manipulated objects and required targets, containers, or supports.
- Correct omitted or incorrectly mapped categories.
- Do not add an unrelated category merely to make an affordance match.
- Do not output affordance names or a complete plan.


[6. Planning agent: assemble the atomic plan]

You are the planning agent in LADDER. The affordances below have passed
compatibility checks. Match each one to the asset intended by the
instruction and assemble an executable atomic plan.

Instruction: {instruction}
Extracted categories: {categories}
Validated affordances and compatible categories: {validated_matches}
```

```
ATOMS: {packed_knowledge_base}

Output JSON only:
{"task_name": "hang_mug",
 "assets": [
   {"id": "mug_1", "type": "mug", "asset_kind": "auto"},
   {"id": "rack_1", "type": "rack", "asset_kind": "auto"}
 ],
 "atomic_plan": [
   {"asset": "mug_1", "atom": "pick", "params": {}},
   {"asset": "mug_1", "atom": "place", "params": {}},
   {"asset": "rack_1", "atom": "hang", "params": {}}
 ]}

Rules:
- Use the instruction to choose the intended asset among compatible ones.
- Every type must be an extracted category and a key in ATOMS.types.
- Every atom must belong to the expanded affordance set of its asset type.
- Give distinct objects of the same category distinct IDs.
- Every atomic_plan.asset must reference an ID in assets.
- Do not include unused assets or add a table asset.
- Order atomic tasks according to the instruction and task dependencies.
- Use a simultaneous affordance name from ATOMS when required.
- If no valid plan can represent the instruction, return
  {"status": "failed"} instead of inventing steps.


[7. Asset agent: review the plan]

You are the asset agent in LADDER. Review the planning agent's
configuration against the instruction and ATOMS. Report specific errors.

Instruction: {instruction}
Configuration: {configuration}
Extracted categories: {categories}
Validated matches: {validated_matches}
ATOMS: {packed_knowledge_base}

Output JSON only:
{"valid": true, "errors": []}

Rules:
- Check whether each affordance is assigned to the intended asset.
- Check whether required objects and actions are present.
- Check task order, dependencies, and required simultaneous execution.
- If the plan is incorrect, set valid to false and list specific reasons.
- Do not silently rewrite the configuration.


[8. Planning agent: revise after review]

You are the planning agent in LADDER. Revise the configuration in
response to the asset agent's errors. Output the complete configuration.

Instruction: {instruction}
Previous configuration: {configuration}
Asset agent's errors: {review_errors}
ATOMS: {packed_knowledge_base}

Use the JSON schema from Step 6. Output JSON only.

Rules:
- Address each reported error while preserving correct parts of the plan.
- Use only categories and expanded affordances supported by ATOMS.
- Keep IDs consistent and remove unused assets.
- Preserve the required task order and simultaneity.
- If no valid plan can be produced, return {"status": "failed"}.


[9. Validation and repair]

The configuration failed local semantic validation. Repair it and output
the complete corrected configuration using the schema from Step 6.

Instruction: {instruction}
Previous configuration: {configuration}
Validator errors: {error}
ATOMS: {packed_knowledge_base}

Rules:
- Every asset type must be a key in ATOMS.types.
- Every atom must belong to the expanded affordance set of its asset type.
- Every referenced asset ID must exist in assets.
```

```
- Remove unused assets.
- A simultaneous bimanual task must use an appropriate simultaneous
  affordance name available in ATOMS.
- Preserve the instruction's meaning. Do not invent categories or atoms.
- If the configuration cannot be repaired, return {"status": "failed"}.
- Output JSON only.
```

B.3 PACKED KNOWLEDGE BASE

The following example illustrates the packed knowledge-base format supplied to the agents.

Packed ATOMS (format)

```
{
  "families": {
    "block": ["beat","hand_over","stack_top",
              "stack_middle","stack_bottom"],
    "sim_pick": ["pick_simultaneously_left",
                 "pick_simultaneously_right"],
    "pick3": ["pick","@sim_pick"],
    "into2": ["into_1","into_2"]
  },
  "types": {
    "mug": ["pick","place"],
    "rack": ["hang"],
    "bottle": ["pick","shake","shake_horizontally",
               "@sim_pick"],
    "block_small": ["@block","place_small"],
    "block_middle": ["@block","place_middle"],
    "block_big": ["@block","place_big"],
    "cabinet": ["open","into"]
  }
}
```

Figure 6: Packed knowledge-base fragment injected as {packed_knowledge_base}. The full table covers the categories used in Sec. 3.1; this listing only shows the format.

B.4 LOCAL SEMANTIC CHECKS

The validator in Sec. 3.2 is rule-based and does not call the LLM. A configuration is accepted only if (i) every `type` is a key of `ATOMS.types`; (ii) every `atom` belongs to the expanded atom set of that type; (iii) every `atomic_plan.asset` exists in `assets[].id`; (iv) every listed asset is used by at least one step; (v) if the instruction requires simultaneous two-arm motion, the corresponding atoms contain `simultaneously`. Failed files are returned with the error string in the repair prompt. After a bounded number of repairs the attempt is discarded and does not enter the demonstration set.

C DEMONSTRATION COLLECTION ON RIGID-BODY TASKS

C.1 EVALUATION PROTOCOL

Collection success is the fraction of attempted executions that complete the task and are retained as valid demonstrations; it is not the success rate of a learned VLA. For RoboTwin 2.0, task completion is determined by the official success checker. LADDER rollouts that fail inverse kinematics, drop an object, or do not meet the corresponding task criterion are discarded.

C.2 PER-TASK RESULTS

Table 6: Per-task demonstration-collection success rate (%) on 50 RoboTwin 2.0 tasks.

Task	RoboTwin 2.0	LADDER
adjust bottle	93.0	98.0
place cans plasticbox	100.0	100.0
beat block hammer	64.0	95.0
place container plate	89.0	100.0
blocks ranking rgb	96.0	98.0
place dual shoes	77.0	88.0
blocks ranking size	96.0	97.0
place empty cup	92.0	100.0
click alarmclock	92.0	100.0
place fan	95.0	97.0
click bell	100.0	100.0
place mouse pad	99.0	100.0
dump bin bigbin	84.0	100.0
place object basket	74.0	92.0
grab roller	95.0	91.0
place object scale	78.0	100.0
handover block	83.0	92.0
place object stand	97.0	100.0
handover mic	87.0	85.0
place phone stand	66.0	88.0
hanging mug	63.0	58.0
place shoe	84.0	100.0
lift pot	27.0	73.0
press stapler	98.0	100.0
move can pot	93.0	100.0
put bottles dustbin	71.0	100.0
move playingcard away	99.0	97.0
put object cabinet	14.0	95.0
move stapler pad	92.0	95.0
rotate qrcode	75.0	92.0
open laptop	82.0	100.0
scan object	4.0	85.0
open microwave	96.0	69.0
shake bottle horizontally	90.0	95.0
pick diverse bottles	51.0	100.0
shake bottle	89.0	100.0
pick dual bottles	92.0	100.0
stack blocks three	94.0	73.0
place a2b left	80.0	100.0
stack blocks two	98.0	92.0
place a2b right	81.0	95.0
stack bowls three	43.0	98.0
place bread basket	89.0	100.0
stack bowls two	78.0	95.0
place bread skillet	34.0	100.0
stamp seal	56.0	98.0
place burger fries	97.0	97.0
turn switch	74.0	100.0

Continued on next page

Table 6: Per-task results (continued)

Task	RoboTwin 2.0	LADDER
place can basket	70.0	97.0
move pillbottle pad	67.0	86.0
Average success rate	78.8	94.2

C.3 Collection time

Figures 7 and 8 report the time to collect 50 successful demonstrations.

D Demonstration collection on articulated tasks

D.1 Evaluation protocol

Unless stated otherwise, collection success is the fraction of attempted executions that complete the task and are retained as valid demonstrations; it is not the success rate of a learned VLA. For the RoboGen comparison, RoboGen rollout success is assessed manually by human evaluators. We assess LADDER rollouts against the same task-specific descriptions of successful completion used by RoboGen. LADDER rollouts that fail inverse kinematics, drop an object, or do not meet the corresponding task criterion are discarded.

The two methods also use different end effectors for articulated-object manipulation. RoboGen employs a suction gripper that can attach to an object or one of its links upon contact, whereas LADDER uses a gripper to grasp the relevant part, such as a door handle. Suction attachment can simplify interactions that require grasping in LADDER; this difference should be considered when interpreting the success rates. LADDER nevertheless achieves a higher average collection success rate on the evaluated tasks.

D.2 Per-task results

Table 7: Per-task demonstration-collection success rate (%) on 50 RoboGen articulated tasks.

Task	RoboGen	LADDER
rotate laptop screen	100.0	100.0
open toilet lid	100.0	100.0
extend suitcase handle	100.0	100.0
close trashcan lid	33.0	50.0
open window halfway	100.0	100.0
change lamp direction	100.0	100.0
flush the toilet	67.0	100.0
partially close window	50.0	100.0
move clock ahead for daylight saving	38.0	100.0
close dispenser lid (after use)	100.0	100.0
close the oven door	83.0	100.0
open drawer	75.0	100.0
open trashcan lid	100.0	88.0
close table drawer	75.0	100.0
extend display screen	100.0	75.0
open trash can	100.0	38.0
turn on faucet	83.0	100.0

Continued on next page

Table 7: Per-task results (continued)

Task	RoboGen	LADDER
remove pot lid	100.0	100.0
unfold chair	50.0	50.0
close toilet lid	100.0	100.0
open stapler lid	50.0	88.0
open washing machine door	100.0	100.0
turn on the printer	100.0	100.0
move door slightly open	67.0	100.0
lift kettle by handle	83.0	100.0
open door	50.0	100.0
close the drawer of the table	75.0	100.0
turn off faucet	67.0	100.0
close door	50.0	100.0
close window	80.0	100.0
open partial box lid	83.0	75.0
open box	80.0	50.0
pull drawer out	67.0	100.0
rotate clock hands	40.0	50.0
turning on coffee machine	50.0	100.0
pull lever to start coffee brewing	100.0	75.0
press start button	100.0	100.0
open dishwasher door	60.0	100.0
close dispenser lid	25.0	100.0
tilt display screen	60.0	38.0
open dispenser lid	0.0	100.0
unfold the chair	100.0	75.0
rotate fan rotor	100.0	100.0
rotate globe horizontally	100.0	50.0
turn on water faucet	100.0	100.0
turn on lamp	25.0	100.0
open laptop	80.0	100.0
open kettle lid	75.0	75.0
opening both refrigerator doors	80.0	100.0
open microwave door	25.0	100.0
Average success rate	74.5	89.5

E Demonstration quality for policy learning

E.1 Policy training and evaluation

We follow the RoboTwin 2.0 benchmark protocol, using 50 clean demonstrations per task across 50 tasks with the Aloha-AgileX dual-arm embodiment. In our experiments, we replace the RoboTwin demonstrations with LADDER demonstrations while keeping each model's architecture and training configuration the same across the two data sources.

We fully fine-tune the pretrained $\pi_{0.5}$ model using the OpenPI configuration `pi05_base_aloha_full_sim_arx-x5_seed_0`. X-VLA is fully fine-tuned from its pre-trained checkpoint for 100,000 steps with a batch size of 32; its backbone is frozen for the first 1,000 steps. Both models are jointly trained on all 50 tasks. DP3 is trained independently for each task for 3,000 epochs, with a batch size of 256, a planning horizon of 8, and 1,024-point observations with background and tabletop segmentation.

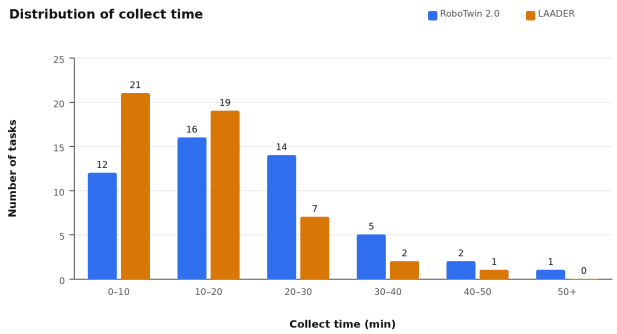


Figure 7: Distribution of the time required to collect 50 demonstrations per task.

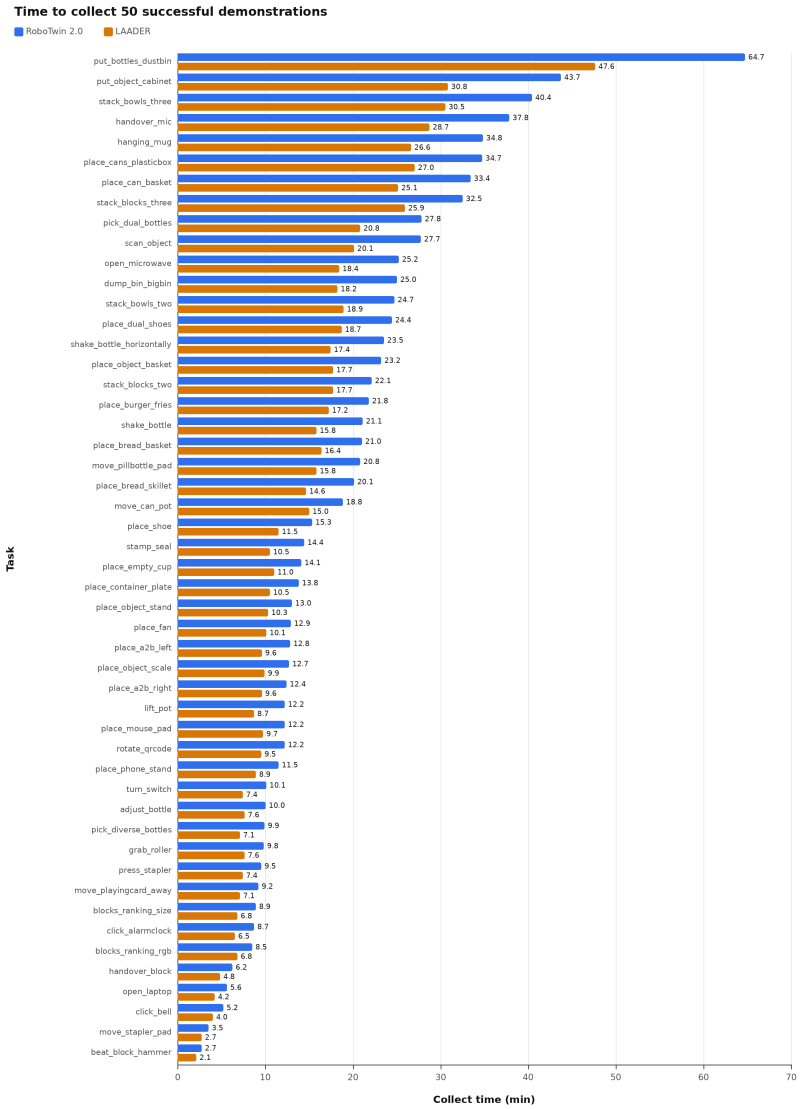


Figure 8: Per-task collection time on 50 RoboTwin 2.0 tasks.

We trained $\pi_{0.5}$ and X-VLA on two NVIDIA H100 GPUs each, and DP3 on four NVIDIA GeForce RTX 4090 GPUs. Policies trained on LADDER demonstrations are evaluated for 100 trials per task under both the clean and domain-randomized settings, using the RoboTwin 2.0 task success checkers. The RT entries in Table 8 are taken from the official RoboTwin 2.0 leaderboard. Accordingly, the X-VLA hard score for `handover block` is reported as 1.3%, exactly as listed by the official leaderboard.

E.2 PER-TASK RESULTS

Table 8: Per-task success rate (%) on RoboTwin 2.0 for $\pi_{0.5}$, DP3 and X-VLA. RT is the same architecture fine-tuned on RoboTwin 2.0 expert demonstrations; LADDER is fine-tuned on LADDER demonstrations. Higher number in each RT/LADDER pair is shaded. Tasks 1–50.

	$\pi_{0.5}$				DP3				X-VLA			
	Clean		Hard		Clean		Hard		Clean		Hard	
Task	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER
adjust bottle	99.0	99.0	77.0	74.0	99.0	100.0	3.0	12.0	99.0	100.0	33.0	55.0
beat block hammer	93.0	96.0	23.0	24.0	72.0	72.0	8.0	9.0	87.0	87.0	9.0	12.0
blocks ranking rgb	72.0	35.0	44.0	28.0	3.0	28.0	0.0	8.0	3.0	19.0	4.0	24.0
blocks ranking size	45.0	30.0	21.0	23.0	2.0	0.0	0.0	0.0	56.0	52.0	25.0	20.0
click alarmclock	65.0	96.0	50.0	74.0	77.0	100.0	14.0	19.0	62.0	77.0	43.0	56.0
click bell	31.0	96.0	35.0	84.0	90.0	90.0	0.0	0.0	100.0	100.0	63.0	58.0
dump bin bigbin	92.0	85.0	84.0	77.0	85.0	91.0	53.0	58.0	83.0	87.0	41.0	54.0
grab roller	99.0	100.0	97.0	99.0	98.0	98.0	2.0	4.0	100.0	100.0	20.0	25.0
handover block	30.0	42.0	12.0	24.0	70.0	100.0	0.0	1.0	77.0	96.0	1.3	4.0
handover mic	98.0	98.0	6.0	42.0	100.0	100.0	3.0	11.0	98.0	100.0	1.0	21.0
hanging mug	17.0	32.0	14.0	20.0	17.0	28.0	1.0	7.0	19.0	26.0	2.0	16.0
lift pot	96.0	83.0	35.0	56.0	97.0	100.0	0.0	6.0	96.0	100.0	15.0	29.0
move can pot	60.0	81.0	10.0	34.0	70.0	70.0	6.0	7.0	58.0	58.0	0.0	3.0
move pillbottle pad	46.0	42.0	32.0	28.0	41.0	49.0	0.0	6.0	66.0	71.0	3.0	18.0
move playingcard away	84.0	80.0	65.0	26.0	68.0	93.0	3.0	3.0	84.0	100.0	5.0	4.0
move stapler pad	20.0	74.0	18.0	62.0	12.0	40.0	0.0	1.0	42.0	60.0	0.0	3.0
open laptop	93.0	92.0	68.0	78.0	82.0	80.0	7.0	14.0	85.0	84.0	16.0	33.0
open microwave	86.0	95.0	59.0	74.0	61.0	80.0	22.0	29.0	43.0	55.0	5.0	21.0
pick diverse bottles	66.0	42.0	42.0	24.0	52.0	68.0	1.0	0.0	50.0	60.0	30.0	27.0
pick dual bottles	72.0	86.0	57.0	55.0	60.0	69.0	1.0	5.0	41.0	47.0	51.0	61.0
place a2b left	71.0	78.0	49.0	57.0	46.0	44.0	2.0	7.0	53.0	52.0	15.0	26.0
place a2b right	68.0	79.0	39.0	52.0	49.0	43.0	0.0	2.0	49.0	45.0	13.0	18.0
place bread basket	63.0	72.0	60.0	32.0	26.0	28.0	1.0	0.0	85.0	86.0	52.0	49.0
place bread skillet	76.0	81.0	47.0	64.0	19.0	24.0	0.0	0.0	90.0	93.0	17.0	14.0
place burger fries	82.0	86.0	86.0	80.0	72.0	64.0	18.0	18.0	90.0	85.0	35.0	34.0
place can basket	56.0	70.0	10.0	15.0	67.0	75.0	2.0	2.0	70.0	75.0	9.0	9.0
place cans plasticbox	31.0	42.0	73.0	40.0	48.0	59.0	3.0	11.0	93.0	100.0	60.0	79.0
place container plate	92.0	100.0	72.0	76.0	86.0	86.0	1.0	0.0	94.0	94.0	39.0	34.0
place dual shoes	70.0	79.0	45.0	65.0	13.0	40.0	0.0	1.0	30.0	47.0	12.0	15.0
place empty cup	96.0	98.0	80.0	78.0	65.0	81.0	1.0	9.0	90.0	100.0	16.0	36.0
place fan	66.0	79.0	26.0	63.0	36.0	37.0	1.0	1.0	63.0	64.0	1.0	0.0
place mouse pad	34.0	44.0	31.0	32.0	4.0	5.0	1.0	7.0	53.0	54.0	0.0	15.0
place object basket	70.0	76.0	29.0	40.0	65.0	87.0	0.0	8.0	81.0	95.0	6.0	25.0
place object scale	78.0	64.0	38.0	42.0	15.0	37.0	0.0	5.0	76.0	90.0	6.0	19.0
place object stand	85.0	92.0	47.0	66.0	60.0	74.0	0.0	1.0	91.0	100.0	43.0	45.0
place phone stand	65.0	78.0	37.0	51.0	44.0	60.0	2.0	2.0	84.0	94.0	5.0	4.0
place shoe	82.0	94.0	50.0	72.0	58.0	83.0	2.0	7.0	83.0	99.0	39.0	52.0
press stapler	79.0	96.0	63.0	78.0	69.0	70.0	3.0	9.0	87.0	88.0	73.0	87.0
put bottles dustbin	78.0	88.0	50.0	61.0	60.0	79.0	21.0	28.0	47.0	59.0	20.0	37.0
put object cabinet	39.0	64.0	22.0	52.0	72.0	75.0	1.0	4.0	41.0	43.0	21.0	28.0
rotate qrcode	90.0	94.0	21.0	62.0	74.0	99.0	1.0	7.0	81.0	97.0	1.0	15.0
scan object	49.0	64.0	40.0	52.0	31.0	50.0	1.0	2.0	57.0	69.0	9.0	10.0
shake bottle	100.0	100.0	100.0	98.0	98.0	98.0	19.0	24.0	100.0	100.0	76.0	89.0
shake bottle horizontally	99.0	100.0	100.0	97.0	100.0	100.0	25.0	26.0	100.0	100.0	76.0	77.0
stack blocks three	77.0	56.0	25.0	18.0	1.0	4.0	0.0	6.0	0.0	2.0	0.0	15.0
stack blocks two	91.0	74.0	43.0	44.0	24.0	46.0	0.0	5.0	10.0	24.0	2.0	13.0
stack bowls three	81.0	86.0	52.0	70.0	57.0	58.0	5.0	8.0	77.0	78.0	9.0	17.0

Continued on next page

Table 8: Per-task results (continued)

	$\pi_{0.5}$				DP3				X-VLA			
	Clean		Hard		Clean		Hard		Clean		Hard	
Task	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER	RT	LADDER
stack bowls two	94.0	96.0	70.0	78.0	83.0	80.0	6.0	10.0	85.0	83.0	15.0	25.0
stamp seal	53.0	72.0	21.0	43.0	18.0	23.0	0.0	0.0	50.0	53.0	2.0	0.0
turn switch	52.0	68.0	26.0	38.0	46.0	52.0	8.0	13.0	40.0	44.0	6.0	17.0
Average	70.6	77.1	46.0	54.4	55.24	64.34	4.96	8.46	68.0	73.8	20.9	29.0

Table 8 expands the $\pi_{0.5}$, DP3 and X-VLA averages in Table 3.

F Evaluation of the manipulation knowledge base

F.1 Experimental setup

Sec. 4.3 compares two skill-learning runs on the same generated articulated tasks (Wang et al., 2023). Both runs use GPT-5.6 to write the reward, SAC and 1M environment steps. The baseline prompt contains the task description, the initial scene yaml, the URDF articulation tree, PartNet-Mobility semantics, the task-relevant joints and links, and RoboGen's helper APIs (`get_joint_state`, `get_link_state`, `get_eef_pos`, joint limits, and bounding boxes).

In the other run the method name is still RoboGen: the prompt, the APIs, and the RL trainer are unchanged. We only append three fields from the Manipulation Knowledge Base, aligned to the same URDF instance: (i) the operated part (the link or joint the skill should act on, e.g., the door handle rather than the timer knob); (ii) a grasp or contact pose $T^\star$ of the end-effector on that part, obtained by mapping the parameterized affordance onto the live object pose; (iii) the motion type of the skill (revolute, prismatic, or lift). GPT-5.6 is instructed to use these fields when writing the reward, for example by attracting the end-effector to $T^\star$ and by moving the named joint in the direction implied by the motion type. We do not insert a hand-designed dense reward in place of RoboGen's generator, and we do not give the trainer the waypoint sequence used later for demonstration collection. Per-task scores are in Table 9.

F.2 Per-task results

Table 9 expands the results in Table 4. The RoboGen and RoboGen+MKB columns report success rates using GPT-5.6-generated rewards and SAC. RoboGen+MKB additionally receives the operated part, a contact pose, and the motion type from the Manipulation Knowledge Base.

Table 9: Per-task success rates (%) on 50 RoboGen articulated tasks. LADDER reports demonstration collection success. MKB denotes the Manipulation Knowledge Base.

Task	LADDER	RoboGen+MKB	RoboGen (GPT-5.6)
Rotate laptop screen	100.0	100.0	100.0
Extend suitcase handle	100.0	100.0	100.0
Open window halfway	100.0	100.0	100.0
Flush the toilet	100.0	63.0	75.0
Move clock ahead for daylight saving	100.0	75.0	38.0
Close the oven door	100.0	88.0	88.0
Open trashcan lid	88.0	75.0	100.0
Extend display screen	75.0	63.0	100.0
Turn on faucet	100.0	75.0	88.0
Unfold chair	50.0	50.0	50.0
Open stapler lid	88.0	88.0	50.0
Turn on the printer	100.0	63.0	100.0

Continued on next page

Table 9: Per-task success rates (continued)

Task	LADDER	RoboGen+MKB	RoboGen (GPT-5.6)
Lift kettle by handle	100.0	100.0	88.0
Close the drawer of the table	100.0	100.0	75.0
Close door	100.0	100.0	50.0
Open partial box lid	75.0	63.0	88.0
Pull drawer out	100.0	100.0	75.0
Turning on coffee machine	100.0	100.0	50.0
Press start button	100.0	75.0	100.0
Close dispenser lid	100.0	100.0	25.0
Open dispenser lid	100.0	75.0	0.0
Rotate fan rotor	100.0	100.0	100.0
Turn on water faucet	100.0	100.0	100.0
Open laptop	100.0	100.0	88.0
Opening both refrigerator doors	100.0	100.0	100.0
Open toilet lid	100.0	100.0	100.0
Close trashcan lid	50.0	50.0	38.0
Change lamp direction	100.0	100.0	100.0
Partially close window	100.0	100.0	50.0
Close dispenser lid (after use)	100.0	100.0	100.0
Open drawer	100.0	100.0	75.0
Close table drawer	100.0	88.0	75.0
Open trash can	38.0	25.0	100.0
Remove pot lid	100.0	75.0	100.0
Close toilet lid	100.0	100.0	100.0
Open washing machine door	100.0	63.0	100.0
Move door slightly open	100.0	100.0	75.0
Open door	100.0	100.0	50.0
Turn off faucet	100.0	63.0	63.0
Close window	100.0	63.0	88.0
Open box	50.0	38.0	88.0
Rotate clock hands	50.0	50.0	38.0
Pull lever to start coffee brewing	75.0	75.0	100.0
Open dishwasher door	100.0	100.0	63.0
Tilt display screen	38.0	38.0	63.0
Unfold the chair	75.0	63.0	100.0
Rotate globe horizontally	50.0	50.0	100.0
Open kettle lid	75.0	50.0	75.0
Open microwave door	100.0	100.0	25.0
Turn on lamp	100.0	100.0	25.0
Average success rate	89.5	80.9	76.4

G Evaluation of the multi-agent workflow

G.1 Experimental setup

The multi-agent ablation uses GPT-5.6 (OpenAI, 2026) on 25 RoboTwin 2.0 tasks with 10 instructions per task. Both sides receive the same packed knowledge from manipulation knowledge base. The FDR setting runs the asset agent and the planning agent in parallel and then matches names to categories. The ablated setting asks a single agent to emit the configuration without that split. Human experts check whether the plan is a reasonable decomposition. For each generated decomposition, all 17 experts independently refer to RoboGen's task-specific description and inspect the actual task completion in simulation. A task is counted as successful only if all 17 experts judge that it is completed; otherwise, it is counted as unsuccessful. We report planning success rate (ASR) and

average tokens. Token counts include the tokens generated during inter-agent communication and in the final output, averaged per instruction. Per-task numbers are in Table 10.

G.2 PER-TASK RESULTS

Table 10: Per-task planning success rate (%) and average tokens on 25 RoboTwin 2.0 tasks. Both settings use GPT-5.6 and the same packed knowledge base. FDR is the fine-grained decomposition and reassembly method in Sec. 3.2.

	multi-agent workflow		LLM	
Task	SR	Token	SR	Token
beat_block_hammer	100.0	1378	80.0	1774
blocks_ranking_rgb	100.0	1441	40.0	1946
blocks_ranking_size	100.0	1443	60.0	1891
dump_bin_bigbin	80.0	1389	50.0	1896
grab_roller	100.0	1343	100.0	1457
handover_block	90.0	1346	70.0	1798
handover_mic	100.0	1345	60.0	1797
hanging_mug	100.0	1371	90.0	1653
lift_pot	100.0	1371	90.0	1597
open_laptop	100.0	1330	100.0	1276
open_microwave	100.0	1335	100.0	1336
place_bread_basket	100.0	1386	80.0	1780
place_bread_skillet	90.0	1424	40.0	1930
place_container_plate	100.0	1381	100.0	1550
place_dual_shoes	100.0	1459	100.0	1569
place_object_scale	100.0	1393	100.0	1392
place_phone_stand	100.0	1383	80.0	1665
put_bottles_dustbin	100.0	1531	50.0	2032
put_object_cabinet	90.0	1403	70.0	1853
rotate_qrcode	100.0	1367	90.0	1536
scan_object	100.0	1439	90.0	1718
shake_bottle_horizontally	100.0	1343	100.0	1287
stack_bowls_three	100.0	1425	90.0	1592
stamp_seal	100.0	1353	100.0	1354
turn_switch	100.0	1331	100.0	1276
Average	98.0	1388.4	81.2	1638.2

Table 10 expands the overall ASR and token numbers in Table 5.