JARVIS: FROM VISUAL MANUALS TO AUTONOMOUS ROBOTIC ASSEMBLY

Anonymous authors
Paper under double-blind review

ABSTRACT

Visual manuals communicate procedural knowledge through stepwise diagrams, but translating these instructions into reliable long-horizon robotic execution remains challenging. We introduce JARVIS, a method-agnostic benchmark for autonomous visual-manual-guided execution without additional language instructions. JARVIS comprises 236,679 LEGO assembly tasks across 8,987 scenes, with a carefully curated JARVIS-v1 suite of 50 tasks for interactive robotic evaluation. Tasks span varying initial structural complexities and assembly horizons, requiring agents to interpret manuals, ground actions in an evolving workspace, and execute the intended procedure. A structure-aware evaluation protocol captures geometric accuracy, structural relations, construction order, partial progress, and completion. We further introduce JARVIS-agentic, a tool-grounded embodied agent that composes robot primitives into executable programs and uses execution feedback for verification and recovery, without task-specific action trajectories or model parameter updates. Across nine multimodal and reasoning models, task-oriented primitives increase the average assembly score. Performance nevertheless declines sharply with assembly horizon, and execution errors remain difficult to correct. Prior-task experience improves initial performance, while repeated reflection provides inconsistent benefits, highlighting persistent limitations in connecting visual procedural reasoning with reliable physical execution.

1 INTRODUCTION

Across many everyday activities, people learn how to build, assemble, and operate things by following visual manuals. These manuals organize tasks into standardized step-by-step diagrams, providing a compact and transferable representation of procedural knowledge that conveys both the desired outcome and the process required to achieve it (Tie et al., 2025b). Following such manuals requires humans to interpret the visual instructions, align each illustrated step with the current physical state, and translate the inferred procedure into concrete actions. This ability goes beyond visual recognition alone, requiring visual reasoning to be tightly coupled with sequential decision-making and physical execution (Wang et al., 2022). As multimodal models increasingly move beyond passive perception toward embodied interaction (Ke et al., 2026; Hong et al., 2026), a fundamental question is whether machines can interpret procedural knowledge from visual manuals and autonomously translate it into action.

Recent studies have taken initial steps toward this capability, particularly in manual-guided robotic assembly (Tie et al., 2025b; Zhang et al., 2025; Gu et al., 2026a). Existing approaches have primarily developed task-specific solutions for extracting assembly structures, predicting part poses, and generating assembly plans (Tie et al., 2025b; Pun et al., 2025b; Wang et al., 2022; Yan & Mu, 2025b; Li et al., 2026). Many therefore stop before physical execution; when execution is included, it is often realised through predefined or manually engineered control pipelines. As a result, they do not directly evaluate whether a model can autonomously close the loop from visual manual understanding to physical construction. Benchmarking this capability further requires end-to-end execution tasks and metrics that capture partial progress and structural correctness beyond binary task success (Heo et al., 2025; Wang et al., 2025). Consequently, it remains unclear how well current multimodal models can use a visual manual as the task specification and autonomously carry out a complete procedure over multiple steps.

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.

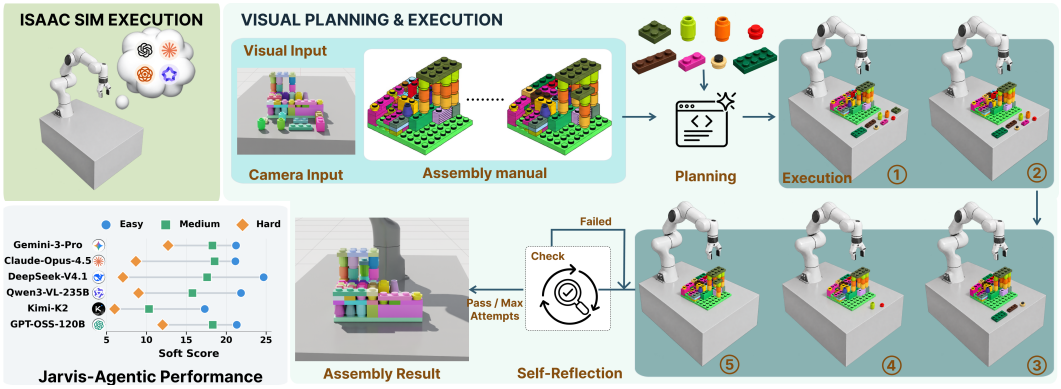


Figure 1: Overview of the visual planning, execution, and reflection workflow in JARVIS.

To systematically evaluate this capability, we introduce JARVIS, a method-agnostic benchmark for autonomous visual-manual-guided execution. We construct a large-scale task collection comprising 236,679 assembly tasks across 8,987 LEGO scenes, spanning diverse initial structures and assembly horizons. From this collection, we curate JARVIS-v1, a suite of 50 tasks across three difficulty levels for interactive robotic evaluation. As shown in Fig. 1, an agent receives a visual assembly manual and workspace observations, infers the intended procedure, and executes it while adapting to the evolving physical state. Each manipulation affects subsequent steps, requiring coherent planning and execution throughout assembly. We further introduce a structure-aware evaluation protocol that measures relative geometry, structural relations, and completion, capturing partial progress and accumulated errors beyond binary task success.

Turning manual understanding into physical execution poses a separate practical challenge. Learned control policies typically require task-specific robot action supervision, as low-level trajectories are tied to the robot's kinematics and control interface. Collecting dedicated demonstrations or trajectories for each new construction task is therefore costly. We introduce JARVIS-agentic, a tool-grounded embodied agent that bridges manual-derived reasoning and physical execution without task-specific action trajectories. Equipped with a compact library of verified robot primitives, the agent grounds the manual, composes executable behaviours, and uses execution reflection for verification, recovery, and replanning.

Using JARVIS, we evaluate 9 multimodal and reasoning models on manual-guided assembly tasks spanning different structural complexities and assembly horizons. Task-oriented robot skills raise the average assembly score from 7.88 to 14.71 without task-specific action trajectories, demonstrating the value of tool-grounded execution. Nevertheless, performance declines sharply with assembly horizon, and improvements in intermediate grounding do not reliably translate into successful assembly. Prior-task experience improves first-round performance, whereas repeated reflection and recovery from execution errors remain unreliable. These findings expose the difficulty of connecting procedural reasoning, physical execution, and reflection-based correction in long-horizon assembly.

Our main contributions are summarized as follows:

- We formulate **autonomous visual-manual-guided execution** for long-horizon robotic assembly and introduce JARVIS, comprising 236,679 tasks across 8,987 scenes, with a curated JARVIS-v1 suite for interactive evaluation. A structure-aware protocol measures geometric and relational correctness, partial progress, and completion.
- We introduce **JARVIS-agentic**, a tool-grounded embodied agent that translates visual assembly instructions into executable programs through reusable robot primitives. The framework integrates planning, execution verification, and recovery without task-specific action trajectories or model parameter updates.
- We evaluate **nine multimodal and reasoning models**, revealing persistent difficulties in translating manual understanding into reliable long-horizon execution. Our analysis shows that task-oriented tools and prior-task experience improve performance, while online reflection provides inconsistent benefits and error recovery remains limited.

2 RELATED WORK

Visual assembly planning. Visual assembly planning focuses on recovering structural and procedural representations from images. MEPNet predicts stepwise component poses from LEGO manuals (Wang et al., 2022), while Manual-PA estimates assembly order and part poses from instruction diagrams (Zhang et al., 2025). Neural Assembler infers component graphs and assembly sequences from multiple views of a completed structure (Yan & Mu, 2025a), and AssemLM predicts assembly poses from manuals, point clouds, and language (Jing et al., 2026). Related generative methods produce buildable structures from text (Pun et al., 2025a) or represent brick sequences through connectivity (Kulits & Schmid, 2026). These tasks assess whether a model can infer a valid assembly representation. Autonomous execution additionally requires grounding that representation in a changing workspace and maintaining its validity through physical interaction.

Robotic execution from assembly instructions. Several systems connect assembly representations to robot control. Manual2Skill uses assembly graphs, learned pose estimation, and motion planning for furniture assembly (Tie et al., 2025b), and Manual2Skill++ incorporates explicit connector constraints (Tie et al., 2025a). BrickCraft derives assembly sequences and situated manuals from digital designs to guide learned visuomotor skills (Yu et al., 2026). ManualVLA generates intermediate manuals from goal states to condition action execution (Gu et al., 2026b). These systems demonstrate assembly under their respective representations and control designs. A distinct evaluation question is how reliably general-purpose multimodal agents can interpret a supplied visual manual, determine executable actions, and respond to deviations during construction. JARVIS formulates this capability as a method-agnostic benchmark.

Execution benchmarks and feedback. AssemblyBench evaluates assembly order and physically plausible part trajectories (Li et al., 2026), while WorkBenchMark evaluates robotic construction of target LEGO Duplo structures (Ma et al., 2026). CaP-X studies coding agents under different tool abstractions and feedback settings (Fu et al., 2026), and ENPIRE investigates policy improvement through repeated physical trials (Xiao et al., 2026). The central question in JARVIS is whether visual procedural understanding survives execution: can an agent preserve the intended structure as actions accumulate, and recover when the observed state departs from the manual? Our structure-aware evaluation measures partial assembly quality, while controlled experiments examine the effects of horizon, initial complexity, and execution feedback.

3 JARVIS BENCHMARK

JARVIS evaluates autonomous visual-manual-guided execution across the full planning–execution pipeline. We instantiate the benchmark with long-horizon LEGO construction in an interactive assembly environment, using visual manuals and structured target states to define each task. This section presents the task formulation, benchmark construction, environment, task composition, and evaluation protocol.

Task Formulation. Let $\mathcal{M} = \{I_0, \ldots, I_T\}$ denote a visual assembly manual, where each image I_t depicts the target construction state s_t at step t. The ordered image sequence specifies the desired progression from the initial state s_0 to the completed structure s_T. At execution step t, the agent receives an observation o_t of the current environment and produces an action a_t, inducing the state transition $s_{t+1} = \mathcal{T}(s_t, a_t)$, where $\mathcal{T}$ denotes the environment dynamics. The objective is to control the robotic manipulator to realize the final assembly s_T, using only the visual manual $\mathcal{M}$ and observations.

Task Composition and Difficulty. JARVIS is constructed from $8,987$ unique LEGO assembly scenes paired with visual manuals and structured 3D models. Rather than treating each complete construction as a single task, we decompose each assembly trajectory into consecutive segments of varying lengths. Each segment defines an independent execution task, consisting of an initial partially assembled state, the corresponding visual manual segment, the required LEGO components, and a target construction state.

Table 1: Statistics of the JARVIS-v1 benchmark.

Split	#Tasks	Assembly Horizon		Initial Complexity N_{init}		Total Bricks	
		avg.	range	avg.	range	avg.	range
Easy	20	2.45	1-4	33.05	2-105	35.50	4-107
Medium	15	4.13	2-7	78.80	2-213	82.93	7-215
Hard	15	9.87	3-29	127.00	57-236	136.87	75-239

This decomposition naturally produces tasks with different execution horizons and scene complexities. In total, JARVIS contains $236,679$ assembly tasks, each requiring the manipulation of 1-43 LEGO bricks. We further render the same manual instructions in multiple visual styles to evaluate robustness to variations in manual appearance. Details of the data collection, manual-model alignment, and task generation pipeline are provided in the supplementary material.

To characterize task difficulty, we consider two complementary factors: the complexity of the initial partial assembly and the required assembly workload. For task i, we define

$$S_i^{\text{scene}} = \frac{N_i^{\text{scene}} - N_{\min}^{\text{scene}}}{N_{\max}^{\text{scene}} - N_{\min}^{\text{scene}}}, \qquad S_i^{\text{act}} = \frac{N_i^{\text{act}} - N_{\min}^{\text{act}}}{N_{\max}^{\text{act}} - N_{\min}^{\text{act}}}, \tag{1}$$

where N_i^{scene} is the number of bricks already present in the pre-assembled structure at the beginning of task i, and N_i^{act} is the number of additional bricks that the agent must manipulate and assemble to complete the target structure. The minimum and maximum values are computed over all tasks in the evaluation benchmark, such that both normalized measures lie in $[0, 1]$.

The overall task difficulty is defined as

$$D_i = \alpha S_i^{\text{scene}} + (1 - \alpha) S_i^{\text{act}}, \tag{2}$$

where we set $\alpha = 0.3$, assigning a larger weight to the execution horizon because manipulation errors can accumulate over successive assembly steps. Here, S_i^{scene} measures the normalized total number of bricks present in the task scene, whereas S_i^{act} measures the normalized number of bricks that must be manipulated and assembled.

From the full task pool, we first curate a representative candidate set of $1,067$ validation tasks, selected to cover diverse source assemblies, construction scales, and execution horizons. We then rank these candidates by D_i and partitioned into three difficulty strata: the bottom 40% constitute the Easy stratum, the next 30% the Medium stratum, and the top 30% the Hard stratum. Within each stratum, we sample tasks at approximately uniform intervals along the difficulty distribution while promoting diversity among the source LEGO assemblies. The resulting JARVIS-v1 comprises 50 interactive tasks: 20 Easy, 15 Medium, and 15 Hard as shown in Table1.

Interactive Assembly Environment. Each task is instantiated as an interactive robotic assembly episode in NVIDIA Isaac Sim 6.0.1 (Liang et al., 2018), using a Franka Emika Panda manipulator (Gaz et al., 2019) with a parallel-jaw gripper. Each scene contains a partially assembled LEGO structure and the movable bricks required by the corresponding manual segment. The agent receives top-down RGB observations and robot-state feedback, and interacts with the arm and gripper through ROS 2 (Macenski et al., 2022). Robot motion is executed with NVIDIA cuMotion/RMPflow, with MoveIt (Chitta, 2016) supported as an alternative backend.

During each episode, the evaluated method receives the visual manual and current scene observations and must autonomously complete the assembly until termination or the execution budget is exhausted. Ground-truth structure, object poses, and target placements are reserved for evaluation and are not exposed to the method, except for explicitly defined oracle baselines. Further implementation details are provided in the supplementary material.

Structure-Aware Assembly Evaluation. A binary success criterion cannot fully capture assembly quality. We therefore adopt a structure-aware evaluation that separately measures geometric accuracy, task completion, and final-state validity. Specifically, the pose score S_{pose} evaluates each manipulated brick with respect to its intended support, accounting for relative position, vertical alignment, symmetry-aware orientation, and a low-weight absolute-position anchor. Completion

S^{comp} captures how much of the required assembly has been achieved. These components are combined into an overall soft score, with a bounded time penalty applied only when execution exceeds a task-dependent budget.

Pose accuracy. We evaluate each required brick against its target pose in the simulator world frame. For brick j, we compute its Euclidean position error e_j^{pos}, vertical error e_j^z, and angular orientation error e_j^θ. Each error is converted into a soft score using a Gaussian kernel,

$$s_j^k = \exp\left[-\frac{1}{2}\left(\frac{e_j^k}{\sigma_k}\right)^2\right], \qquad k \in \{\text{pos}, z, \theta\}, \tag{3}$$

where σ_k controls the tolerance of each term. Missing bricks and bricks that have not been moved from their initial poses receive zero credit. We average the component scores over the N_i required bricks in task i:

$$S_i^{\text{pose}} = \frac{1}{N_i}\sum_{j=1}^{N_i}\left(\lambda_{\text{pos}} s_j^{\text{pos}} + \lambda_z s_j^z + \lambda_\theta s_j^\theta\right), \tag{4}$$

where $\lambda_{\text{pos}}+\lambda_z+\lambda_\theta = 1$. This formulation provides partial credit for near-correct placements while smoothly penalising larger errors. The kernel bandwidths and component weights are reported in the supplementary material.

Completion. The completion score measures the fraction of required bricks successfully placed near their target positions. A brick is counted as completed if it has been moved by more than $5\,\text{mm}$ from its initial pose, its Euclidean distance to the target position does not exceed $5\,\text{cm}$, and it is not held by the gripper when the episode terminates. Thus,

$$S_i^{\text{comp}} = \frac{1}{N_i}\sum_{j=1}^{N_i}\mathbb{I}\left[d_j^{\text{move}} > 5\,\text{mm} \wedge e_j^{\text{pos}} \le 5\,\text{cm} \wedge \neg h_j\right], \tag{5}$$

where N_i is the number of required bricks in task i, d_j^{move} is the displacement from the initial pose, and h_j indicates whether the brick remains held by the gripper.

Final-state validity. We consider whether each manipulated brick remains in a valid state at the end of the episode. A brick receives validity credit if it has been moved beyond the movement tolerance, its horizontal bounding-box projection remains within the designated tabletop region, and it is no longer held by the gripper.

Overall score. Before accounting for execution time, the assembly-quality score for task i is

$$S_i^{\text{quality}} = \lambda_{pose} S_i^{\text{pose}} + \lambda_{\text{comp}} S_i^{\text{comp}} \tag{6}$$

where $\lambda_{\text{pose}} + \lambda_{\text{comp}} = 1$. The score is set to zero if no required brick is moved beyond the movement tolerance. We define a task-dependent time budget as

$$T_i = T_{\text{base}} + N_i T_{\text{brick}}, \tag{7}$$

where T_{base} is the base time allowance, and T_{brick} is the additional allowance per brick. Let t_i denote the agent's actual execution time for task i. The final episode score combines assembly quality with a bounded time penalty:

$$S_i = 100 S_i^{\text{quality}}\left[\alpha + (1-\alpha)\exp\left(-\frac{\max(0, t_i - T_i)}{T_i}\right)\right], \tag{8}$$

where the time-dependent factor mildly penalizes slower execution while remaining bounded. The corresponding hyperparameters are reported in the supplementary material.

4 JARVIS-AGENTIC

We introduce JARVIS-agentic, a tool-grounded vision-language-action framework for long-horizon LEGO assembly as shown in Figure 2. Given a visual assembly manual and the current workspace observation, the agent assesses assembly progress, identifies the next step, and generates executable

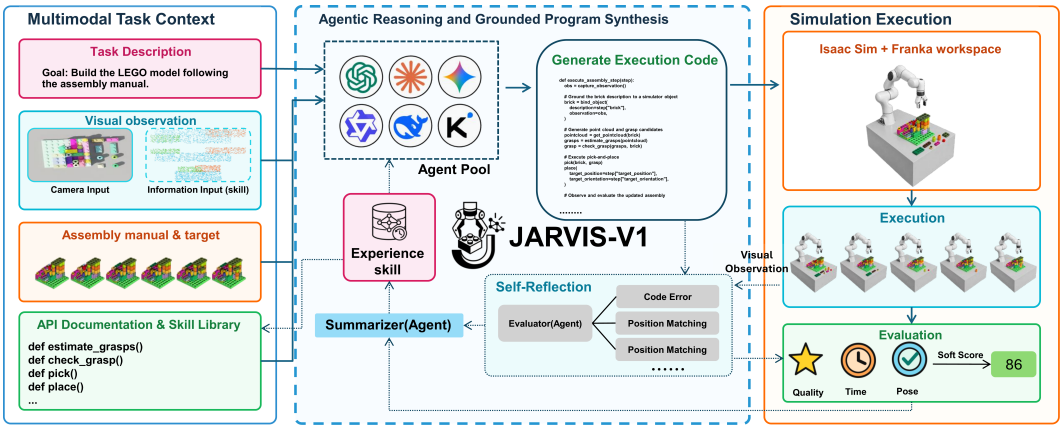


Figure 2: Overview of JARVIS-V1. The framework integrates multimodal task context, agentic code generation, simulation execution, evaluation, and self-reflection in a closed-loop assembly process.

code using constrained robot APIs and reusable manipulation skills. After execution, it observes the resulting scene and operation feedback, allowing it to verify progress, recover from failures, and replan over multiple interaction rounds.

The framework integrates planning and execution while decoupling the agent's decisions from low-level robot control. Adapting VLA policies to new robot setups and manipulation tasks often requires demonstrations and fine-tuning (Kim et al., 2024), making trajectory collection costly for manual-guided assembly. In JARVIS-agentic, the agent interprets the manual, grounds intended actions in the current scene, and composes skills into executable programs; the skill library handles grasping, constrained motion, placement, and operation-level checks. This division lets an LLM control the full assembly process through interaction with the environment, without predicting robot trajectories or training a task-specific action policy.

Grounded Program Generation. The adapted agent turns a manual-derived assembly step into an executable sequence of tool calls. It queries the perception APIs to locate the relevant bricks and assess the current assembly state, then grounds the intended manipulation in the observed workspace. Based on these results, it generates Python code that selects and parameterizes skills for grasping, transporting, and placing the brick. The skills execute the underlying robot motions and return observable outcomes, which the agent can use to verify the operation and determine its next step. The agent composes existing robot capabilities through code, enabling assembly without training a task-specific action policy or collecting robot action trajectories.

Robot motions are carried out through the motion-planning backend in Isaac Sim Liang et al. (2018). API definitions, skill implementations, and program constraints are provided in the supplementary material.

Execution Memory and Verification. After each execution round, the agent receives updated observations and feedback on the attempted operation. A structured memory records the generated plan and code, API calls, observed object states, and relevant grasp, placement, and runtime outcomes. Before the next reasoning step, this history is compressed into a task-relevant summary. The agent can then assess whether the intended change occurred and, if necessary, revise its next action using the current scene rather than its previous plan alone. Ground-truth target poses, target-scene states, and final evaluation scores remain accessible only to the external evaluator.

Reflection from Execution Experience. We consider two optional reflection settings. In *Standard Reflection*, the agent uses observations, generated code, and execution feedback from the current task to diagnose an unsuccessful attempt and revise its program. *Reflection + Prior-Task* additionally provides task-independent lessons summarized from execution logs of training tasks, including common failures, effective strategies, and API usage. These lessons inform subsequent decisions while the agent continues to reflect on feedback from the current task. The comparison isolates the

Table 2: Model and tool-library performance across task difficulty levels. Underline and blue denote the best generic and task-oriented results; **bold** marks the best Overall score within each library.

Model	Tool library	Soft Score ↑				Valid (%) ↑	Failed actions per task ↓
		Easy	**Medium**	**Hard**	**Overall**		
Gemini-3-Pro (Google, 2026)	Generic	13.27	6.61	4.68	8.19	98.00	6.04
	Task-oriented	22.16	19.30	9.94	17.13	100.00	4.96
Claude-Opus-4.5 (Anthropic, 2026b)	Generic	15.84	16.13	9.51	**13.83**	92.00	7.36
	Task-oriented	22.20	18.58	10.35	17.04	100.00	5.31
Claude-Haiku-4.5 (Anthropic, 2026a)	Generic	11.69	3.22	2.88	5.93	90.00	6.70
	Task-oriented	19.51	9.41	6.12	11.68	96.00	6.10
DeepSeek-V4.1 (Xu et al., 2026)	Generic	15.37	6.33	3.76	8.49	90.00	6.50
	Task-oriented	25.66	18.52	7.98	**17.39**	96.00	4.26
Qwen3-VL-235B (Bai et al., 2025)	Generic	13.78	6.01	5.03	8.27	90.00	6.50
	Task-oriented	21.85	15.75	9.01	15.54	94.00	5.30
Kimi-K2.5 (Team et al., 2026)	Generic	10.37	3.53	2.83	5.58	88.00	6.90
	Task-oriented	17.31	10.31	6.01	11.21	94.00	5.90
GPT-4o (Hurst et al., 2024)	Generic	10.29	2.64	0.93	4.62	24.00	0.72
	Task-oriented	16.21	11.23	7.21	11.55	70.00	3.50
GPT-5.1 (OpenAI, 2026)	Generic	12.38	5.88	4.64	7.63	90.00	6.80
	Task-oriented	20.04	16.26	8.66	14.99	96.00	14.84
GPT-OSS-120B* (Agarwal et al., 2025)	Generic	12.77	6.26	6.12	8.38	92.00	6.50
	Task-oriented	20.66	17.20	9.85	15.90	96.00	5.20

*For GPT-OSS-120B, instruction-manual understanding is performed by GPT-5.1, while subsequent planning and execution are handled by GPT-OSS-120B. Valid (%) denotes the proportion of generated programs that can be executed. Overall is the unweighted mean across the three difficulty levels.

benefit of transferring execution experience across tasks. Neither setting updates model parameters. Memory construction and reflection prompts are detailed in the supplementary material.

5 Experimental Results & Analysis

We evaluate how contemporary multimodal and reasoning models perform when embedded in the JARVIS-agentic framework for full pipeline visual-manual-guided assembly. Our experiments examine not only overall task performance, but also the factors that determine successful physical execution. Particularly, we study whether strong visual planning translates into reliable execution, how agents respond to execution errors, and whether additional reflection improves long-horizon assembly.

Effect of Reasoning Models. We first benchmark 9 contemporary multimodal and reasoning models using the same JARVIS-agentic execution framework and interaction budget. Table 2 reports their assembly performance across tasks of varying construction complexity and assembly horizon.

Tool abstraction has a substantial impact on execution performance. Replacing generic robot-control APIs with our task-oriented primitives improves the Overall score for all nine models, raising the average from 7.88 to 14.71. The gain is largest on Medium tasks, where the average score increases from 6.29 to 15.17. On Hard tasks, the improvement is more modest (4.49 → 8.35), suggesting that better execution abstractions alone are insufficient once long-horizon reasoning and state tracking become dominant bottlenecks.

The tool interface also changes the relative model ranking. Claude-Opus-4.5 (Anthropic, 2026b) leads with generic APIs, whereas DeepSeek-V4.1 (Xu et al., 2026) achieves the highest Overall score (17.39) with task-oriented primitives. Notably, the best Overall performance does not coincide with the fewest failed actions: GPT-4o (Hurst et al., 2024) makes fewer failed actions per task than DeepSeek-V4.1, yet attains a lower Overall score. This suggests that assembly performance depends not only on avoiding invalid operations, but also on whether successful actions produce meaningful structural progress and whether the agent can recover from intermediate mistakes. More broadly, evaluations under low-level APIs may partly conflate reasoning ability with tool orchestration.

What makes long-horizon assembly difficult? Our study shows that assembly difficulty is jointly determined by assembly horizon and initial structural complexity, with current models appearing particularly sensitive to increases in horizon. As shown in Table 3, under low initial complexity, in-

Table 3: Joint effect of initial scene complexity and assembly horizon. Each cell reports the mean soft score, with the number of evaluated tasks in parentheses. Results are computed from the latest no-reflection execution round for each model.

Model	Initial complexity N_{init}	Assembly horizon N_{act} 1–2	3–5	6+
Gemini-3-Pro (Google, 2026)	0–29	43.42 (5)	15.40 (10)	–
	30–78	28.03 (4)	17.99 (8)	10.40 (6)
	79+	16.40 (4)	12.96 (7)	6.01 (6)
Claude-Opus-4.5 (Anthropic, 2026b)	0–29	36.46 (5)	18.38 (10)	–
	30–78	30.87 (4)	14.47 (8)	11.88 (6)
	79+	17.61 (4)	14.05 (7)	5.44 (6)
DeepSeek-V4.1 (Xu et al., 2026)	0–29	40.16 (5)	21.08 (10)	–
	30–78	31.25 (4)	19.47 (8)	9.78 (6)
	79+	15.91 (4)	12.12 (7)	1.83 (6)
Macro average	0–29	**40.01**	**18.29**	–
	30–78	**30.05**	**17.31**	**10.69**
	79+	**16.64**	**13.05**	**4.43**

Table 4: Manual planning and spatial grounding before execution. Matched and missing steps count correctly identified and omitted assembly steps; coverage is the fraction of required steps matched. Position and orientation errors measure placement accuracy over matched steps. Position and orientation errors are computed on matched steps. 3D denotes structured world-frame descriptors added to RGB-D observations. More details about 3D descriptors are in supplementary.

Model	Input	Position Error (mm) ↓	Orientation Error (°) ↓	Matched Steps ↑	Missing Steps ↓	Coverage (%) ↑
Gemini-3-Pro (Google, 2026)	RGB-D observation	78.08	16.87	**40**	**219**	**15.44**
	RGB-D + 3D descriptors	**16.76**	**12.00**	34	225	13.13
Claude-Opus-4.5 (Anthropic, 2026b)	RGB-D observation	49.49	**18.00**	**65**	**194**	**25.10**
	RGB-D + 3D descriptors	**29.09**	23.57	63	196	24.32
Claude-Haiku-4.5 (Anthropic, 2026a)	RGB-D observation	184.12	20.96	**50**	**209**	**19.31**
	RGB-D + 3D descriptors	**30.34**	**16.67**	29	230	11.20
DeepSeek V4.1 Xu et al. (2026)	RGB-D observation	179.84	**16.36**	11	248	4.25
	RGB-D + 3D descriptors	**23.00**	18.97	**26**	**233**	**10.04**
GPT-4o (Hurst et al., 2024)	RGB-D observation	189.64	**9.57**	47	212	18.15
	RGB-D + 3D descriptors	**22.39**	13.49	**48**	**211**	**18.53**
GPT-5.1 (OpenAI, 2026)	RGB-D observation	154.09	**22.50**	16	243	6.18
	RGB-D + 3D descriptors	**20.36**	41.57	**33**	**226**	**12.74**
Qwen3-VL-235B (Bai et al., 2025)	RGB-D observation	171.39	**5.30**	17	242	6.56
	RGB-D + 3D descriptors	**21.42**	9.68	**32**	**227**	**12.36**
Kimi-K2.5 (Team et al., 2026)	RGB-D observation	178.73	**18.46**	26	233	10.04
	RGB-D + 3D descriptors	**21.67**	23.29	**41**	**218**	**15.83**

creasing the assembly horizon from 1-2 to 3-5 bricks reduces the macro-average score sharply from 40.01 to 18.29.. Similar degradation is observed across other complexity regimes, suggesting that even a moderate increase in sequential manipulation steps can substantially increase error accumulation and planning difficulty. Initial structure complexity also affects performance: for short-horizon tasks, the macro-average score decreases from 40.01 to 16.64 as the initial structure grows from 0-29 to $79+$ bricks. However, this effect is generally more gradual and less consistent across individual bins, indicating a stronger dependence on the particular structure and spatial configuration being manipulated. Performance is lowest when both factors are high, suggesting that errors in grounding and execution become harder to contain over longer assembly sequences. The task-distribution heatmap in the supplementary material shows that the Easy, Medium, and Hard partitions span this joint difficulty gradient. Across complexity regimes, assembly horizon remains the more consistent source of performance degradation.

Why Full-Pipeline Evaluation Matters. Planning and grounding scores alone do not capture whether an agent can complete an assembly. Table 4 shows that models can achieve relatively accurate placement estimates for matched steps, while the best model covers only 25.10% of the required steps. Structured 3D descriptors further reduce position error for several models (e.g., 78.08 to 16.76 mm for Gemini-3-Pro), but these intermediate improvements do not ensure successful execution. In the complete pipeline, the best Overall assembly score is only 17.39 (Table 2). Manipulation and placement failures, state changes, and error accumulation introduce challenges that planning metrics cannot measure. We therefore evaluate the assembled outcome, with execution-stage analysis provided in the supplementary material.

Table 5: Performance across reflection rounds with the task-oriented tool library. R1-R3 denote scores after one to three reflection rounds, respectively. Prior-task summaries are distilled from completed assembly tasks. Percentage changes are relative to the preceding round, and Overall is the macro-average across Low, Medium, and High. Blue highlights the largest improvement in each percentage column.

Model	Reflection Framework	Difficulty	w/ Time Penalty					w/o Time Penalty				
			R1	R2	R3	R1→R2 (%)	R2→R3 (%)	R1	R2	R3	R1→R2 (%)	R2→R3 (%)
Gemini-3-Pro	Standard Reflection	Easy	13.926	9.922	9.361	−28.75	−5.65	14.311	13.079	11.220	−8.61	−14.21
		Medium	12.617	11.352	10.241	−10.03	−9.79	13.651	12.320	11.760	−9.75	−4.55
		Hard	11.770	11.891	9.218	+1.03	−22.48	12.452	13.475	10.703	+8.22	−20.57
		Overall	**12.771**	**11.055**	**9.607**	**−13.44**	**−13.10**	**13.471**	**12.958**	**11.228**	**−3.81**	**−13.35**
	Reflection + Prior-Task Experience Summary	Easy	21.370	22.450	19.450	+5.05	−13.36	24.920	26.770	22.130	+7.42	−17.33
		Medium	20.700	21.500	20.060	+3.86	−6.70	23.320	24.520	23.050	+5.15	−6.00
		Hard	13.820	13.200	13.200	−4.49	0.00	13.910	13.800	13.800	−0.79	0.00
		Overall	**18.630**	**19.050**	**17.570**	**+2.25**	**−7.77**	**20.720**	**21.700**	**19.660**	**+4.73**	**−9.40**
Claude-Opus-4.5	Standard Reflection	Easy	14.280	18.070	18.040	+26.54	−0.17	16.510	22.510	22.500	+36.34	−0.04
		Medium	16.610	13.570	13.370	−18.30	−1.47	19.250	16.520	16.510	−14.18	−0.06
		Hard	10.600	9.920	9.620	−6.42	−3.02	10.740	10.190	9.900	−5.12	−2.85
		Overall	**13.830**	**13.850**	**13.680**	**+0.14**	**−1.23**	**15.500**	**16.410**	**16.310**	**+5.87**	**−0.61**
	Reflection + Prior-Task Experience Summary	Easy	25.070	15.890	16.000	−36.62	+0.69	28.250	18.970	19.420	−32.85	+2.37
		Medium	19.010	17.280	17.200	−9.10	−0.46	20.780	19.850	19.880	−4.48	+0.15
		Hard	10.570	10.290	10.730	−2.65	+4.28	11.000	11.130	11.770	+1.18	+5.75
		Overall	**18.220**	**14.490**	**14.640**	**−20.47**	**+1.04**	**20.010**	**16.650**	**17.020**	**−16.79**	**+2.22**
DeepSeek-V4.1	Standard Reflection	Easy	7.950	11.260	10.060	+41.64	−10.66	8.450	13.860	12.640	+64.02	−8.80
		Medium	11.490	15.620	19.880	+35.94	+27.27	15.940	21.130	21.260	+32.56	+0.62
		Hard	10.090	12.230	12.040	+21.21	−1.55	10.180	13.290	13.290	+30.55	0.00
		Overall	**9.840**	**13.040**	**13.990**	**+32.52**	**+7.29**	**11.520**	**16.090**	**15.730**	**+39.67**	**−2.24**
	Reflection + Prior-Task Experience Summary	Easy	25.820	22.690	23.410	−12.12	+3.17	30.730	29.270	30.470	−4.75	+4.10
		Medium	15.330	17.210	18.310	+12.26	+6.39	23.310	21.660	22.310	−7.08	+3.00
		Hard	12.810	10.960	10.060	−14.44	−8.21	13.480	12.260	11.460	−9.05	−6.53
		Overall	**17.990**	**16.950**	**17.260**	**−5.78**	**+1.83**	**22.510**	**21.060**	**21.410**	**−6.44**	**+1.66**

Can Agents Recover from Execution Errors? Across the 25 perturbed tasks, the average score increases only from 54.73 to 55.14 (+0.42 points), indicating limited overall recovery. Outcomes vary substantially across tasks: only 8 improve, although the largest gain reaches 26.22 points, while the largest decline is 15.99 points. Manual inspection suggests that successful recovery tends to occur when the assembly has lower structural complexity at the point of failure, consistent with the performance trends observed above. Task-level results are reported in the supplementary material.

How Much Does Closed-Loop Feedback Contribute to Long-Horizon Execution? We compare standard reflection and reflection augmented with prior-task experience against the no-reflection results in Table 5. Without reflection, Gemini-3-Pro, Claude-Opus-4.5, and DeepSeek-V4.1 score 17.13, 17.04, and 17.39, respectively as shown in Table 2. Under standard reflection, their R1 scores fall to 12.77, 13.83, and 9.84 with the time penalty. Even after removing the penalty, all three R1 scores remain below the no-reflection results. In contrast, prior-task experience raises the time-penalized R1 scores to 18.63, 18.22, and 17.99.

Further reflection rarely closes this gap. DeepSeek-V4.1 improves under standard reflection from 9.84 at R1 to 13.04 at R2, but remains below its no-reflection score; additional rounds provide little consistent benefit across models. These results indicate that current agents struggle to turn feedback from spatially and physically grounded execution into reliable corrections. Summarized experience from previous tasks is more useful than repeated reflection on the current attempt.

6 Conclusion

We introduced JARVIS, a benchmark for autonomous visual-manual-guided execution with structure-aware evaluation, and JARVIS-agentic, a tool-grounded agent requiring no task-specific action trajectories. Experiments show that robot skills and prior-task experience improve assembly performance, while longer horizons, execution errors, and unreliable reflection remain substantial challenges. These findings motivate agents that can more reliably connect visual procedural knowledge with physical execution and error correction.

AI USE STATEMENT

In this work, we used generative AI tools to obtain feedback on the conceptual framing, assist with software development and debugging. We also used them for grammar and clarity edits and suggestions on figure colour schemes.

We did not use generative AI to generate synthetic datasets, formulate mathematical claims, construct or prove theoretical results, or clean or reformat datasets. These tasks were either not applicable or were completed without generative AI assistance.

The authors reviewed all AI-assisted suggestions, code, and edits. We take responsibility for the final content of this work.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. Claude Haiku 4.5. `https://www.anthropic.com/claude/haiku`, 2026a. Accessed: `2026-09-22`.

Anthropic. Introducing claude opus 4.5, 2026b. URL `https://www.anthropic.com/news/claude-opus-4-5`. Accessed: 2026-09-19.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Sachin Chitta. Moveit!: an introduction. In *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pp. 3–27. Springer, 2016.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, et al. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca. Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 4(4):4147–4154, 2019.

Google. Gemini 3 [large language model], 2026. URL `https://gemini.google.com`. Accessed: 2026-09-19.

Chenyang Gu, Jiaming Liu, Hao Chen, Runzhong Huang, Qingpo Wuwu, Xiaoqi Li, Zhuoyang Liu, Ying Li, Renrui Zhang, Peng Jia, Pheng-Ann Heng, and Shanghang Zhang. From manuals to actions: A unified vla model for chain-of-thought manual generation and robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13540–13552, June 2026a.

Chenyang Gu, Jiaming Liu, Hao Chen, Runzhong Huang, Qingpo Wuwu, Xiaoqi Li, Zhuoyang Liu, Ying Li, Renrui Zhang, Peng Jia, Pheng-Ann Heng, and Shanghang Zhang. From manuals to actions: A unified vla model for chain-of-thought manual generation and robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13540–13552, June 2026b.

Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. *The International Journal of Robotics Research*, 44(10-11):1863–1891, 2025.

Yining Hong, Jiageng Liu, Han Yin, Manling Li, Leonidas Guibas, Li Fei-Fei, Jiajun Wu, and Yejin Choi. Esi-bench: Towards embodied spatial intelligence that closes the perception-action loop. *arXiv preprint arXiv:2605.18746*, 2026.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Zhi Jing, Jinbin Qiao, Ouyang Lu, Jicong Ao, Shuang Qiu, Huazhe Xu, Yu-Gang Jiang, and Chenjia Bai. Assemlm: A spatial reasoning multimodal large language model for robotic assembly. *arXiv preprint arXiv:2604.08983*, 2026.

Fucai Ke, Zhixi Cai, Boying Li, Long Chen, Beibei Lin, Weiqing Wang, Pari Delir Haghighi, Gholamreza Haffari, and Hamid Rezatofighi. View2space: Studying multi-view visual reasoning from sparse observations. *arXiv preprint arXiv:2603.16506*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=ZMnD6QZAE6.

Peter Kulits and Cordelia Schmid. Bricknet: Graph-backed generative brick assembly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 39252–39261, June 2026.

LDraw.org. Ldraw: An open standard for lego cad programs. https://www.ldraw.org/, n.d. Accessed: 2026-09-26.

Danrui Li, Jiahao Zhang, Bernhard Egger, Moitreya Chatterjee, Suhas Lohit, Tim K. Marks, and Anoop Cherian. Assemblybench: Physics-aware assembly of complex industrial objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17326–17335, June 2026.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto (eds.), *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pp. 270–282. PMLR, 29–31 Oct 2018. URL https://proceedings.mlr.press/v87/liang18a.html.

Wenbo Ma, Daniel Swoboda, Matteo Tschesche, and Till Hofmann. Workbenchmark: A lego-based assembly benchmark with an assembly-by-disassembly baseline for the smart manufacturing league. *arXiv preprint arXiv:2606.19358*, 2026.

Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022.

OpenAI. Gpt-5.1: A smarter, more conversational chatgpt. https://openai.com/index/gpt-5-1/, 2026. Accessed: 2026-09-22.

Ava Pun, Kangle Deng, Ruixuan Liu, Deva Ramanan, Changliu Liu, and Jun-Yan Zhu. Generating physically stable and buildable brick structures from text. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14798–14809, 2025a. doi: 10.1109/ICCV51701.2025.01373.

Ava Pun, Kangle Deng, Ruixuan Liu, Deva Ramanan, Changliu Liu, and Jun-Yan Zhu. Generating physically stable and buildable brick structures from text. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14798–14809. IEEE, 2025b.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Ziwei Chai, Y Charles, HS Che, Cheng Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Chenrui Tie, Shengxiang Sun, Yudi Lin, Yanbo Wang, Zhongrui Li, Zhouhan Zhong, Jinxuan Zhu, Yiman Pang, Haonan Chen, Junting Chen, et al. Manual2skill++: Connector-aware general robotic assembly from instruction manuals via vision-language models. *arXiv preprint arXiv:2510.16344*, 2025a.

Chenrui Tie, Shengxiang Sun, Jinxuan Zhu, Yiwei Liu, Jingxiang Guo, Yue Hu, Haonan Chen, Junting Chen, Ruihai Wu, and Lin Shao. Manual2skill: Learning to read manuals and acquire robotic skills for furniture assembly using vision-language models. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025b.

Ruocheng Wang, Yunzhi Zhang, Jiayuan Mao, Chin-Yi Cheng, and Jiajun Wu. Translating a visual lego manual to a machine-executable plan. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVII*, pp. 677–694, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-19835-9. doi: 10.1007/978-3-031-19836-6_38. URL https://doi.org/10.1007/978-3-031-19836-6_38.

Yi Ru Wang, Carter Ung, Christopher Tan, Grant Tannert, Jiafei Duan, Josephine Li, Anh Le, Rishabh Oswal, Markus Grotz, Wilbert Pumacay, et al. Roboeval: Where robotic manipulation meets structured and scalable evaluation. *arXiv preprint arXiv:2507.00435*, 2025.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, et al. Enpire: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4. 1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Hongyu Yan and Yadong Mu. Neural assembler: Learning to generate fine-grained robotic assembly instructions from multi-view images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14717–14725, 2025a.

Hongyu Yan and Yadong Mu. Neural assembler: Learning to generate fine-grained robotic assembly instructions from multi-view images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14717–14725, 2025b.

Jichuan Yu, Bowei Li, Zhenran Tang, Guanxing Lu, Chuxiong Hu, Ruixuan Liu, and Changliu Liu. Brickcraft: Visuomotor skill composition with situated manual guidance for long-horizon interlocking brick assembly, 2026. URL https://arxiv.org/abs/2605.07605.

Jiahao Zhang, Anoop Cherian, Cristian Rodriguez, Weijian Deng, and Stephen Gould. Manual-pa: Learning 3d part assembly from instruction diagrams. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6304–6314, 2025. doi: 10.1109/ICCV51701.2025.00595.

SUPPLEMENTARY MATERIAL

A SIMULATED ASSEMBLY TASKS

Figure 3 summarizes the benchmark construction and evaluation pipeline. We begin with visual LEGO instructions paired with their corresponding 3D structural models, preserving step-level correspondence between instructional images and executable assembly states. Rather than treating an entire manual as a single episode, we divide each construction procedure into consecutive task segments. Each segment specifies a pre-assembled initial state, the components that must be manipulated, and the corresponding target construction. The resulting tasks are instantiated in NVIDIA Isaac Sim with physical contact, friction, collision, and robot-control interfaces. Evaluated methods interact with the environment in closed loop, while performance is computed from the final physical construction rather than agreement with a reference action trajectory.

We provide a standardized simulation environment with a Franka Panda robotic platform and simulated LEGO bricks. A total of 50 assembly tasks are designed for evaluation, covering easy, medium, and difficult levels of assembly complexity. For each task, agent are given the corresponding LEGO assembly manual and are required to autonomously construct the target structure in the provided simulation environment.

Each benchmark task is instantiated as an interactive robotic assembly episode in NVIDIA Isaac Sim 6.0.1 Liang et al. (2018), using a Franka Emika Panda manipulator Gaz et al. (2019) with a parallel-jaw gripper. Each scene contains a partially assembled LEGO structure and the movable bricks required by the corresponding manual segment. The agent receives top-down RGB observations and robot-state feedback, and controls the arm and gripper through ROS 2 Macenski et al. (2022). Robot motion is executed with NVIDIA cuMotion/RMPflow, with MoveIt Chitta (2016) supported as an alternative backend.

The environment provides RGB observations from a single planner camera with a top-down view of the LEGO workspace, together with robot joint-state feedback and interfaces for arm and gripper control. Robot execution is integrated through ROS 2 Macenski et al. (2022). By default, robot motion is generated using NVIDIA cuMotion/RMPflow, with MoveIt Chitta (2016) supported as an alternative backend. LEGO bricks are modeled as dynamic rigid bodies with collision geometry, contact dynamics, and friction.

For each episode, the evaluated method receives the visual instruction manual and observations of the initial scene. Depending on the evaluation setting, it operates either without reflection or with closed-loop reflection. In the non-reflective setting, the model generates an executable program directly from the initial inputs, and the program is executed without intermediate model feedback. In the reflective setting, the model receives an updated scene observation after each execution step, assesses the resulting state, and iteratively determines whether to continue, replan, or terminate. Execution ends when the model declares completion or reaches the predefined interaction or wall-clock budget. Ground-truth structural models, object poses, and target placements are used exclusively for evaluation and are not exposed to the evaluated method, except for methods explicitly designated as privileged or oracle baselines.

B ASSEMBLY TRAINING DATASET

We provide more than 200,000 LEGO assembly tasks for model training, constructed from multiple sources, including a synthetically generated LEGO 3D release dataset(Wang et al., 2022), LDraw-based LEGO 3D models (LDraw.org, n.d.), and real-world assembly manuals paired with their LOGO 3D structures. Each 3D model is decomposed into step-wise assembly sequences, where the action brick for each step is identified and the corresponding intermediate assembly scene is reconstructed. For every assembly step, we further record the ground-truth state after the action, including the expected brick configuration and structure state. Each task therefore provides the assembly instructions, target structure, step-level action-brick annotations, intermediate scene configurations, and ground-truth assembly states required to instantiate an interactive robotic simulation environment. This structured formulation enables agents to perform sequential LEGO assem-

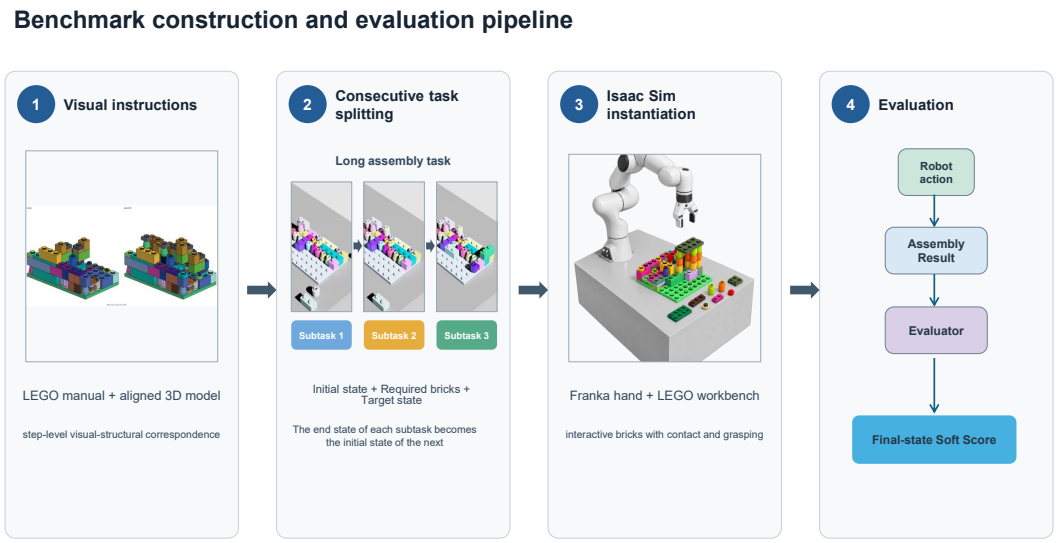


Figure 3: Overview of the benchmark construction and evaluation pipeline. Visual LEGO instructions are first aligned with their corresponding 3D structural models. Long construction procedures are then decomposed into consecutive, self-contained episodes, each defined by an initial state, a set of required bricks, and a target state. These episodes are instantiated as physically interactive robotic assembly environments in NVIDIA Isaac Sim. During evaluation, a method observes the current scene, produces robot actions, and receives updated observations through closed-loop interaction. Performance is measured from the resulting construction state using the proposed Soft Score.

bly while providing reliable ground-truth references for evaluating assembly planning, manipulation execution, intermediate-state correctness, and final construction quality.

To improve data reliability, the constructed tasks are further reviewed by three human evaluators to verify the consistency of the assembly instructions, step decomposition, action-brick annotations, and ground-truth states. However, the current dataset does not provide reference execution trajectories for every training task. In particular, we do not prescribe a unique sequence of low-level robot motions,

B.1 Limitations and Future Directions

JARVIS provides assembly tasks and target brick poses, but does not specify robot motion trajectories. Mapping these targets to feasible motions requires embodiment-specific inverse kinematics and collision-aware planning. Developing methods that generalize this mapping across robot platforms remains an open direction.

B.2 Training-Set Composition and Distribution

The training set comprises 236,679 executable tasks derived from 8,987 valid LEGO assemblies, spanning a broad range of assembly states from sparse local structures to densely populated scenes. As shown in Fig. 4(a), the distribution of assembly horizons is strongly right-skewed. Each task requires an average of 4.94 action bricks, with a median of 4, and approximately 95% of the tasks involve no more than 10 action bricks. Tasks requiring more than 20 action bricks account for only 0.49% of the dataset, while the longest task contains 43 action bricks. The dataset therefore primarily consists of short- to medium-horizon tasks, while retaining a small number of long-horizon examples that capture more challenging cases involving sequential decision-making, state updates, and error accumulation.

Initial-scene complexity exhibits substantially broader coverage, ranging from 0 to 740 pre existing bricks, with a median of 57. As illustrated in Fig. 4(b), tasks containing 30–78 and 79–149 initial

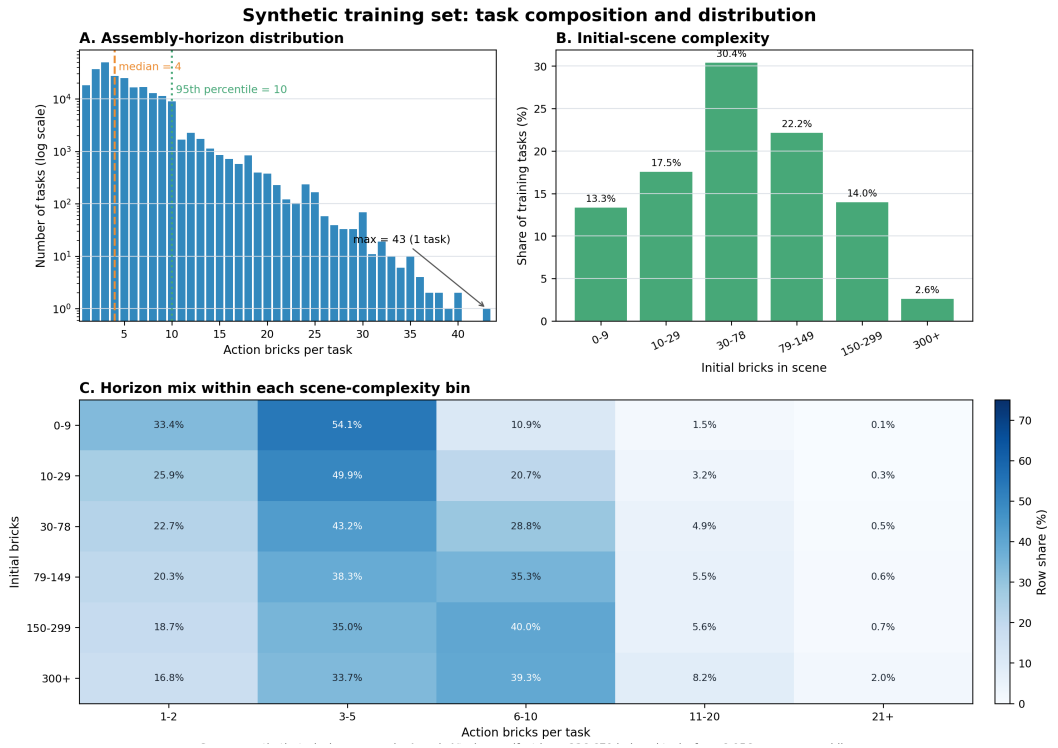


Figure 4: Distribution of the LOGO training set. **(a)** Distribution of assembly horizons measured by the number of action bricks required in each task. **(b)** Distribution of initial-scene complexity measured by the number of bricks present before execution. **(c)** Joint distribution of assembly horizon and initial-scene complexity, where each cell reports the horizon composition within the corresponding scene-complexity interval.

bricks constitute the largest groups, accounting for 30.4% and 22.2% of the dataset, respectively, whereas scenes containing more than 300 initial bricks account for 2.6%. Consequently, the training set includes both relatively simple scenes that support the acquisition of basic localization and placement capabilities and highly cluttered scenes that require target identification among numerous candidate bricks and complex structures.

The joint distribution in Fig. 4(c) further reveals only a weak correlation between initial-scene complexity and the number of action bricks (Pearson's $r = 0.181$). Thus, perceptual complexity and execution horizon are not tightly coupled: the dataset contains both structurally complex tasks requiring only a small number of actions and visually simpler tasks involving extended placement sequences. This distribution enables models to learn target identification in cluttered environments and state tracking over long-horizon execution as partially independent capabilities, while reducing the likelihood that task difficulty can be inferred from scene size alone. Nevertheless, the training distribution remains dominated by short-horizon tasks, with long-horizon and extremely dense scenes confined to the distribution tail.

B.3 Validation Set and Benchmark

The original validation set is generated from 48 source LEGO models and contains 1,277 converted tasks. After filtering for scene completeness, geometric validity, and task executability, 1,067 valid candidate tasks remain. These tasks constitute the validation pool from which the evaluation Benchmark is subsequently sampled. The validation set covers a broad range of assembly horizons and initial-scene complexities. As shown in Figure 5(a), the valid validation tasks require an average of 4.82 action bricks, with a median of 4. Furthermore, 95% of the tasks require no more than 12 action bricks, while the longest task requires 29 brick-placement actions. The distribution is dominated by short- and medium-horizon tasks: tasks requiring 1–2, 3–5, and 6–10 action bricks account

for 22.68%, 50.70%, and 17.34% of the validation pool, respectively. Only two tasks require more than 20 action bricks, corresponding to 0.19% of the valid validation set.

The number of bricks in the initial scene ranges from 0 to 411, with a mean of 80.44 and a median of 54. As shown in Figure 5(b), the largest proportion of validation tasks falls within the range of 30–78 initial bricks, while a small number of high-density scenes contain more than 300 bricks. The resulting validation pool therefore covers basic placement tasks, long-horizon assembly tasks, and visually challenging scenes containing substantial clutter.

Benchmark selection. Task difficulty is determined using the joint criterion introduced in Section 3. For each candidate task, we compute a normalized initial-scene complexity score S_i^{scene} from the number of bricks present at the beginning of the task and a normalized assembly-workload score S_i^{act} from the number of action bricks required to complete the target structure. The overall task difficulty is defined as

$$D_i = 0.3S_i^{\text{scene}} + 0.7S_i^{\text{act}}. \tag{9}$$

Assembly workload is assigned a larger weight because localization errors, grasp failures, and state-estimation errors may accumulate over consecutive execution steps.

After ranking the 1,067 valid candidate tasks according to D_i, we assign the bottom 40% to the Easy candidate stratum, the middle 30% to the Medium stratum, and the top 30% to the Hard stratum. Within each stratum, tasks are sampled at approximately uniform intervals along the ranked difficulty distribution. When multiple candidates have similar difficulty scores, tasks originating from different source LEGO models are prioritized to reduce structural redundancy and increase model diversity. This procedure produces a 50-task Benchmark comprising 20 Easy, 15 Medium, and 15 Hard tasks.

Benchmark distribution. The JARVIS Benchmark contains 50 representative tasks selected from the 1,067 valid validation tasks. Each Benchmark task requires an average of 5.18 action bricks, with a median of 3.5 and a maximum of 29. The initial scenes contain an average of 74.96 bricks, with a median of 58.5 and a maximum of 236. The assembly-horizon distribution remains concentrated in the 3–5 action-brick range, which contains 25 tasks and accounts for 50% of the Benchmark. At the same time, the Benchmark retains an extreme long-horizon task requiring 29 action bricks. This allows the compact evaluation set to assess error accumulation and long-term state tracking under extended execution sequences. In terms of initial-scene complexity, the Benchmark covers scenes ranging from sparse configurations to highly complex scenes containing 150–299 initial bricks, although no task with more than 300 initial bricks is selected.

Relationship between the validation set and the Benchmark. The Benchmark is a representative subset of the valid validation pool rather than an independent data split. Figures 5(a) and 5(b) compare their relative distributions with respect to assembly horizon and initial-scene complexity. Overall, the Benchmark preserves the principal characteristics of the validation distribution. In both sets, tasks requiring 3–5 action bricks form the largest group, and initial-scene complexity is most frequently concentrated in the 30–78-brick interval.

Figure 5(c) further illustrates how the Benchmark covers the joint distribution of initial-scene complexity and assembly horizon. The selected tasks cover most representative combinations of the two factors, while the inclusion of the unique 29-action-brick task strengthens evaluation at the long-horizon tail. Nevertheless, because the Benchmark contains only 50 tasks, several sparsely populated combinations are not represented, particularly the extremely complex region containing more than 300 initial bricks. The purpose of the Benchmark is therefore to cover the principal difficulty gradients of the validation pool under a manageable execution cost, rather than to reproduce every extreme case in the complete validation distribution. Experimental results should consequently be interpreted as a systematic evaluation over the selected representative task range, rather than as an exhaustive measurement of all long-tail cases in the validation set.

Figure 7 presents a representative long-horizon task from Benchmark-50. The episode begins with a partially assembled structure containing 74 bricks and requires the agent to place 29 additional bricks to complete the 103-brick target. Although its initial-scene complexity is moderate relative

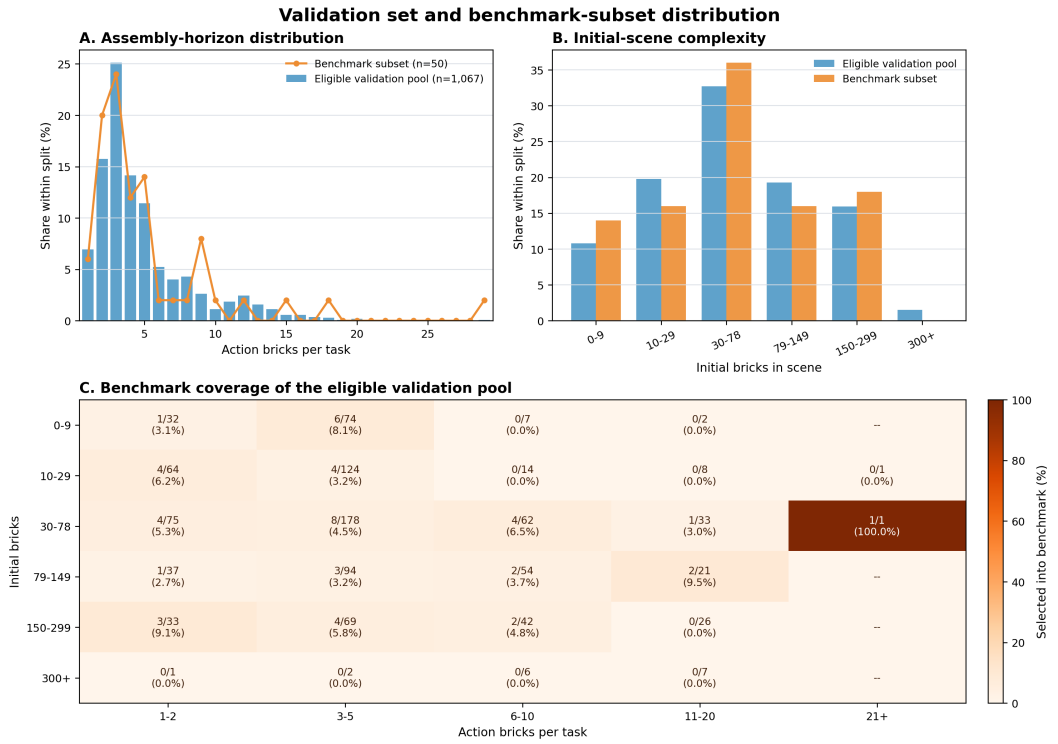


Figure 5: Distributions of the LOGO validation set and the selected JARVIS Benchmark. **(a)** Assembly-horizon distributions measured by the number of action bricks required in each task. **(b)** Initial-scene complexity distributions measured by the number of bricks present before execution. **(c)** Benchmark coverage over the joint distribution of assembly horizon and initial-scene complexity, where each cell reports the number of selected Benchmark tasks relative to the number of eligible validation tasks in the corresponding interval, together with the selection rate.

to the densest validation scenes, its assembly horizon is the longest in the benchmark. The task therefore places substantial demands on instruction understanding, sequential planning, target-brick identification, and state tracking. Because each placement changes the physical scene and constrains subsequent actions, early localization or manipulation errors may propagate throughout the remaining construction sequence. This case exemplifies how the benchmark captures difficulty arising jointly from visual-scene complexity and accumulated long-horizon execution errors.Although the initial construction has only moderate scene complexity, the episode requires the longest normalized action horizon. It is therefore classified as Hard, illustrating how the benchmark captures tasks whose difficulty arises from extended manipulation sequences rather than from scene complexity alone.

C Task Execution Pipeline and Case Study

C.1 Representative Execution Cases

Each task episode follows a unified perception–planning–execution pipeline. The evaluated method first receives a visual assembly manual together with an observation of the current workspace. The manual specifies the target structure and its intended construction procedure, while the scene observation captures the partially assembled structure and the available action bricks. Based on these inputs, the model identifies the relevant bricks, infers the assembly sequence, and determines the next operations to be executed.

The inferred plan is then translated into executable code using the provided API documentation and task-oriented skill library. The generated program can invoke perception, geometric reasoning,

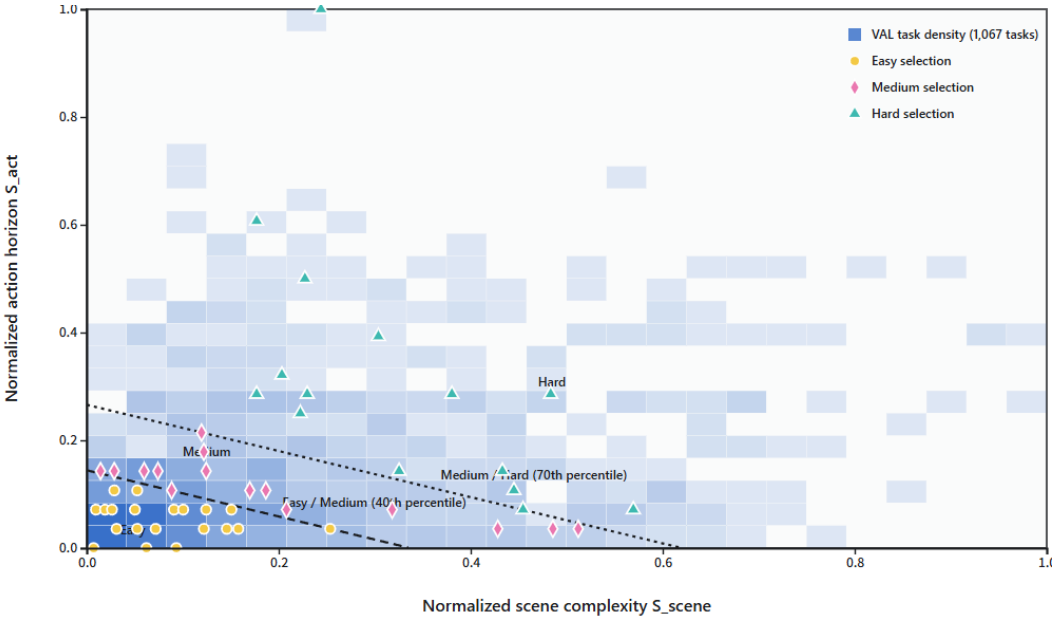


Figure 6: Distribution of the 1,067 eligible VAL tasks and the selected Benchmark-50 tasks across normalized scene complexity and action horizon. For each occupied grid cell, color intensity represents the mean normalized difficulty of the tasks within that cell, with lighter and darker colors indicating lower and higher difficulty, respectively. Empty cells contain no eligible tasks. The overlaid markers denote the selected Easy, Medium, and Hard tasks, while the dashed boundaries indicate the 40th and 70th percentiles used to define the three difficulty strata.

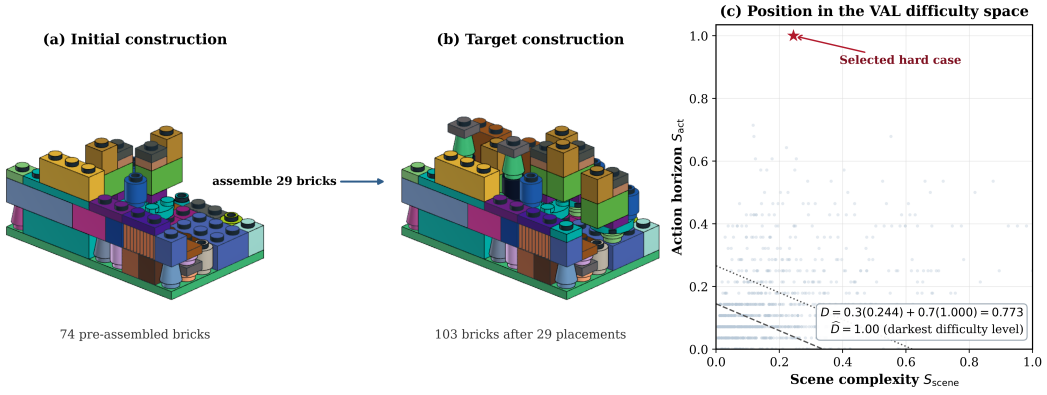


Figure 7: A representative Hard assembly task from Benchmark-50. The initial structure contains 74 pre-assembled bricks, and the agent must place 29 additional bricks to obtain the 103-brick target construction. The task has $S_{\text{scene}} = 0.244$ and $S_{\text{act}} = 1.000$, yielding $D = 0.3S_{\text{scene}} + 0.7S_{\text{act}} = 0.773$ and a normalized difficulty of 1.00 among the 1,067 eligible VAL tasks. Its position in the candidate-task distribution is marked in red.

grasping, and robot-control functions to localize the selected brick, determine its target placement, and execute the corresponding pick-and-place operation. To maintain a consistent and controlled evaluation interface, all methods must use the provided APIs and manipulation primitives; generated programs are not allowed to redefine these interfaces or directly specify unverified grasp coordinates, inverse-kinematics targets, or gripper forces.

Figure 8 illustrates how a generated high-level program is grounded into physical robot execution. Given the current scene state and the target pose specified by the plan, the execution layer selects the required brick and invokes shared grasping and placement primitives. The grasping primitive ap-

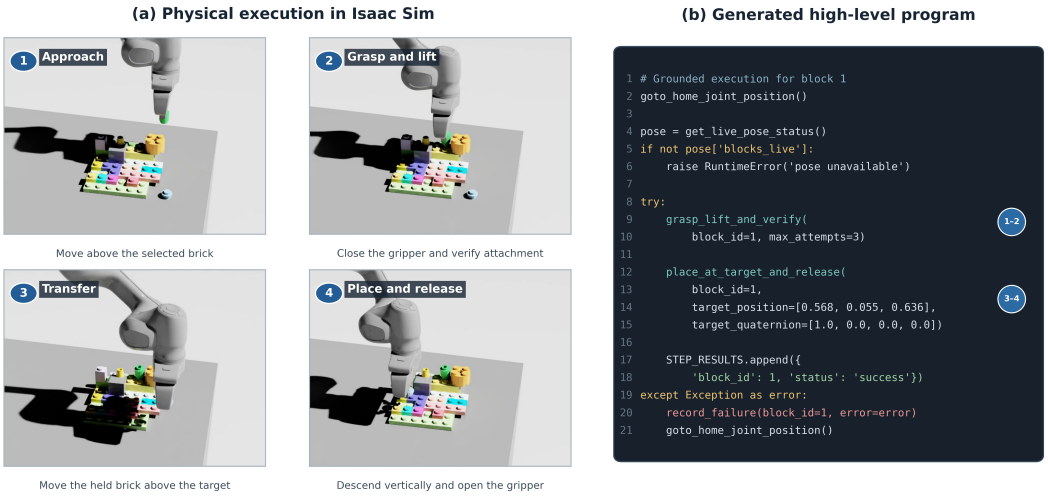


Figure 8: Example of grounded program execution in the interactive assembly environment. The left panel shows four stages from an actual Isaac Sim execution: approaching the selected brick, grasping and lifting it, transferring it above the target, and vertically placing and releasing it. The right panel shows the corresponding high-level program generated for the same operation. The program specifies the selected brick and target pose, while the shared execution primitives handle pose validation, grasp verification, collision-aware motion, and failure recovery.

proaches the brick, closes the gripper, verifies attachment, and lifts the brick vertically. The placement primitive transfers the grasped brick above the target pose, performs a vertical descent, and releases it. These primitives internally perform pose validation, grasp verification, collision-aware motion, and failure handling. During execution, updated object poses are continuously recorded, and exceptions are exposed as explicit execution feedback for subsequent agent decisions. 3D descriptor is a compact representation of a brick's geometry and spatial properties, generated using the provided skill tools to support object identification and pose reasoning.

We evaluate two execution modes. In the *non-reflective* mode, the model generates a complete executable program from the initial manual and scene observation, and the program is subsequently executed without further model intervention. In the *reflective* mode, execution proceeds incrementally in a closed loop. After each operation, the model receives an updated workspace observation and execution feedback, assesses whether the resulting state is consistent with the expected assembly progress, and decides whether to continue, revise the plan, perform a corrective action, or terminate. This produces a closed-loop process of observation, reasoning, code generation, execution, and state reassessment.

C.2 API INTERFACE AND REFLECTIVE

The provided API documentation defines a constrained set of callable skills for scene observation, object grounding, geometric reasoning, point-cloud-based grasp estimation, motion planning, gripper control, and pick-and-place execution. Each function specifies its required inputs, returned observations, and execution constraints, allowing the model to compose robot programs without directly accessing low-level simulator state or issuing unverified control commands. In reflective execution, the model invokes these APIs incrementally rather than generating the entire action sequence at once. After each pick-and-place operation, it receives an updated camera observation and structured execution feedback, compares the observed state with the expected assembly progress, and determines whether the previous action succeeded. The model then either proceeds to the next brick, regenerates the affected portion of the program, performs a corrective action, or terminates the episode.

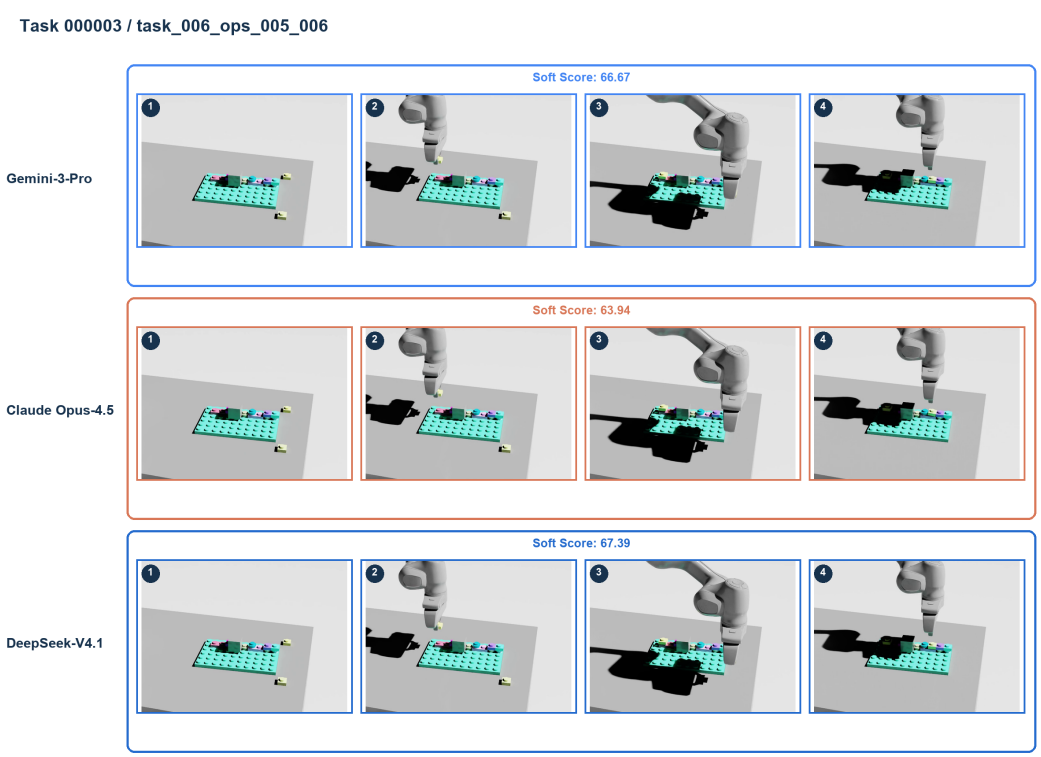


Figure 9: No-reflection execution comparison on Task 000003. The four frames in each row show the initial scene, approach and grasp, transfer, and final placement generated by the corresponding model. Gemini-3-Pro, Claude Opus-4.5, and DeepSeek-V4.1 obtain soft scores of 66.67, 63.94, and 67.39, respectively. All three models complete the manipulation sequence, with the remaining performance differences primarily reflecting final placement accuracy.

C.3 Execution example

We provide representative examples of actual robot execution to analyze how different backbone models behave throughout the assembly process and how differences in their generated actions affect the final construction quality. Specifically, we compare the no-reflection execution trajectories of Gemini-3-Pro, Claude Opus-4.5, and DeepSeek-V4.1 on three representative tasks. Each example presents four key stages—the initial scene, approach and grasp, transfer, and final placement—together with the resulting soft score. These cases allow us to distinguish basic execution feasibility from placement quality and illustrate how positional, vertical, and rotational errors lead to different scores even when all models generate complete and executable pick-and-place sequences.

Task 000003. Figure 9 compares the no-reflection execution trajectories of Gemini-3-Pro, Claude Opus-4.5, and DeepSeek-V4.1 on Task 000003. All three models successfully identify the action brick and generate a complete sequence covering approach, grasping, transfer, and placement. Their final results are visually similar, producing soft scores of 66.67, 63.94, and 67.39, respectively. DeepSeek-V4.1 achieves the highest score, followed closely by Gemini-3-Pro, whereas Claude Opus-4.5 exhibits slightly larger residual placement error. The small score differences indicate that all three models perform reliably on this task and that their remaining differences primarily arise from final pose accuracy rather than execution feasibility.

Task 000024. Figure 10 presents the corresponding comparison on Task 000024. Despite the different backbone models, the three executions follow similar manipulation trajectories and produce comparable final assembly states. Gemini-3-Pro obtains a soft score of 59.26, Claude Opus-4.5 obtains 58.57, and DeepSeek-V4.1 obtains 59.64. The narrow score range suggests that this task is solved with similar effectiveness by all three models. DeepSeek-V4.1 provides the highest-scoring



Figure 10: No-reflection execution comparison on Task 000024. Each row presents four representative stages of execution for one backbone model. Gemini-3-Pro, Claude Opus-4.5, and DeepSeek-V4.1 achieve soft scores of 59.26, 58.57, and 59.64, respectively. Their closely grouped scores indicate similar execution quality, with only minor differences in the final placement pose.

placement, although the margin over Gemini-3-Pro and Claude Opus-4.5 is small. Thus, the observed variation is mainly attributable to minor positional, vertical, or rotational deviations in the final brick pose.

Task 000048. Task 000048 produces a clearer difference among the three backbone models, as shown in Fig. 11. Although all models generate executable pick-and-place sequences, their final placement quality differs substantially. Gemini-3-Pro obtains a soft score of 42.68, compared with 55.06 for Claude Opus-4.5 and 57.78 for DeepSeek-V4.1. The lower Gemini-3-Pro score indicates a larger discrepancy between the achieved and target poses, despite the successful completion of the basic manipulation sequence. Claude Opus-4.5 produces a more accurate final state, while DeepSeek-V4.1 achieves the best alignment among the three models. This case illustrates that successful code execution alone does not guarantee high assembly quality: small errors introduced during grasping, transfer, or release can lead to meaningful differences in the structure-aware soft score.

Case 1: Collision-Induced Failure in Non-Reflective Execution. Moreover, from the perspective of failure modes, an important issue is not the model's inability to generate basic pick-and-place operations, but unintended collisions between the robot arm or the grasped brick and surrounding LEGO bricks during execution. As illustrated in Fig. 13, both Claude Opus-4.5 and GPT-5.1 successfully approach and manipulate the selected brick in the non-reflective setting. However, collisions occurring during transfer and placement disturb the surrounding structure, causing neighboring bricks to shift, topple, or scatter. Consequently, the actual workspace state progressively deviates from that assumed by the initially generated program. Because no updated visual feedback is provided for replanning or corrective execution, these disturbances propagate through the remaining actions and ultimately result in assembly failure.



Figure 11: No-reflection execution comparison on Task 000048. Gemini-3-Pro, Claude Opus-4.5, and DeepSeek-V4.1 obtain soft scores of 42.68, 55.06, and 57.78, respectively. All three models generate executable manipulation sequences, but their final placement accuracy differs substantially. DeepSeek-V4.1 produces the highest-quality final state, whereas the larger residual pose error of Gemini-3-Pro results in a lower soft score.

Case 2: Recovery from Grasp Failure through Reflection. When a grasp cannot be executed successfully, reflection enables the model to inspect the updated visual observation and determine whether the target brick has been correctly grasped and whether the resulting scene is consistent with the expected state. If the brick remains at its initial location, the grasp is misaligned, or the operation disturbs the surrounding structure, the model can re-identify the target, adjust the grasp pose or approach trajectory, and generate a corrective action instead of continuing with a plan based on an incorrect state. As illustrated in Fig. 12, the grasp failure propagates to the subsequent placement stage without reflection, resulting in a soft score of 0.00. Structured reflection partially recovers the assembly process through state reassessment and corrective execution, increasing the soft score to 17.05. This example demonstrates that reflection primarily limits the propagation of local grasping errors, although it does not guarantee complete task recovery.

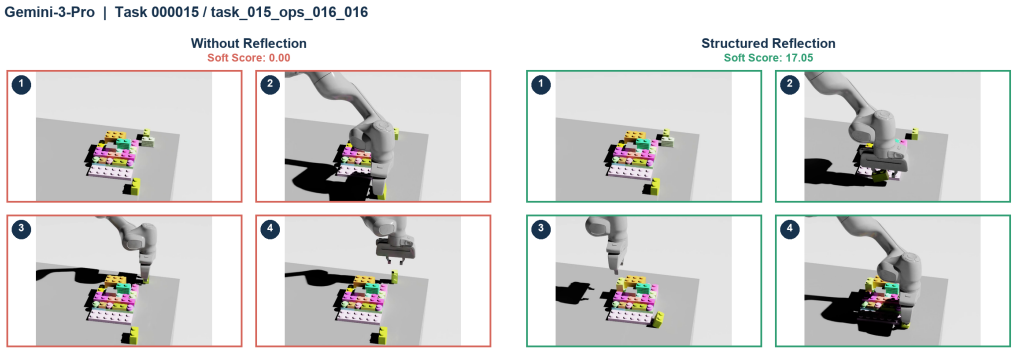


Figure 12: Comparison of Gemini-3-Pro execution with and without structured reflection on the same assembly task. Without reflection, an unsuccessful or misaligned grasp remains undetected and propagates to subsequent actions, producing a soft score of 0.00. With structured reflection, the model uses updated visual observations to reassess the workspace state and generate corrective actions, improving the soft score to 17.05. The numbered frames show representative stages from the initial scene to the final assembly state.

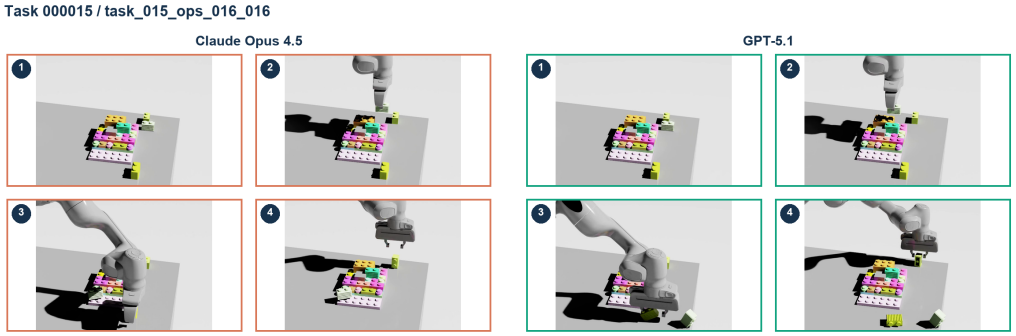


Figure 13: Comparison of non-reflective execution by Claude Opus-4.5 and GPT-5.1 on the same low-scoring task. The numbered frames show representative stages from the initial scene through grasping and transfer to the final state. Although both models generate executable manipulation sequences, unintended collisions with surrounding LEGO bricks disturb the existing structure and cause bricks to shift or scatter. Without visual feedback and corrective replanning, the resulting scene deviation persists until termination, yielding soft scores of 0.43 for Claude Opus-4.5 and 0.00 for GPT-5.1.

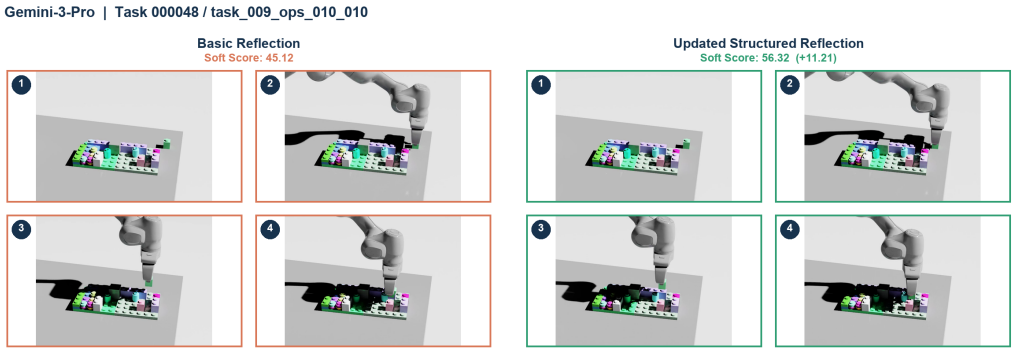


Figure 14: Comparison between *Standard Reflection* and *Reflection + Prior-Task Experience Summary* on the same Gemini-3-Pro assembly task. The numbered frames show representative stages of the two executions. Incorporating prior-task experience produces a more accurate placement, particularly in orientation, increasing the soft score from 45.12 to 56.32. Although the pose error is reduced, neither setting satisfies the strict completion criterion.

Table 6: Effective-action performance under ground-truth-guided assembly. Scores are averaged over effective actions against brick-specific ground truth, without execution-time penalties.

Model	Effective actions	Average score
Gemini 3 Pro	150	**24.25**
Claude Opus 4.5	138	19.68
Claude Haiku 4.5	112	17.66
DeepSeek-V4.1	142	22.66
Qwen3-VL-235B	127	19.23
Kimi-K2	112	17.12
GPT-4o	86	13.20
GPT-5.1	146	19.76
GPT-OSS-120B	**187**	23.76

Case 3: Improved Reflection with Prior-Task Experience. As an example of improved execution, we compare the experience-augmented method, corresponding to the *Reflection + Prior-Task Experience Summary* setting in Table 5, with *Standard Reflection* on the same task. Incorporating prior-task experience primarily improves placement-pose accuracy by helping the model identify positional, vertical, and orientational deviations and translate them into better-constrained corrective actions. In particular, the orientation score increases from 24.57 to 80.63, while the position score improves from 96.20 to 99.72, raising the overall soft score from 45.12 to 56.32. Both settings achieve an execution-validity score and a stability score of 100%, indicating that the improvement arises from the quality of the corrected placement rather than program executability. Nevertheless, the completion score remains zero, showing that prior-task experience substantially reduces the pose error but does not fully recover the intended structural connection in this episode, as illustrated in Fig. 14.

D EXECUTION-STAGE ANALYSIS.

To further isolate execution quality from planning and grounding errors, we evaluate effective actions under ground-truth-guided assembly, as shown in Table 6. For each target brick, the correct brick-specific assembly specification is provided, allowing the model to focus on executing the corresponding manipulation action rather than inferring the next assembly step. The reported score is averaged over effective actions and therefore measures conditional execution quality when a valid assembly target is available.

The results reveal that execution capability and action coverage are distinct factors. Gemini-3-Pro achieves the highest average effective-action score of 24.25 over 150 actions, indicating the strongest per-action execution quality among the evaluated models. In contrast, GPT-OSS-120B produces the largest number of effective actions (187) while achieving a comparable average score of 23.76. DeepSeek-V4.1 also obtains a relatively high average score of 22.66 over 142 effective actions. These results show that some models can execute individual assembly actions reasonably well once the correct target and placement specification are given, even though their full-pipeline assembly performance remains substantially lower.

This gap helps explain why intermediate planning and grounding metrics do not directly translate into successful assembly. In the full pipeline, the agent must repeatedly select the correct construction step, localize the target brick, generate a feasible manipulation command, execute the action, and continue from the resulting physical state. Errors at any stage can propagate to subsequent steps. Consequently, a model with strong conditional execution quality may still achieve a low final assembly score if it fails to generate sufficient valid actions or makes planning errors earlier in the sequence. Conversely, a model that covers more assembly steps may still accumulate larger placement or manipulation errors. This motivates evaluating the final assembled state in addition to planning, grounding, and single-action execution metrics.