DAILYBRICK: LANGUAGE-GUIDED SYNTHESIS OF BRICK ASSEMBLY INSTRUCTIONS

Anonymous authors
Paper under double-blind review



Figure 1: **LEGO models and assembly instructions generated by DailyBrick.** Given a user request, DailyBrick produces a LEGO model together with step-by-step assembly instructions. We recommend watching the video (in the anonymous link) for a better overview.

ABSTRACT

Generating a LEGO model from a user request is more than a plausible 3D shape generation. LEGO models require physical support per brick, and each brick should be placed unblocked. Current methods address this issue by training an expert large language model. We reduce the problem of generating extremely long context on all brick connections to that of writing a program that builds LEGO models like a human, considering each intermediate step could be executed. We introduce DailyBrick, a multi-stage agentic system that produces a LEGO model together with a brick-by-brick assembly instruction for human building and robot assembly. We design a multi-stage harness with roles to help LLMs generate more effectively. A versioned construction SDK compiles multi-level decisions, from semantic subparts to individual bricks, into canonical operations. An external Meta-Reviewer agent replays and inspects the generated package, verifies the model and its construction process, identifies failures, traces them to the responsible stage, and routes them to bounded repairs or to editable candidates for later skill or tool updates. Extensive experiments demonstrate that DailyBrick generates LEGO models based on simple user requests better than recent agentic systems, and show its utility in downstream applications such as robot manipulations on model assembly from instructions and brick sets (`https://anonymousauthors680.github.io/DailyBrick/`).

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.

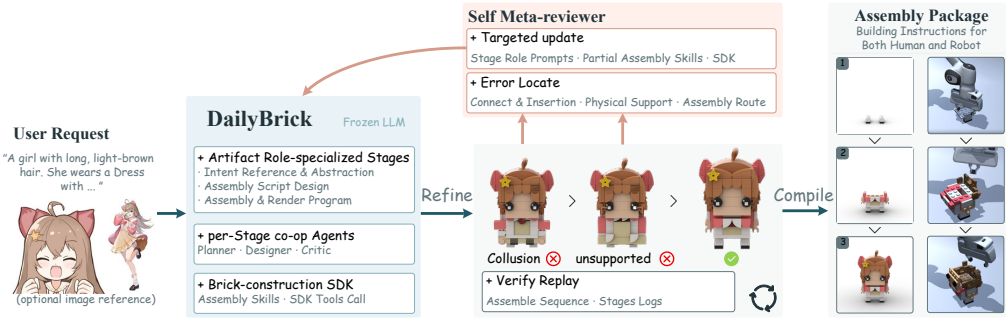

Figure 2: **System overview.** DailyBrick turns a user request into a model, as well as an assembly instruction. Independent prefix replay and an external Meta-Reviewer close the loop by locating failures and routing targeted updates to the responsible stage or SDK component.

1 Introduction

Recent advances in 3D generation have made it possible to create diverse object models from text or images. For objects built from multiple parts, however, generating the final geometry does not by itself specify how those parts should be assembled. Assembly connects a desired object to a sequence of actions and intermediate steps, making it a useful setting for studying long-horizon planning, robot manipulation, and error recovery in embodied systems (Sener et al., 2022; Heo et al., 2023). Generating an object together with its assembly instructions could therefore provide both manuals for a human builder and downstream tasks for robot manipulation. Extending this goal to arbitrary real-world objects is challenging because their parts vary in geometry, connection mechanisms, and physical requirements. LEGO offers a controlled yet expressive setting for studying this problem. Its standardized catalogue defines atomic bricks with fixed geometries and specified connections, while their combinations support diverse objects and structural layouts (Pun et al., 2025; Kulits & Schmid, 2026). This setting allows us to study the generation of both a model and the brick-by-brick sequence that constructs it.

Even with standardized bricks, a plausible LEGO model does not guarantee an executable assembly instruction. Each step must be compatible with the current partial assembly, making construction depend on both the spatial arrangement of bricks and their placement order. Existing approaches address different aspects of this problem. Image2Lego (Lennon et al., 2021) directly fills a final voxel volume with bricks, while ASAP (Tian et al., 2024) plans assembly sequences for known parts and target poses through disassembly search. BrickGPT (Pun et al., 2025) integrates validity checks and physics-aware rollback into autoregressive brick generation. LegoACE (Xu et al., 2025) encodes each brick as tokens, yet its authors report that some outputs remain disconnected. BrickNet (Kulits & Schmid, 2026) further represents typed connections and checks construction prefixes for connectivity and collision. These methods advance assembly planning and model generation, but identifying an invalid step does not by itself determine which construction decision should be revised. A failed step may stem from a brick's pose, an incompatible design, or an earlier structural plan. Linking the failure to the decisions that produced it is therefore important for targeted refinement.

Recent agentic workflows offer a way by decomposing long-context generation into structured stages with intermediate outputs and feedback, enabling iterative refinement of generation programs (Yao et al., 2023; Hong et al., 2023; Wang et al., 2024b). General-purpose coding agents such as Claude Code (Anthropic, 2025), Codex (OpenAI, 2025), and DeepSeek Harness (DeepSeek-AI, 2026) can plan, write, and iteratively refine software with limited human supervision. ReAct (Yao et al., 2023) and CodeAct (Wang et al., 2024b) connect language-model reasoning to tool use and executable actions, while MetaGPT (Hong et al., 2023) coordinates specialized agent roles. In 3D generation, SceneSmith (Pfaff et al., 2026) organizes scene construction into hierarchical stages, and Articraft (Zhou et al., 2026) uses executable programs and feedback to refine generated designs. Applying these ideas to brick assembly requires connecting high-level structural decisions to individual placements, so that a failed construction step can be traced back to the decisions that produced it.

Motivated by these observations, we introduce DailyBrick, **an agentic pipeline that turns user requests into LEGO models and their step-by-step assembly instructions**, reviewing each step and locating errors back to target stages or programs that need refinements. DailyBrick organizes the pipeline as a sequence of bounded stages, from overall layouts and semantic subpart designs to individual brick placement. A Planner-Designer-Critic workflow with separate contexts at each stage, passing only committed stage outputs to the next. Moreover, a Self Meta-Reviewer uses replay and provenance evidence to route detected defects to the implicated stage and stage output. It decides whether the current package(models and instructions) can be accepted or should trigger an update. Additionally, to bridge agents' semantic planning and executable brick operations on LEGO models, we propose a versioned Software Development Kit (SDK). It supports both individual steps (colours, directions, brick ID) with partial assembly skills, allowing agents to express object-level plans as traceable step-by-step assembly instructions. Moreover, Self Meta-Reviewer replays the instructions to verify the result and locates errors in incomplete replay to update related stage, skills, or tools for refinement. Our contributions are as follows:

- We present DailyBrick, a multi-stage agentic pipeline that turns user LEGO requests into 3D models and step-by-step assembly instructions, where a Self Meta-Reviewer would locate errors and achieve target updates.
- We develop a brick-construction SDK with reusable partial assembly skills and traceable brick operations. Combined with verify replay, these operations provide evidence for the Self Meta-Reviewer to locate errors and guide targeted updates.
- Extensive experiments across multiple base models demonstrate that DailyBrick could improve LEGO model generation in visual quality and assembly-sequence validity, while simulation examples illustrate downstream assembly and interaction.

2 Related Work

2.1 Assembly generation and process verification

Assembly datasets and benchmarks study recorded human procedures, robotic execution, and sequence planning (Ben-Shabat et al., 2020; Sener et al., 2022; Heo et al., 2023; Gu et al., 2026; Li et al., 2026). AssemLM (Jing et al., 2026) combines manuals, point clouds, and text for robotic assembly pose reasoning. ASAP (Tian et al., 2024) plans assembly from known parts and target poses through disassembly search. For LEGO generation, Image2Lego (Lennon et al., 2021) converts voxel shapes into bricks and instructions. AssemblyComplete (Chen & Liu, 2024) learns to complete partial LEGO structures while masking physically invalid actions. WorkBenchMark (Ma et al., 2026) studies robot assembly with a disassembly baseline. Brick-Composer (Liu et al., 2026a) studies manual-conditioned brick selection and pose estimation with multimodal models. BrickGPT (Pun et al., 2025) uses validity checks and physics-aware rollback, whereas BrickAnything (Ni et al., 2026) combines structure-aware tokenization with rollback. LegoACE (Xu et al., 2025) generates ordered part tokens, and BrickNet (Kulits & Schmid, 2026) represents typed connections and checks collision. DailyBrick generates the model and assembly instructions from one construction trace, allowing each recorded step to be replayed and located.

2.2 Agentic construction and execution-grounded feedback

ReAct (Yao et al., 2023) and CodeAct (Wang et al., 2024b) connect language-model reasoning to tool use and executable actions, while MetaGPT (Hong et al., 2023) coordinates agent roles. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) use feedback to revise outputs or subsequent behaviour, and Voyager (Wang et al., 2024a) accumulates reusable skills. In 3D generation, SceneSmith (Pfaff et al., 2026) uses agent proposal and critique, and Articraft (Zhou et al., 2026) combines a coding program with execution feedback. AgentTrace (Wang, 2026) analyzes execution traces to localize failures in multi-agent workflows, while SkillForge (Liu et al., 2026b) uses diagnosed failures to refine reusable skills. FAPO (Kassianik et al., 2026) optimizes prompts and pipeline structure using execution feedback, while Lin et al. (Lin et al., 2026) distinguish the ability to produce harness updates from the ability to benefit from them. DailyBrick uses verified evidence to guide targeted updates: the Self Meta-Reviewer locates errors and identifies the relevant stage outputs, skills, or SDK operations to refine.

3 Method

As shown in Fig. 3, DailyBrick takes a user request x and produces a LEGO model with step-by-step assembly instructions, or returns a failure report if the required checks cannot be satisfied. The

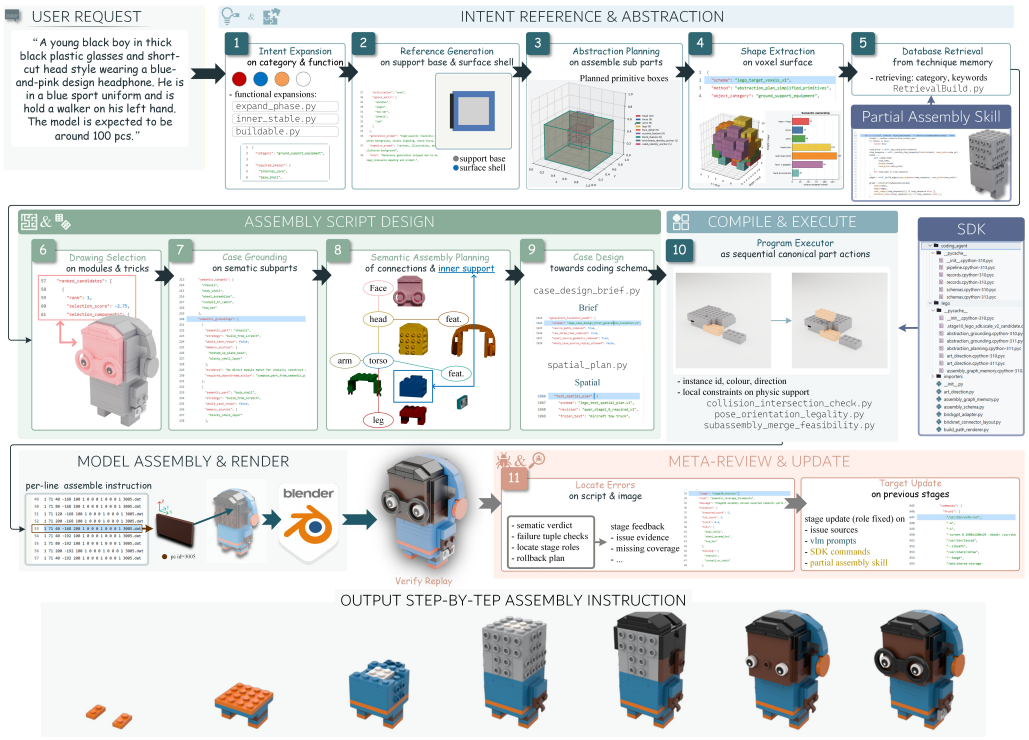

Figure 3: **Overview of DailyBrick.** A multi-stage Planner–Designer–Critic workflow uses partial assembly skills and a construction SDK to generate a shared assembly trace, from which the LEGO model and step-by-step instructions are derived. Independent verify replay provides evidence for the Self Meta-Reviewer to locate errors and guide targeted refinements.

optional image reference would be described as text content for input and verification. Each run uses an updatable configuration Γ, comprising P-D-C prompts, partial assembly skills and the SDK. The assembly instruction is represented by an assembly sequence $\pi = (a_1, \ldots, a_T)$, where each action a_t specifies a brick placement. The resulting construction package is

$$\mathcal{O} = (\mathcal{A}_T, \pi, G, R), \tag{1}$$

where $\mathcal{A}_T$ is the final LEGO model, $G = (g_1, \ldots, g_T)$ contains the instruction views, and R stores verification results and recorded generation dependencies. At step t, the instruction view g_t shows the placement a_t that transforms $\mathcal{A}_{t-1}$ into $\mathcal{A}_t$. Each view g_t illustrates the placement a_t that transforms the partial assembly $\mathcal{A}_{t-1}$ into $\mathcal{A}_t$. The model and instructions therefore describe the same construction sequence, which also provides part identities and ordered poses for downstream simulation and robot planning. Output package acceptance requires both successful verify replay under physical assembly constraints and a review of whether the generated design satisfies the user request. When a verify replay fails, the resulting evidence is passed to the Self Meta-Reviewer for error localizations and targeted updates.

3.1 Multi-stage Planning and Construction

DailyBrick progressively translates the user request into a construction program through explicit stage outputs:

$$x \longrightarrow \mathcal{B} \longrightarrow z \longrightarrow q \xrightarrow{\text{Comp}_\Gamma} \pi, \tag{2}$$

where DesignBrief $\mathcal{B}$ specifies the request, StructuralBlueprint z defines support layout and brick connections. AssemblyProgram q expresses the design through SDK operations. As shown in Fig. 3, these representations summarize the eleven-stage workflow: early stages interpret the request, middle stages develop the structure and its subparts, and final stages compile and verify the assembly. Each stage receives defined inputs and produces an output checked against its requirements.

Each agent-governed stage follows a Planner–Designer–Critic(PDC) loop. The Planner uses a pre-assigned stage role (fixed) and accepted upstream outputs to propose a local plan based on prompts.

The Designer implements the plan within the stage's permitted scope using the available skills and SDK operations. The Critic examines the candidate and requests revisions when needed, while a deterministic reviewer blocks outputs that violate construction requirements and stage roles. The stages use separate contexts and communicate through explicit outputs and feedback on PDC, where dialogues are not passed between stages. Accepted outputs retain their stage identifiers and dependencies, allowing later brick actions to be traced to the decisions that produced them. A refinement creates a new output version while preserving the earlier record.

3.2 Partial Assembly Skills and SDK

The brick-construction SDK connects semantic plans to executable brick operations. Brick-level function calls specify the brick ID, colour, pose, connector, support dependency, and insertion direction. Partial assembly skills compose these calls into reusable procedures for subparts such as a wall, chassis, or roof. Each skill has a defined interface, preconditions, permitted read-write scope, and emitted actions, allowing agents to construct and revise local components within the overall design.

Agents combine these operations and skills in an AssemblyProgram $q = (u_1, \ldots, u_L)$. The deterministic compiler Comp_Γ expands the program into the assembly trace π, recording the originating program operation for each step. Together, these records link a brick action to the stage output and, where applicable, the partial assembly skill and SDK operation that produced it.

3.3 Self Meta-Reviewer: Verify Replay and Refinements

Verify replay checks the assembly trace (π) from the initial state ($\mathcal{A}_0$). Before each placement a_t, it reconstructs the accepted assembly $\mathcal{A}_{t-1}$ and its attachment frontier $\mathcal{F}_{t-1}$, which records the attachment sites available for subsequent placements. It then checks the proposed step against physical constraints in (z):

$$\kappa_t = \mathrm{CheckTransition}_\Gamma(\mathcal{A}_{t-1}, \mathcal{F}_{t-1}, a_t, z), \tag{3}$$

where κ_t records the result of brick pose validity, connector compatibility and subpart assembly connections. A passing placement produces $\mathcal{A}_t$ and updates the frontier. A rejected step stops construction replay and produces a failure report containing the action, violated constraint, and last accepted state.

The Self Meta-Reviewer uses this report and recorded generation dependencies to locate errors and select candidate revision targets. Starting from the rejected step, it traces the associated program operation, stage output, and relevant skill or SDK operation. The violated constraint and dependency links guide the choice of what to revise and which dependent outputs must be re-verified or refined. If the available evidence does not support a revision target, the Self Meta-Reviewer returns `defer`.

Refinement operates at two levels. Within a run, targeted updates modify the affected construction program or stage outputs while keeping the configuration Γ fixed. The revised program is recompiled and undergoes complete verify replay before acceptance. Across completed runs, recurring errors can motivate candidate updates to stage prompts, partial assembly skills, or SDK operations. These candidates are evaluated on the triggering cases and protected cases before being activated in a later configuration.

4 Experiments

4.1 Experimental Setup

Dataset. We evaluate on a self-collected dataset of 358,775 structured LEGO models, including a fixed test split of 18,653 cases. Each model is associated with a text description and brick-count information and is represented using the public LDraw format and part library (LDraw.org Standards Board, 2012). Table 1 compares the dataset with the corpora reported by BrickGPT and BrickNet (Pun et al., 2025; Kulits & Schmid, 2026). These statistics describe dataset scale and part diversity rather than a matched training-data comparison.

Table 1: **Dataset Statistics.** In contrast to BrickGPT (Pun et al., 2025) and BrickNet (Kulits & Schmid, 2026), we increase the scale and part diversity of models.

Dataset	#Models	Avg.	Max	#Types
BrickGPT (Pun et al., 2025)	47,389	107.56	409	8
BrickNet (Kulits & Schmid, 2026)	321,320	126.24	131,632	9,746
Ours	**358,775**	**172.58**	**259,179**	**25,406**



a) BrickGPT

b) Bricknet

c) ours

Figure 4: **Generated Examples.** Rows (a)–(c) show randomly outputs from BrickGPT (Pun et al., 2025), BrickNet (Kulits & Schmid, 2026), and DailyBrick, respectively.

Baselines. We compare with direct pose generation and public implementations of BrickGPT and BrickNet (Pun et al., 2025; Kulits & Schmid, 2026). We also evaluate GPT 6-Astra (OpenAI, 2026) as a text-to-brick baseline and pair it with the DailyBrick framework.

Implementation Details. The main pipeline uses Qwen3.6-27B (Qwen Team, 2026). Planner, Designer, and Critic are assigned as agents with individual context. Each stage permits three role rounds, and a request permits three attempts. A frozen case manifest records the request, inventory, seed, model and prompt versions, role configuration, tool and verifier versions, budgets, serialized artifacts, transition reports, and package decision. Invalid generations remain in the denominator and receive zero for success metrics.

4.2 Main Results

Table 2 evaluates final-model structural validity, visual similarity, and request alignment. Figure 5 presents outputs for the same text prompts, allowing qualitative comparison of object interpretation, component layout, and structural detail. Table 5 evaluates the corresponding construction sequences through independent verify replay, measuring agreement with recorded outcomes and states as well as whether all placements pass the construction checks.



1. A modern representation of a TV stand cabinet with two drawers, featuring with a large screen TV, with a remote control on the it; 2. A charming house with a garage, featuring a small garden; 3. A gray bunker shelter with a sloped roof, solid sides, and an open doorway, designed for minimalistic outdoor protection; 4. A bath duck; 5. A chemistry lab-unit microscope, designed for educational or scientific purposes; 6. A modern coffee maker with a glass carafe, designed for a sleek and functional kitchen setup; 7. A compact lab-usage white container; 8. A cargo ship with a blue hull and red base, featuring a large Empty open hold; 9. A cool-designed road Motorcycle in red & black; 10. A boy talks to a iron robot figure; 11. A medieval castle constructed from bricks, featuring a crenellated top for added fortification; 12. A group of whimsical houseplants display featuring some greens, various potted plants with colorful flowers, and white vases holding red blooms; 13. A small shop with gray brick facade, and a Second brick section on the side, suggesting a retail space; 14. Elephant; 15. A white truck navigating through a forested area, showcasing a realistic miniature landscape; 16. A detailed train track on green grass base, with small plants, designed for a realistic miniature railway scene.

Figure 5: **Text-conditioned generation results.** Outputs for 16 evaluation prompts are arranged in two blocks of eight columns. Within each block, BrickGPT (Pun et al., 2025), BrickNet (Kulits & Schmid, 2026), GPT 6-Astra (OpenAI, 2026) and DailyBrick appear from top to bottom. Each column corresponds to the numbered prompt below.

Efficiency Analysis. Table 3 reports token use and wall-clock time per case alongside construction outcomes, relating generation cost to sequence validity and package acceptance. Prefix pass measures the fraction of cases whose first generated sequence passes all construction checks, whereas

Table 2: **Main Results on Text-Conditioned Generation**. We compare BrickGPT (Pun et al., 2025), BrickNet (Kulits & Schmid, 2026), GPT 6-Astra, and DailyBrick frameworks in visual quality, text alignment, and structural validity. Baseline scores combine published results and additional measurements under their respective evaluation protocols. The best value in each column is shown in **bold**.

Method	Valid (%)↑	Stable (%)↑	CLIP ↑ (Radford et al., 2021)	CLIP-L/14 ↑	DINOv2 ↑ (Oquab et al., 2024)	P_{inv} ↓	VQAScore ↑ (Lin et al., 2024)	PE ↑ (Bolya et al., 2025)	SigLIP2 ↑ (Tschannen et al., 2025)
BrickGPT (Pun et al., 2025)	100.0	97.91	0.322	20.14	0.877	0.064	0.048	0.157	0.050
BrickNet (14B) (Kulits & Schmid, 2026)	96.68	94.30	0.339	22.93	0.920	0.228	0.600	0.282	0.623
GPT 6-Astra (OpenAI, 2026)	100.0	100.0	0.330	26.54	0.895	0.057	0.625	0.297	0.626
DailyBrick (w/ Qwen3.6-27B)	100.0	100.0	0.340	27.85	0.906	0.049	0.736	0.362	0.724
DailyBrick (w/ GPT 6-Astra)	100.0	100.0	**0.358**	**31.26**	**0.923**	**0.044**	**0.767**	**0.363**	**0.734**

Table 3: **Efficiency and Cost Analysis.** Results are reported on the 18,653-case test split. Tokens and time are measured per brick among different model scales. A dash indicates that repair success is not applicable. The best value in each column is **bold**.

Method	Tokens/case ↓	Time (s) ↓	Prefix pass ↑	Repair success ↑	Accept ↑
BrickGPT (Pun et al., 2025)	**2.312k**	65.62	0.706	–	0.684
BrickNet (14B) (Kulits & Schmid, 2026)	3.544k	56.68	0.690	–	0.743
GPT 6-Astra (OpenAI, 2026)	16.24k	324.6	0.884	0.525	0.714
DailyBrick (w/ Qwen3.6-27B)	7.760k	207.7	0.960	0.576	0.834
DailyBrick (w/ GPT 6-Astra)	4.346k	**49.09**	**0.974**	**0.677**	**0.885**

Table 4: **Component ablation of DailyBrick**. Base denotes zero-shot Qwen3.6-27B. Components are added cumulatively: (i) Planner-Designer-Critic collaboration; (ii) the construction SDK and partial assembly skills; and (iii) the Self Meta-Reviewer. Results are reported on the 18,653-case test split. Replay exact, all-prefix pass, and package pass rate are reported in percentages. Failure position is normalized by sequence length. The best values in each column are bold.

Variant	Valid (%)↑	Stable (%)↑	CLIP↑	DINOv2↑	P_{inv}↓	VQAScore↑	PE↑	SigLIP2↑	Replay exact (%)↑	All-prefix pass↑	Failure pos.↑	Accept↑	Tokens/case↓
Base	87.92	82.93	0.310	0.864	0.214	0.567	0.347	0.602	90.57	0.888	0.904	0.563	**5.233k**
Base + (i)	88.17	88.04	0.339	0.897	0.201	0.721	0.360	0.720	94.58	0.936	0.935	0.626	6.175k
Base + (i) + (ii)	**100.0**	99.84	0.339	0.897	0.049	0.721	0.360	0.720	95.62	0.943	0.949	0.676	7.253k
DailyBrick(**full**)	**100.0**	**100.0**	**0.340**	**0.906**	**0.049**	**0.736**	**0.362**	**0.724**	**100.0**	**1.0**	**0.960**	**0.834**	7.760k

Accept measures final package acceptance after the allowed retries. Repair success measures the fraction of repair attempts that pass the required checks.

4.3 Ablation Study

Table 4 evaluates the cumulative addition of Planner–Designer–Critic collaboration, the construction SDK with partial assembly skills, and the Self Meta-Reviewer to single-shot Qwen3.6-27B. This comparison assesses how the components affect model quality, construction validity, and generation cost. Table 6 pairs single-shot generation with DailyBrick for each base model using the same requests and part inventory, testing whether the framework's benefits extend across base models.

Table 5: **Assembly Instruction Evaluation.** Results are reported on the full test split. Replay exact (%) is the fraction of traces whose independent replay matches every recorded step outcome and state, including failures. All-prefix pass (%) is the fraction of sequences with no placement rejected by verify replay. Failure position is the percentage of the first rejected placement, with successful sequences assigned as 1. Best values are bold.

Method	Replay exact (%)↑	All-prefix pass ↑	Failure pos. ↑
BrickGPT (Pun et al., 2025)	**100.0**	0.706	0.98
BrickNet (14B) (Kulits & Schmid, 2026)	96.61	0.690	0.43
GPT 6-Astra (OpenAI, 2026)	96.02	0.884	0.44
DailyBrick (Qwen3.6-27B)	**100.0**	0.960	**1**
DailyBrick (GPT 6-Astra)	**100.0**	**0.974**	**1**

4.4 Assembly Execution and Failure Cases

Figures 6 and 7 illustrate two uses of DailyBrick outputs in simulation. The first shows a simulated manipulator following the generated part identities and ordered poses to assemble a model. The second shows interaction with rotating and articulated components of generated models. Together, these examples illustrate how the construction outputs can support both assembly and subsequent interaction in a simulator.

Table 6: **Base-model ablation.** Each pair compares single-shot generation by a base model with DailyBrick using the same model, denoted by Ours. Results are reported on the test split.

Method	CLIP↑	DINOv2↑	P_{inv}↓	VQAScore↑	SigLIP2↑	Replay exact (%)↑	All-prefix pass↑	Failure pos.↑	Accept↑	Tokens/case↓
GPT-4o	0.306	0.861	0.298	0.543	0.584	89.7	0.871	0.892	0.542	67.8k
Ours (w/ GPT-4o)	0.339	0.903	0.0492	0.728	0.722	96.0	0.958	0.960	0.824	79.0k
Qwen3.6-27B	0.310	0.864	0.214	0.567	0.602	90.6	0.888	0.904	0.563	5.23k
Ours (w/ Qwen3.6-27B)	0.340	0.906	0.0490	0.736	0.724	**100**	0.960	**1.00**	0.834	7.76k
Qwen3.6-35B-A3B	0.316	0.873	0.248	0.599	0.611	91.6	0.900	0.912	0.596	4.97k
Ours (w/ Qwen3.6-35B-A3B)	0.343	0.911	0.0479	0.756	0.727	**100**	0.960	**1.00**	0.846	7.17k
Qwen3.8-27B	0.324	0.885	0.166	0.601	0.622	92.7	0.915	0.927	0.624	4.75k
Ours (w/ Qwen3.8-27B)	0.348	0.917	0.0465	0.758	0.728	**100**	0.965	**1.00**	0.852	7.17k
Gemini-3 Pro	0.327	0.886	0.106	0.619	0.624	95.3	0.946	0.947	0.664	4.99k
Ours (w/ Gemini-3 Pro)	0.357	0.917	0.0452	0.760	**0.734**	**100**	0.971	**1.00**	0.861	6.57k
GPT 6-Astra	0.330	0.895	0.0570	0.625	0.626	96.0	0.884	0.440	0.714	16.2k
Ours (w/ GPT 6-Astra)	**0.358**	**0.923**	**0.0440**	**0.767**	**0.734**	**100**	**0.974**	**1.00**	**0.885**	**4.35k**

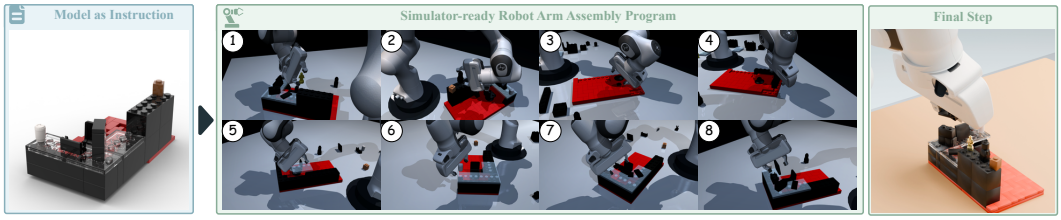

Figure 6: **Step-by-Step Assembly in Simulation.** Left: the generated model. Middle: eight snapshots of a simulated manipulator following the assembly instructions. Right: the manipulator holding the final brick before placement.

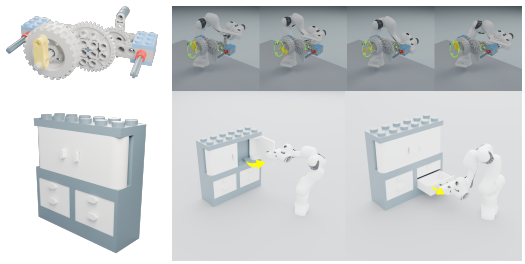

Figure 7: **Physical Interaction in Simulation.** The snapshots show a robot arm could interact with DailyBrick-generated models (rotating & articulated).

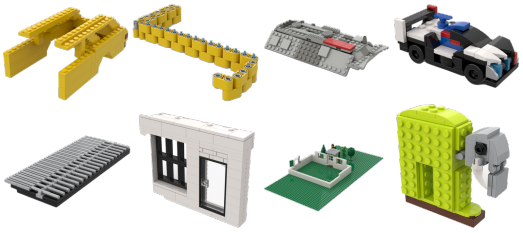

Figure 8: **Limitation.** Models with valid assembly instructions can still exhibit inconsistent color choices, excessively repetitive structures, and unintended asymmetry.

5 Conclusion

We presented DailyBrick, an agentic framework that jointly generates brick models and step-by-step assembly instructions from user requests. By treating the construction sequence as an explicit output, DailyBrick maintains consistency among the final model, intermediate assembly states, and instructions through a shared assembly trace. The framework combines staged planning, partial assembly skills, and a construction SDK with independent verify replays, while recorded dependencies allow the Self Meta-Reviewer to locate errors and guide targeted updates. Experiments across multiple base models show improvements over single-shot generation in visual quality, text alignment, and assembly-sequence validity, supporting the value of jointly generating and checking models and their construction processes. Despite these gains, construction validity does not guarantee design quality. As shown in Figure 8, valid assemblies can still exhibit inconsistent colour choices, excessively repetitive structures, and unintended asymmetry.

6 AI USE STATEMENT

In this work, we used generative AI tools for assist with language polishing. We have not used generative AI tools for generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, interpret results and formulate questions for surveys or interviews, create or modify scientific figures or images, suggest experimental parameters, ,creation of artifacts, draft parts of a research paper, transcribe recordings of research material, summarize or analyse existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, format references, suggest a structure for a research paper, propose a title or keywords for a research paper do not apply to this work. Additionally, we used generative AI tools for grammar checking. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by 2 authors We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. Claude code, 2025. URL https://github.com/anthropics/claude-code. Official product page: https://claude.com/product/claude-code. Accessed: 2026-09-02.

Yizhak Ben-Shabat, Xin Yu, Fatemeh Sadat Saleh, Dylan Campbell, Cristian Rodriguez-Opazo, Hongdong Li, and Stephen Gould. The IKEA ASM dataset: Understanding people assembling furniture through actions, objects and pose, 2020. URL https://arxiv.org/abs/2007.00394.

Daniel Bolya, Po-Yao Huang, Peize Sun, Jang Hyun Cho, Andrea Madotto, Chen Wei, Tengyu Ma, Jiale Zhi, Jathushan Rajasegaran, Hanoona Bangalath, Junke Wang, Marco Monteiro, Hu Xu, Shiyu Dong, Nikhila Ravi, Shang-Wen Li, Piotr Dollar, and Christoph Feichtenhofer. Perception Encoder: The best visual embeddings are not at the output of the network. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2036. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/57bc0a850255e2041341bf74c7e2b9fa-Abstract-Conference.html.

Alan Chen and Changliu Liu. AssemblyComplete: 3d combinatorial construction with deep reinforcement learning, 2024. URL https://arxiv.org/abs/2410.15469.

DeepSeek-AI. Deepseek harness: Everything is a plugin, 2026. URL https://github.com/deepseek-ai/deepseek-harness. Official product page: https://www.deepseek.com/harness/. Accessed: 2026-09-02.

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)*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Gu_From_Manuals_to_Actions_A_Unified_VLA_Model_for_Chain-of-Thought_CVPR_2026_paper.html.

Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J. Lim. FurnitureBench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*, 2023. URL https://arxiv.org/abs/2305.12821.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework, 2023. URL https://arxiv.org/abs/2308.00352.

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, 2026. URL https://arxiv.org/abs/2604.08983.

Paul Kassianik, Baturay Saglam, Huaibo Zhao, Blaine Nelson, Supriti Vijay, Aman Priyanshu, and Amin Karbasi. FAPO: Fully automated prompt optimization of multi-step LLM pipelines, 2026. URL https://arxiv.org/abs/2606.19605.

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)*, 2026. URL https://arxiv.org/abs/2604.22984.

LDraw.org Standards Board. LDraw File Format Specification, 2012. URL https://www.ldraw.org/article/218.html. Revision 1.0.2, 29 April 2012. Accessed: 2026-09-26.

Kyle Lennon, Katharina Fransen, Alexander O'Brien, Yumeng Cao, Matthew Beveridge, Yamin Arefeen, Nikhil Singh, and Iddo Drori. Image2Lego: Customized LEGO set generation from images, 2021. URL https://arxiv.org/abs/2108.08477.

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)*, 2026. URL https://merl.com/research/highlights/assemblybench.

Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2605.30621.

Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *European Conference on Computer Vision*, pp. 366–384, 2024. URL https://arxiv.org/abs/2404.01291.

Jiateng Liu, Bingxuan Li, Zhenhailong Wang, Rushi Wang, Kaiwen Hong, Cheng Qian, Jiayu Liu, Denghui Zhang, Katherine Driggs-Campbell, Manling Li, and Heng Ji. Brick-Composer: MLLMs construct everything from building blocks, 2026a. URL https://github.com/Lumos-Jiateng/Brick-Composer. Manuscript.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. SkillForge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*, 2026b. URL https://arxiv.org/abs/2604.08618.

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, 2026. URL https://arxiv.org/abs/2606.19358.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://dblp.org/rec/conf/nips/MadaanTGHGW0DPY23.

Zhengyang Ni, Feng Yan, Yu Guo, and Fei Wang. BrickAnything: Geometry-conditioned buildable brick generation with structure-aware tokenization, 2026. URL https://arxiv.org/abs/2605.26182.

OpenAI. Codex, 2025. URL https://github.com/openai/codex. Official product page: https://chatgpt.com/codex. Accessed: 2026-09-02.

OpenAI. GPT-6 Astra. OpenAI API model documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-6-astra. Accessed: 2026-09-26.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=a68SUt6zFt.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. SceneSmith: Agentic generation of simulation-ready indoor scenes, 2026. URL https://arxiv.org/abs/2602.09153.

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

Qwen Team. Qwen3.6-27B, 2026. URL https://huggingface.co/Qwen/Qwen3.6-27B. Official model card. Accessed: 2026-09-26.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Fadime Sener, Dibyadip Chatterjee, Daniel Shelepov, Kun He, Dipika Singhania, Robert Wang, and Angela Yao. Assembly101: A large-scale multi-view video dataset for understanding procedural activities, 2022. URL https://arxiv.org/abs/2203.14712.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://dblp.org/rec/conf/nips/ShinnCGNY23.

Yunsheng Tian, Karl D. D. Willis, Bassel Al Omari, Jieliang Luo, Pingchuan Ma, Yichen Li, Farhad Javid, Edward Gu, Joshua Jacob, Shinjiro Sueda, Hui Li, Sachin Chitta, and Wojciech Matusik. ASAP: Automated sequence planning for complex robotic assembly with physical feasibility. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4380–4386, 2024. doi: 10.1109/ICRA57147.2024.10611595. URL https://doi.org/10.1109/ICRA57147.2024.10611595.

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. SigLIP 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features, 2025. URL https://arxiv.org/abs/2502.14786.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024b. URL https://proceedings.mlr.press/v235/wang24h.html.

Zhaohui Geoffrey Wang. AgentTrace: Causal graph tracing for root cause analysis in deployed multi-agent systems, 2026. URL https://arxiv.org/abs/2603.14688.

Hao Xu, Yuqing Zhang, Yiqian Wu, Xinyang Zheng, Yutao Liu, Xiangjun Tang, Yunhan Yang, Ding Liang, Yingtian Liu, Yuanchen Guo, Yanpei Cao, and Xiaogang Jin. LegoACE: Autoregressive construction engine for expressive LEGO assemblies. In *SIGGRAPH Asia 2025 Conference Papers*. ACM, 2025. doi: 10.1145/3757377.3763881. URL https://doi.org/10.1145/3757377.3763881.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation, 2026. URL https://arxiv.org/abs/2605.15187.

A Support Material of DailyBrick



Figure 9: **Staged planning and local assembly in DailyBrick.** The large house example illustrates how DailyBrick plans the main structure and its supports, then completes local components, including an interior sofa, an exterior balcony, and a decorative tree. Unlike approaches that fill a target shape with bricks while overlooking its interior, the staged construction represents both internal layout and external details.

A.1 Additional Qualitative Results

Staged construction of a large model. Figure 9 illustrates how a large design is organized into a main structure and local components. The interior sofa, exterior balcony, and decorative tree represent different parts of the requested scene, while the house provides their spatial context. The complete and exploded views make both the internal arrangement and external details visible, illustrating the component-level organization used in staged construction.

Generation with a restricted inventory. Figure 10 examines generation when specialized parts are excluded from the available inventory. This setting tests how DailyBrick expresses a design using the remaining parts. The examples provide a qualitative view of the resulting shapes and local construction details under the inventory restriction.



Figure 10: **Generation with a restricted brick inventory.** Examples generated by DailyBrick (w/ Qwen3.6-27B) when specialized parts are excluded from the available brick inventory.

Step-by-step assembly instructions. Figure 11 presents step-by-step assembly instructions generated by DailyBrick with Qwen3.6-27B, showing how successive brick placements build up a model from partial assemblies to its completed form. Figure 12 further compares intermediate assemblies from five methods on two furniture examples. Together, these visualizations show how DailyBrick's output specifies a construction process in addition to a final model, and allow comparison with the sequences produced by the other methods.

A.2 Additional Method Details

A.2.1 Stage Coordination and Intermediate Outputs

DailyBrick divides generation into request interpretation, abstraction and shape planning, database grounding, semantic assembly planning, and SDK construction. Each LLM-governed stage follows a Planner–Designer–Critic loop with a prescribed output schema. The Planner uses the stage requirements and upstream outputs to direct the local task. The Designer produces a candidate, and the Critic reviews it together with deterministic validation results. Each round records the planner action, designer output, validation results, critic feedback, and acceptance decision. These records provide evidence for subsequent review and refinement.

A.2.2 Construction Programs and Partial Assembly Skills

An assembly program expresses the design through construction SDK commands. Available operations include frames, panels, and shells, as well as reusable assembly patterns such as stepped roofs, wheel pairs, hinge pairs, and support cores. Partial assembly skills allow local components to be specified through these operations and expanded into explicit brick instances and build steps. Before execution, the program validator checks the command format, permitted operations, and coverage of the expected semantic parts. When required, it also checks whether the program incorporates the retrieved construction techniques and runtime constraints. The resulting construction sequence provides the common basis for the final model, instruction views, and construction checks.

A.2.3 Verify Replay and Construction Evidence

Verify replay checks the recorded construction sequence against the partial assembly at each step. The construction checks examine instance identities, geometric intersections, support relationships, orientations, attachments, and declared dependencies. Step reports record the relevant instance identifiers, support sources, and individual check outcomes. When a check fails, the reported action and violated constraint provide evidence for error localization. These checks assess construction feasibility within the declared geometric and connection constraints. Robot grasping, continuous motion, and force-dependent behavior require additional downstream evaluation.

A.2.4 Self Meta-Reviewer and Targeted Refinement

The Self Meta-Reviewer uses construction reports and stage-level evidence to locate errors and select candidate revision targets. The evidence includes the affected field, defect type, severity, and associated stage outputs. Missing semantic components can motivate a revision of the component plan, color failures can point to database grounding, and program or layout failures can require changes to SDK construction. Within a run, revisions modify the generated program or stage outputs while keeping the configuration Γ fixed. The affected downstream outputs are regenerated and checked again before acceptance. If the evidence is insufficient to select a revision target, automatic refinement is deferred. This subsection describes repairs within a run; updates to prompts, skills, or SDK definitions belong to the later-configuration update process described in the main paper.



Figure 11: **Step-by-Step Assembly Instructions from DailyBrick.** Intermediate assembly states generated by DailyBrick (w/ Qwen3.6-27B), shown in construction order from the initial placements to the completed model.

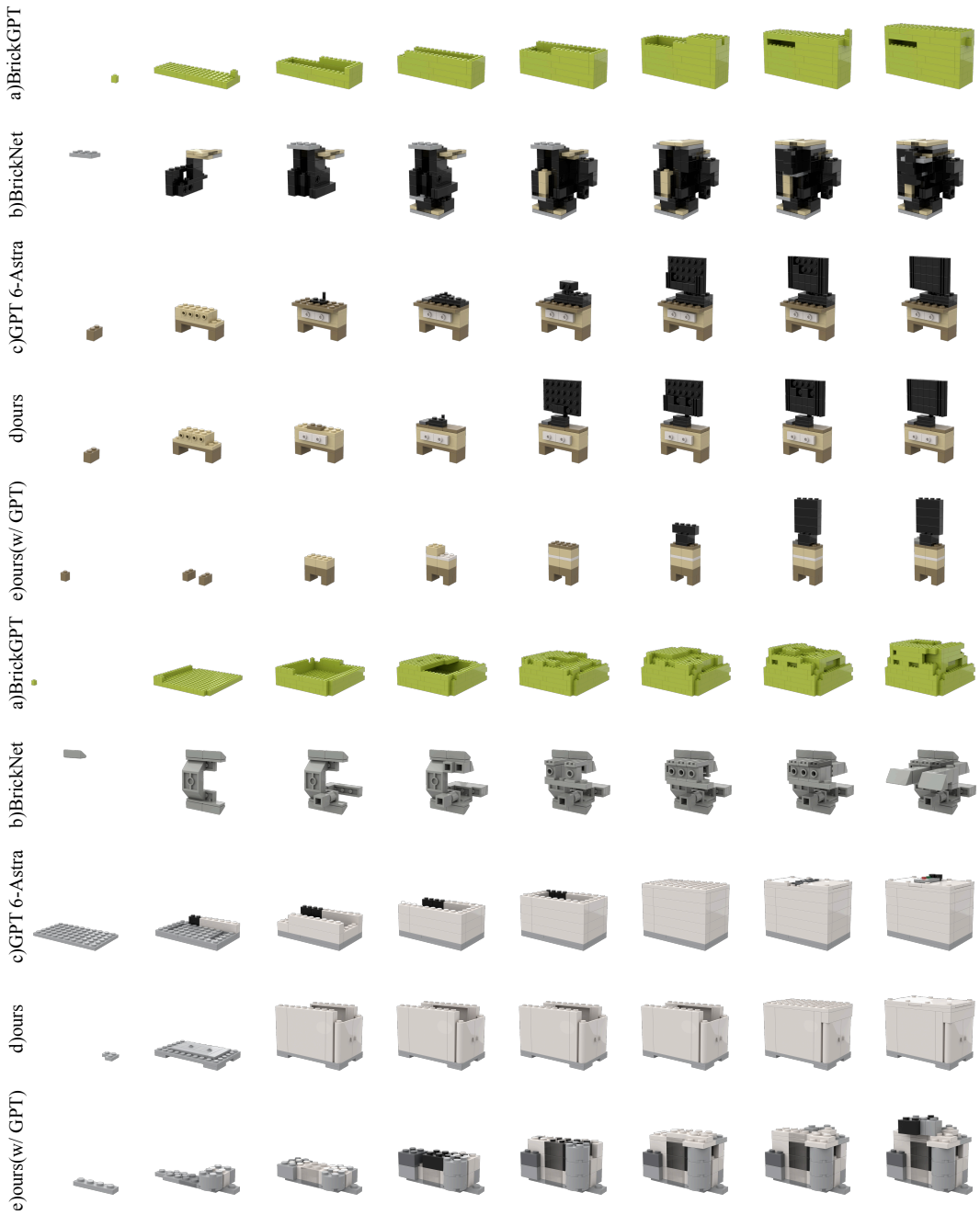

Figure 12: **Step-by-Step Assembly Instruction Visualization of LEGO Models**. The television cabinet and white cabinet are each shown using five methods: BrickGPT, BrickNet, GPT 6-Astra, ours (with Qwen3.6-27B) and ours w/ GPT 6-Astra. Each row presents eight stages of one model: the first assembly step, six uniformly sampled intermediate steps, and the completed model.