BLENDERBUILDER: THINKING FROM A BUILDER'S PERSPECTIVE TO SYNTHESIS MORE COMPLEX OBJECTS

Anonymous authors
Paper under double-blind review



Figure 1: **BlenderBuilder** is a multi-agent framework for image-to-object and image-to-scene generation. By formulating 3D modeling as an operation-centric construction process, it enables the generation of more complex 3D content while maintaining low inference cost.

ABSTRACT

3D coding provides a promising path toward editable, controllable, and executable 3D content generation. However, existing 3D coding agents predominantly construct objects from a parametric perspective, requiring models to describe complex geometry through code or combinations of simple primitives. This formulation becomes increasingly difficult for objects with complex surfaces, fine-grained structures, and long construction processes. We introduce BlenderBuilder, an operation-centric framework that reformulates 3D modeling from *describing target geometry* to *reasoning about how to construct it*. Given a text description or reference image, a Builder plans semantically meaningful modeling operations and selects appropriate tools from a structured Blender Toolkit, while a Coder translates these decisions into executable Blender code. An independent Critic actively inspects intermediate results from informative viewpoints and provides targeted revision feedback, forming an iterative construction and refinement loop. To systematically evaluate 3D coding at both object and scene levels, we further introduce *BlenderObject-250* and *BlenderScene-40*, covering diverse objects and complex indoor and outdoor scenes. Extensive experiments show that BlenderBuilder achieves state-of-the-art performance on both object-level and scene-level generation, with larger gains on more complex scenes. Surprisingly, clearer construction planning and targeted revision also reduce redundant exploration and token consumption. Moreover, evolution experiments show that BlenderBuilder provides increasingly larger gains as the underlying foundation model becomes stronger, suggesting that operation-centric construction can continue to benefit from future advances in 3D reasoning and code generation.

1 INTRODUCTION

Recent advances in multimodal large language models and professional 3D modeling software have made the automatic generation of editable 3D content from text or images an increasingly important research direction. Compared with directly generating non-editable 3D representations, Code-to-3D

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.

methods explicitly construct objects and scenes through programs or software operations, offering better controllability, editability, and reproducibility. Existing approaches can already interact with modeling software such as Blender to generate 3D content according to user instructions. However, they still struggle with complex object modeling, especially when dealing with intricate surfaces, fine-grained local structures, and irregular topology.

We argue that this limitation partly stems from the modeling paradigm adopted by current Code-to-3D methods. Existing approaches typically first describe the target geometry using natural language or parametric representations, and then translate such descriptions into code that approximates the target through the composition, scaling, rotation, and deformation of basic primitives such as cubes, spheres, and cylinders. While this paradigm works reasonably well for regular structures, directly describing the final geometry becomes increasingly difficult when the target contains complex curvature, local deformations, or fine-grained topology. More importantly, this formulation often encourages models to decompose complex objects into collections of simple primitives, without fully exploiting the rich modeling operations available in professional tools such as Blender. In other words, existing methods mainly focus on "what the target geometry should look like", rather than "how the target should be constructed through modeling operations".

This differs substantially from how human artists use 3D modeling software. When modeling complex objects, human users rarely attempt to explicitly describe the complete final geometry in advance. Instead, they reason about the target appearance and structure and plan a sequence of executable modeling operations. For example, a modeler may first create a base mesh and then progressively apply extrusion, subdivision, beveling, Boolean operations, deformation, and other editing operations to obtain the desired geometry. Therefore, although complex objects may be difficult to describe compactly, they can often be constructed effectively through an appropriate sequence of operations. Motivated by this observation, we argue that 3D modeling should not be formulated merely as a mapping from language to geometric code, but rather as a construction problem in which the model plans and executes a sequence of software-native operations conditioned on the target appearance.

Based on this perspective, we introduce **BlenderBuilder**, an operation-centric framework for 3D construction in Blender. Given a target specified by text or images, BlenderBuilder first reasons about how the target should be constructed and decomposes the modeling process into a sequence of semantically meaningful operations. For each step, the Builder jointly determines the modeling action and selects an appropriate tool from a structured Blender Toolkit, while the Coder translates the selected operation, tool interface, and current scene state into executable Blender code. This design separates high-level construction reasoning from low-level software implementation. To further improve the generated results, an independent Critic actively explores the current 3D model from informative viewpoints, identifies modeling discrepancies, and provides targeted revision feedback to guide subsequent construction. Together, these components form an iterative process of planning and tool selection, coding, execution, critique, and revision, enabling BlenderBuilder to progressively construct complex and editable 3D objects and scenes.

We systematically evaluate BlenderBuilder on both object-level and scene-level 3D generation tasks. To support these two settings, we further introduce and release *BlenderObject-250* and *BlenderScene-40*, serving as benchmarks and accompanying datasets for object-level and scene-level 3D coding, respectively. Experimental results show that BlenderBuilder achieves state-of-the-art performance on both tasks, with particularly large gains on the more challenging scene-level setting. **Further ablation and cost analyses demonstrate that clearer construction planning and targeted revision feedback reduce redundant exploration and unnecessary modifications, improving modeling quality while lowering overall token consumption**. More importantly, **our evolution experiments show that the gains brought by BlenderBuilder increase as the underlying foundation models become stronger, suggesting that the framework is not merely compensating for current model limitations, but can continue to benefit from future advances in 3D understanding and code generation.** Our main contributions are summarized as follows:

- We identify a key limitation of existing 3D coding agents: they primarily rely on direct program synthesis or parametric geometric representations, which struggle to effectively express and construct complex surfaces and fine-grained structures. To address this issue, we introduce a **construction-centric modeling paradigm that reformulates 3D genera-**

tion from directly describing target geometry into planning and executing a sequence of modeling operations, substantially improving the ability to model complex objects.

- Based on this paradigm, we propose BlenderBuilder, a closed-loop agent framework that mimics the human modeling process in Blender through iterative planning, execution, inspection, and revision. BlenderBuilder significantly improves modeling quality and complex-object construction capability while **substantially reducing overall modeling and inference costs**.
- We release BlenderObject-250 and BlenderScene-40, comprising 250 high-quality Blender objects and 40 Indoor and outdoor scenes, respectively. The datasets cover diverse categories and structural configurations and provide complete executable construction processes implemented in Blender code, offering new data and evaluation resources for programmatic 3D generation, editing, and scene modeling.

2 Related Works

Multimodal Multi-Agent Systems Large language models have increasingly evolved from passive text generators into agents capable of reasoning, planning, and acting in external environments. ReActYao et al. (2022) interleaves reasoning and action, allowing language models to continuously adapt their behavior according to environmental feedback, while ToolformerSchick et al. (2023) demonstrates the ability of language models to autonomously select and invoke external tools. Subsequent work has extended LLM agents to increasingly complex interactive environments. VoyagerWang et al. (2023) enables long-horizon exploration and skill acquisition in an open world through executable skill libraries and environmental feedback. WebArenaZhou et al. (2024) and VisualWebArenaKoh et al. (2024) extend agentic interaction to realistic web environments, SWE-agentYang et al. (2024) investigates autonomous software engineering through agent–computer interaction, and OSWorldXie et al. (2024) evaluates multimodal agents in real operating systems and desktop applications. Meanwhile, benchmarks such as AgentBenchLiu et al. (2024) reveal that long-horizon planning, reliable tool use, and effective incorporation of environmental feedback remain central challenges for LLM agents. Collectively, these studies establish a general "observe, reason, act, feedback" paradigm, in which models accomplish complex tasks through continuous interaction with external environments. BlenderBuilder follows this paradigm but specializes it to professional 3D modeling software, where an agent must understand a target, plan a modeling process, and manipulate the evolving 3D state through software-native tools.

Programmatic and Agentic 3D Modeling Programmatic representations have long provided a controllable and editable way to construct 3D content. ProcTHORDeitke et al. (2022), InfinigenRaistrick et al. (2023), and Infinigen IndoorsRaistrick et al. (2024) generate large-scale structured 3D environments through procedural rules and programs, laying the foundation for automated 3D content creation. With the growing code generation and spatial reasoning capabilities of large language models, recent work has explored directly generating and manipulating 3D programs with LLMs. 3D-GPTSun et al. (2025) employs multiple agents for procedural modeling in Blender, while SceneCraftHu et al. (2024) combines scene graphs, Blender Python programs, and visual feedback for complex scene generation. Open-UniverseAguina-Kang et al. (2024) Indoor Scene Generation further combines LLM-based program synthesis with open asset repositories to construct indoor environments. Related efforts such as BlenderAlchemyHuang et al. (2024), ScenethesisLing et al. (2026), and 3D-GeneralistSun et al. (2026) investigate VLM-based Blender editing, visually guided scene optimization, and sequential decision making for 3D world construction, respectively.

More recent work has improved complex 3D modeling through code abstraction, structured representations, and closed-loop agents. LL3MLu et al. (2025) coordinates specialized agents for planning, API retrieval, code generation, debugging, and visual refinement. ShapeCraftZhang et al. (2026a) represents objects using structured procedural shape graphs, while ProcFuncRaistrick et al. (2026) introduces function-level abstractions over Blender operations to reduce the complexity of procedural code generation. VIGAYin et al. (2026) further integrates program generation, execution, and visual verification through a continuous code–render–inspect loop for long-horizon interaction.

Despite recent progress, existing approaches largely construct 3D objects from a parametric perspective, representing the target through code, procedural programs, or structured components. In

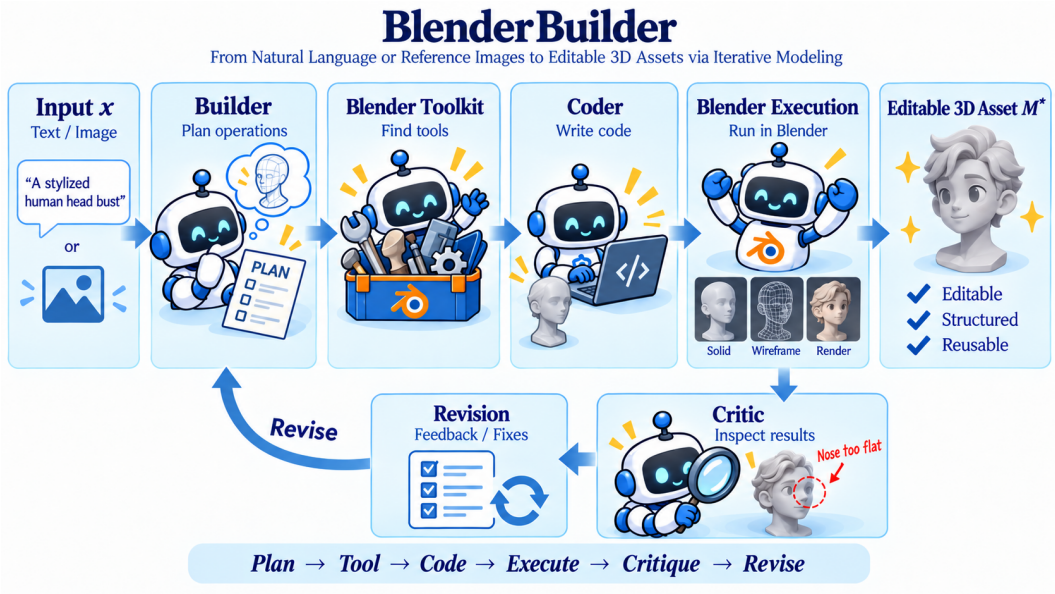

Figure 2: **Overview of BlenderBuilder.** Given a text description or reference image, the **Builder** plans modeling operations and selects appropriate tools from the **Blender Toolkit**. The **Coder** converts each operation into executable Blender code, while the **Critic** inspects intermediate results and provides targeted revision feedback. Through iterative *plan–execute–critique–revise*, BlenderBuilder progressively constructs complex and editable 3D assets.

contrast, BlenderBuilder starts from the construction process itself, reasoning about how an object should be built through a sequence of modeling operations, thereby enabling the construction of more complex 3D objects.

3 METHOD

Overview Given a text description or reference image x, our goal is to construct an editable 3D asset M^* in Blender that satisfies the target specification. BlenderBuilder formulates 3D modeling as an iterative construction process. At iteration t, the **Builder** reasons about how the target should be progressively constructed and jointly determines the next modeling operation o_t and the corresponding tool τ_t from the **Blender Toolkit**. The **Coder** then translates the selected operation and tool, together with the current scene state S_t, into executable Blender code c_t. Executing c_t updates the scene to S_{t+1}. The **Critic** subsequently inspects the updated 3D state, identifies modeling discrepancies, and produces targeted feedback f_{t+1} to guide the next construction step. This process continues until the Critic determines that the target has been satisfactorily reconstructed.

3.1 BLENDER TOOLKIT

We build a structured **Blender Toolkit** on top of Blender's native `bpy` API, encapsulating low-level modeling capabilities into callable actions with explicit semantics and unified interfaces. Each tool is described in a structured format that specifies its name, parameter schema, input types, target objects, return values, and potential error constraints, allowing the model to directly invoke Blender operations without repeatedly implementing low-level APIs from scratch.

We denote the available tool set as $\mathcal{T}$. The current Toolkit covers a broad range of modeling capabilities, including object creation and transformation, mesh and topology editing, curve and surface modeling, sculpting, and local deformation. These operations provide substantially richer modeling capabilities than simple primitive composition, enabling the model to progressively construct complex surfaces, local structures, and highly detailed 3D objects through combinations of software-native operations.

3.2 BUILDER

The Builder is responsible for transforming the target into a sequence of executable construction decisions. Rather than directly generating a complete Blender program, it reasons about *how the target should be progressively constructed* and decomposes the modeling process into semantically meaningful operations.

At iteration t, the Builder jointly determines the next modeling operation and the corresponding tool according to the target input x, the current scene state S_t, the historical construction context H_t, the current feedback f_t, and the available tool set $\mathcal{T}$:

$$(o_t, \tau_t) = \pi_B(x, S_t, H_t, f_t, \mathcal{T}),$$

where $f_0 = \varnothing$ for the initial construction step. The construction history is defined as

$$H_t = \{(o_i, \tau_i, c_i)\}_{i=0}^{t-1}.$$

Each modeling operation is represented as

$$o_t = (g_t, a_t, p_t),$$

where g_t denotes the target object or region to be modified, a_t denotes the modeling operation type, and p_t specifies the high-level modeling intent or constraint. The variable $\tau_t \in \mathcal{T}$ denotes the Blender tool selected for executing the current operation.

Importantly, tool selection is not treated as an independent retrieval process. Instead, it is integrated into the Builder's construction reasoning: the model decides both *what should be done next* and *which tool should be used to accomplish it*. As the scene evolves, the Builder continuously adapts subsequent construction decisions according to the current 3D state and the latest revision feedback.

3.3 CODER

The Coder translates the Builder's selected modeling operation and tool into executable Blender code under the current scene state. Given S_t, o_t, and τ_t, the Coder generates a local executable program

$$c_t = \pi_C(S_t, o_t, \tau_t).$$

The program is then executed in Blender to update the scene state:

$$S_{t+1} = \mathrm{Exec}(S_t, c_t).$$

If execution fails, the system uses the error messages returned by Blender to locally repair c_t, rather than regenerating the entire modeling program. This local execution mechanism reduces the complexity of each code-generation step and mitigates failures caused by long-range state dependencies and API errors in monolithic programs.

3.4 CRITIC

The Critic identifies modeling errors in the updated 3D state and provides targeted revision feedback for subsequent construction. Since geometric discrepancies are often viewpoint-dependent, the Critic is allowed to actively explore the current model by autonomously adjusting the camera rather than relying on a fixed set of observations.

After the t-th operation is executed, the Critic inspects the updated scene state S_{t+1}. It first observes the model from a global viewpoint, identifies potentially problematic regions, and then dynamically explores additional viewpoints according to the accumulated visual evidence. This process allows the Critic to inspect suspicious structures from multiple angles until sufficient evidence is obtained.

Based on these observations, the Critic produces structured feedback

$$f_{t+1} = (r_{t+1}, e_{t+1}, s_{t+1}),$$

where r_{t+1} denotes the problematic region, e_{t+1} describes the identified modeling error, and s_{t+1} specifies the suggested revision direction. This feedback is passed to the Builder and used in the next iteration:

$$(o_{t+1}, \tau_{t+1}) = \pi_B(x, S_{t+1}, H_{t+1}, f_{t+1}, \mathcal{T}).$$

By actively exploring informative viewpoints and providing structured revision guidance, the Critic enables BlenderBuilder to progressively correct modeling errors throughout the construction process.

4 EXPERIMENT

4.1 BENCHMARK CONSTRUCTION

To systematically evaluate scene-level 3D modeling in complex environments, we construct the BlenderScene benchmark. The benchmark covers eight major scene themes, including home, logistics, retail, healthcare and elderly care, street blocks, laboratories, industrial environments, and suburban outdoor scenes, spanning representative indoor and outdoor settings. For each scene theme, we first use GPT-6 to generate a structured scene graph that specifies the object composition, hierarchical organization, and spatial relationships within the scene. Based on the scene graph, we then employ gpt-image-2.5 to generate corresponding reference views, which provide unified visual targets for downstream scene construction and evaluation. To ensure the quality and accuracy of the benchmark, every scene is further manually verified. We examine whether the object categories and quantities are reasonable, whether the spatial relations are correct, whether the scene semantics match the intended theme, and whether the generated reference images are consistent with the scene graph. Samples containing obvious errors or implausible structures are manually corrected. Through this pipeline of structured scene design, visual reference generation, and human quality control, BlenderScene provides high-quality scenes with diverse categories, structures, and complex spatial relations, enabling a more comprehensive evaluation of models in terms of object construction, spatial composition, and overall scene consistency. Meanwhile, we release BlenderObject-250 to evaluate the model's ability to construct individual 3D objects. More details can be found in Appendix.

4.2 EXPERIMENTAL SETTINGS

We evaluate BlenderBuilder on two tasks: **object-level generation** and **scene-level generation**. Given a reference image, the model is required to construct the corresponding 3D object or complete scene from an empty Blender environment. The scene-level task further requires multi-object construction, spatial arrangement, and overall scene organization, making it substantially more challenging than object-level generation.

We compare against direct Blender code generation with GPT-6-Astra, DeepSeek-V4, and Claude-fable-5.1, as well as representative agentic 3D methods, including VIGAYin et al. (2026) and Code2WorldsZhang et al. (2026b). Since Code2Worlds is originally text-conditioned, we convert each reference image into a structured scene description using the same vision-language model. We additionally include Tripo 2.0 as an implicit 3D generation baseline.

For BlenderBuilder, we use GPT-6-Astra as both the Builder and Critic, and instantiate the Coder with either GPT-6-Astra or DeepSeek-V4. We do not impose a fixed number of iterations; the modeling process terminates once the Critic judges the current result satisfactory. All methods use the same target inputs, rendering procedure, and evaluation protocol whenever applicable.

We evaluate generation quality and target consistency using **Photometric Loss(PL)**, **Negative-CLIP Score(N-CLIP)**, and **VLM-Score**, and report total consumed **tokens** as the inference cost.

We further adopt a unified active-view evaluation procedure, allowing informative viewpoints to be selected automatically during comparison. More details are provided in the Appendix.

4.3 Main Results

The results are summarized in Table 1. Overall, BlenderBuilder achieves **state-of-the-art performance** on both object-level and scene-level generation.

Integrating BlenderBuilder consistently improves both GPT-6-Astra and DeepSeek-V4 over their corresponding direct code-generation baselines, indicating that the gains arise from the construction planning and closed-loop refinement framework rather than from a stronger backbone. Moreover, the relative improvement is larger on scene-level generation, suggesting that **operation-centric construction** becomes increasingly beneficial as modeling complexity, spatial relationships, and long-horizon dependencies grow.

Interestingly, despite introducing multi-stage interactions, BlenderBuilder does not incur substantially higher token consumption. Instead, it achieves higher modeling quality with lower or competitive inference cost. We attribute this counter-intuitive result to more effective construction planning and targeted revision, which reduce blind exploration and redundant modifications, leading to a better **performance–cost trade-off**.

Table 1: **Main results on object-level and scene-level generation.** We report PL, N-CLIP, and VLM-Score for generation quality, together with token consumption for inference cost. ↑ indicates higher is better, while ↓ indicates lower is better. The best results are shown in **bold**, and the second-best results are underlined.

Method	Object-level Generation				Scene-level Generation			
	PL↓	N-CLIP↓	VLM-Score↑	Tokens (k)↓	PL↓	N-CLIP↓	VLM-Score↑	Tokens (k)↓
Direct Generation / Coding								
GPT-6-Astra	28.74	41.63	2.71	**2,436**	61.48	78.36	1.67	**60,458**
DeepSeek-V4	31.82	45.76	2.46	10,505	66.37	83.42	1.48	260,751
Claude-fable-5.1	34.65	49.83	2.21	24,382	71.24	88.51	1.29	605,230
Implicit Generation Methods								
Tripo 2.0	13.76	15.94	3.96	N/A	N/A	N/A	N/A	N/A
Agentic / Code-based 3D Methods								
VIGA (ECCV 2026)	19.42	29.58	3.06	5,927	43.61	56.82	2.27	147,114
Code2Worlds (ICML 2026)	17.68	26.74	3.22	5,074	39.84	51.93	2.49	125,954
BlenderBuilder (Ours)								
BlenderBuilder (DeepSeek-V4)	12.84	18.97	3.63	17,508	26.48	35.17	3.08	434,585
BlenderBuilder (GPT-6-Astra)	**7.36**	**10.92**	**4.18**	4,059	**15.72**	**21.46**	**3.81**	100,763

4.4 Ablation Study

We conduct component ablations on the **scene-level generation** task to analyze the contribution of different components to scene construction. All settings use **GPT-6-Astra** as the same backbone model. We compare four configurations: (1) directly generating Blender code with GPT-6-Astra (*Base Model*); (2) introducing the **Builder and Blender Toolkit**, where the Builder plans the modeling procedure and selects the appropriate tools within a single construction pass (*+ Builder & Toolkit*); (3) introducing the **Critic** on top of direct modeling for iterative refinement (*+ Critic*); and (4) the complete BlenderBuilder with Builder, Toolkit, and Critic .

Table 2: **Ablation study and token consumption analysis on scene-level generation.** All variants use GPT-6-Astra as the same backbone. ↑ indicates higher is better, while ↓ indicates lower is better.

(a) Component ablation.

Setting	PL↓	N-CLIP↓	Tokens (k)↓
Base Model	61.48	78.36	**60,458**
+ Critic	46.27	60.84	134,948
+ Builder & Toolkit	29.63	39.72	78,120
BlenderBuilder	**15.72**	**21.46**	100,763

(b) Token consumption analysis.

Setting	Initial (k)	Avg. Revision (k)
+ Critic	**59,763**	12,830
BlenderBuilder	68,120	**4,548**

As shown in Table 2a, the **Builder and Toolkit provide the largest performance improvement**. Compared with the GPT-6-Astra, adding the Builder and Toolkit substantially improves scene modeling quality without requiring additional refinement rounds. This suggests that reformulating direct code generation into planning how to construct the scene and selecting appropriate tools to execute the plan significantly improves the model's initial construction capability. The Critic also improves the results through iterative refinement, but its standalone gain is smaller than that of the Builder and Toolkit. Combining high quality initial construction with targeted subsequent refinement yields the best overall performance in the full BlenderBuilder.

To further understand why BlenderBuilder reduces token consumption, we compare the token usage of **Base Model + Critic** and the full **BlenderBuilder** across different modeling stages. As shown in Table 2b Specifically, we separately measure the token cost of the initial construction round and the average token consumption of subsequent refinement rounds. Although BlenderBuilder incurs a higher token cost in the first round due to additional construction planning and tool selection, its average token consumption in later editing rounds is substantially lower than that of the Critic-only setting.

Our analysis suggests that, during iterative refinement, the model tends to modify the existing 3D state locally rather than reconstruct problematic structures from scratch. Consequently, when the initial model contains substantial errors, later rounds may require repeated diagnosis, modification, and debugging to gradually correct them, resulting in high cumulative inference cost. In contrast, BlenderBuilder produces a higher-quality and better-structured initial model through the Builder and Toolkit, allowing the Critic to focus primarily on correcting residual local errors. Therefore, although BlenderBuilder spends more tokens during the initial construction stage, the improved initialization significantly reduces the cost of subsequent refinement and enables each editing round to produce more stable performance gains, ultimately leading to better overall token efficiency. ‘

4.5 Evolution Experiment

To investigate whether BlenderBuilder remains effective as the underlying model capability improves, we conduct an evolution experiment using models of increasing capability from the GPT family, including GPT-5, GPT-5.6, and GPT-6 Astra. For each backbone, we compare direct modeling with BlenderBuilder. As shown in Fig. 3, BlenderBuilder consistently improves performance across all model scales, with larger gains on stronger backbones.

We attribute this trend to two complementary effects. First, BlenderBuilder shifts the modeling objective from *describing geometry* to *planning how to construct it*, allowing stronger models to better translate their improved 3D understanding and code-generation ability into executable modeling operations. Second, the Critic replaces unguided trial-and-error refinement with targeted diagnosis and correction, enabling increasingly capable models to make more effective revisions.

These results suggest that BlenderBuilder is not merely a mechanism for compensating for weak models, but a scalable modeling paradigm that continues to benefit from advances in foundation-model capability: **from describing objects to constructing objects, and from blind refinement to targeted correction**.

To investigate whether BlenderBuilder remains effective as the underlying model capability improves, we conduct an evolution experiment using GPT-5, GPT-5.6, and GPT-6 Astra. For each backbone, we compare direct modeling with BlenderBuilder. As shown in Fig. 3, BlenderBuilder consistently improves performance across all model scales, with larger gains on stronger backbones.

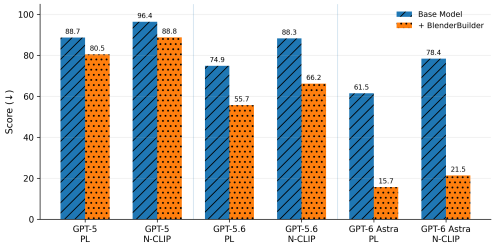

Figure 3: Evolution experiment.

We attribute this trend to two complementary effects. First, BlenderBuilder shifts the modeling objective from *describing geometry* to *planning how to construct it*, allowing stronger models to better translate their improved 3D understanding and code-generation ability into executable mod-

eling operations. Second, the Critic replaces unguided trial-and-error refinement with targeted diagnosis and correction, enabling increasingly capable models to make more effective revisions.

4.6 HUMAN EVALUATION

To further assess the practical quality of the assets generated by BlenderBuilder, we conduct a human evaluation with 30 participants, divided into five groups of six. Each group consists of two graduate students familiar with Blender and four undergraduate students. Participants are allowed to freely inspect the generated 3D assets from multiple viewpoints and rate them on a 1–5 Likert scale along three dimensions: Quality, Realism, and Fidelity. Quality measures geometric completeness, structural correctness, surface quality, and visible artifacts; Realism evaluates whether the generated assets exhibit natural shapes, proportions, and local details; and Fidelity measures consistency with the input text or reference image in terms of overall appearance, structure, and key details.

The results show that BlenderBuilder receives consistently high human ratings across all three dimensions. Specifically, the average scores for Quality, Realism, and Fidelity are 4.42, 4.18, and 4.36 out of 5.0, respectively. To further assess the reliability of the human judgments, we compute the Intraclass Correlation Coefficient (ICC) across the five evaluation groups. The ICC values for Quality, Realism, and Fidelity are 0.86, 0.82, and 0.88, respectively, indicating a high level of agreement across independent evaluator groups. These results demonstrate that BlenderBuilder consistently produces 3D assets with strong modeling quality, visual realism, and target fidelity, further supporting the practical usability of the generated assets.More details can be found in the Appendix.

In addition, we release BlenderScene-40-Datasets and BlenderObject-250-Datasets to the community, containing 40 scene-level Blender assets and 250 object-level Blender assets, respectively, together with their corresponding executable Blender construction processes. We hope these resources can provide a unified and reusable foundation for future research on programmatic 3D generation, object modeling, scene construction, and the evaluation of 3D coding agents.

5 CONCLUSION

3D coding is a highly promising direction for 3D content generation. In this work, we introduce BlenderBuilder, an operation-centric modeling framework that substantially improves the construction of complex objects and scenes, achieving state-of-the-art performance on both object-level and scene-level generation tasks. Further experiments show that, through clear construction planning and targeted revision guidance, BlenderBuilder improves modeling capability while reducing token consumption. Moreover, its benefits grow as the underlying foundation models become stronger, suggesting that BlenderBuilder can continue to provide effective support even as future models further improve.

AI USE STATEMENT

In this work, we used generative AI tools for designing and providing feedback on research methodology and experiments, implementing methods, assisting with translation, and interpreting results. We have not used generative AI tools for generating synthetic datasets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, or cleaning and reformatting datasets, and theoretical model development, qualitative and thematic data analysis, and other proof-related tasks are not applicable to this work. Additionally, we used generative AI tools for creating or editing software code, drafting parts of the paper, summarizing and analyzing existing literature, brainstorming, sourcing and searching for information, identifying relevant literature, and editing the paper to improve readability. We have reviewed all AI-assisted work. LLM-assisted research suggestions were manually evaluated by the authors, relevant literature was checked against the original sources, LLM-assisted code was verified and tested by the authors, and AI-assisted interpretations were checked against the actual experimental results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Rio Aguina-Kang, Maxim Gumin, Do Heon Han, Stewart Morris, Seung Jean Yoo, Aditya Ganeshan, R Kenny Jones, Qiuhong Anna Wei, Kailiang Fu, and Daniel Ritchie. Open-universe indoor scene generation using llm program synthesis and uncurated object databases. *arXiv preprint arXiv:2403.09675*, 2024.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in neural information processing systems*, 35:5982–5994, 2022.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *ICML*, pp. 19252–19282, 2024.

Ian Huang, Guandao Yang, and Leonidas Guibas. Blenderalchemy: Editing 3d graphics with vision-language models. In *European Conference on Computer Vision*, pp. 297–314. Springer, 2024.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Max Li. Scenethesis: A language and vision agentic framework for 3d scene generation. In *International Conference on Learning Representations*, volume 2026, pp. 136596–136629, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Sining Lu, Guan Chen, Nam Anh Dinh, Itai Lang, Ari Holtzman, and Rana Hanocka. Ll3m: Large language 3d modelers. *arXiv preprint arXiv:2508.08228*, 2025.

Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. Infinite photorealistic worlds using procedural generation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12630–12641. IEEE, 2023.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, et al. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21783–21794. IEEE, 2024.

Alexander Raistrick, Karhan Kayan, Jack Nugent, David Yan, Lingjie Mei, Meenal Parakh, Hongyu Wen, Dylan Li, Yiming Zuo, Erich Liang, et al. Procfunc: Function-oriented abstractions for procedural 3d generation in python. *arXiv preprint arXiv:2604.26943*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3d-gpt: Procedural 3d modeling with large language models. In *2025 International Conference on 3D Vision (3DV)*, pp. 1253–1263. IEEE, 2025.

Fan-Yun Sun, Shengguang Wu, Christian Jacobsen, Thomas Yim, Haoming Zou, Alex Zook, Shangru Li, Yu-Hsin Chou, Ethem Can, Xunlei Wu, et al. 3d-generalist: Vision-language-action models for crafting 3d worlds. In *2026 International Conference on 3D Vision (3DV)*, pp. 1873–1883. IEEE, 2026.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, Chenyang Wang, Xiuyu Li, Michael J Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Vision-as-inverse-graphics agent via interleaved multimodal reasoning. In *European Conference on Computer Vision*, pp. 557–575. Springer, 2026.

Shuyuan Zhang, Chenhan Jiang, Zuoou Li, and Jiankang Deng. Shapecraft: Llm agents for structured, textured and interactive 3d modeling. *Advances in Neural Information Processing Systems*, 38:65116–65144, 2026a.

Yi Zhang, Yunshuang Wang, Zeyu Zhang, and Hao Tang. Code2worlds: Empowering coding llms for 4d world generation. *arXiv preprint arXiv:2602.11757*, 2026b.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

APPENDIX

A BLENDER TOOLKIT

We build a structured **Blender Toolkit** on top of Blender's native `bpy` API. Rather than exposing the model directly to low-level Blender APIs or unrestricted scripts, we encapsulate commonly used Blender capabilities into structured callable actions with explicit semantics and unified interfaces.

Each action specifies its tool name, parameter schema, input types, target objects, expected outputs, and possible execution constraints. This structured representation allows the model to invoke Blender operations in a consistent and traceable manner.

The Toolkit covers a broad range of native modeling capabilities. At the object level, it supports object creation, deletion, transformation, grouping, and assembly. For geometry construction, it provides mesh and topology editing operations such as extrusion, beveling, bridging, cutting, Boolean operations, remeshing, and modifier manipulation. It further includes curve and surface modeling, local deformation, sculpting and multiresolution editing, as well as material, UV, Geometry Nodes, hair, rigging, animation, reference matching, asset management, and inspection tools. In addition, the runtime can inspect scene changes, verify execution results, save checkpoints, and recover from failed operations.

These structured actions expose Blender's native modeling space to the agent at a substantially higher semantic level than raw API calls. In particular, operations such as mesh editing, surface manipulation, sculpting, and local deformation allow the model to construct geometry beyond simple primitive composition. By combining these software-native operations, BlenderBuilder can progressively create complex surfaces, local structures, and detailed 3D objects and scenes. "'

B Experiment Details

Model Configuration. In the main experiments, we use **GPT-6-Astra** as both the Builder and the Critic. The Builder is responsible for high-level construction planning and tool selection, while the Critic inspects intermediate 3D results and provides targeted revision feedback. For the Coder, we evaluate two configurations using either **GPT-6-Astra** or **DeepSeek-V4**, allowing us to examine whether BlenderBuilder generalizes across different code-generation backbones. For each backbone, we use a consistent inference configuration across all runs.

Blender Environment. All code-based methods are executed using **Blender 5.0**. We adopt a unified rendering and evaluation pipeline across all methods whenever compatible with their original implementations. Method-specific libraries, procedural assets, or dependencies required by individual baselines are preserved. All quantitative evaluation renderings use the same image resolution of 512×512 and identical rendering configurations across compared methods.

Construction and Tool Use. All BlenderBuilder tasks start from an empty Blender scene. Given a reference image, the Builder reasons about how the target should be progressively constructed and decomposes the task into a sequence of semantically meaningful modeling operations. For each operation, the Builder jointly determines the modeling action and selects an appropriate tool from the Blender Toolkit. Tool selection is formulated as a native tool-use decision rather than a separate retrieval stage. The selected operation and tool interface are subsequently passed to the Coder, which generates executable Blender code conditioned on the current scene state.

Blender Toolkit. The Blender Toolkit is implemented on top of Blender's native `bpy` API. Low-level Blender capabilities are encapsulated into structured callable actions with explicit names, parameter schemas, input types, target objects, return values, and error constraints. The Toolkit covers a broad range of modeling operations, including object creation and transformation, mesh and topology editing, curve and surface modeling, sculpting, and local deformation. This structured interface allows the Builder to reason directly over software-native modeling operations rather than repeatedly implementing low-level Blender APIs.

Iterative Refinement. We do not impose a fixed number of construction or refinement iterations. After the initial construction, the Critic evaluates the current 3D result and determines whether further refinement is required. When discrepancies are detected, the Critic provides targeted revision feedback to the Builder, which then plans the subsequent modeling operation. The process terminates automatically once the Critic determines that the current result sufficiently satisfies the target specification. Consequently, the number of refinement rounds is adaptively determined by the complexity and quality of each generated asset.

Active View Inspection. Geometric discrepancies can be strongly viewpoint-dependent. Therefore, during construction and diagnosis, the Critic is allowed to actively explore the current 3D asset rather than relying on a fixed set of observations. It first inspects the global structure and then autonomously adjusts the camera position, orientation, and zoom level to examine potentially problematic regions from more informative viewpoints. These actively selected views are used only for model diagnosis and revision, and are not used to obtain more favorable quantitative evaluation scores.

Baseline Adaptation. For the direct coding baselines, including **GPT-6-Astra**, **DeepSeek-V4**, and **claude-fable-5.1**, the corresponding model directly generates Blender code from the reference image without using the Builder, Blender Toolkit, or Critic.

For **VIGA**, we preserve its original code–render–inspect interaction paradigm and active camera exploration mechanism, while adapting its underlying reasoning model to **GPT-6-Astra**. We therefore denote this configuration as **VIGA (adapted)**.

For **Code2Worlds**, we similarly use **GPT-6-Astra** as the underlying reasoning model. Since Code2Worlds is originally designed for text-conditioned world generation, GPT-6-Astra first converts each reference image into a structured scene description, which is then passed to the original object/environment dual-stream generation pipeline. We denote this configuration as **Code2Worlds (adapted)**.

Using the same GPT-6-Astra backbone for both adapted agentic baselines reduces performance differences caused by heterogeneous reasoning models and enables a more controlled comparison of the underlying 3D generation strategies.

We additionally include **Tripo 2.0** as an implicit 3D generation baseline. Tripo 2.0 receives the same reference images and directly generates 3D assets. The generated assets are subsequently imported into Blender and rendered using the same quantitative evaluation protocol without manual geometry modification.

Evaluation Tasks. We evaluate all methods on two tasks: **object-level generation** and **scene-level generation**. Object-level generation evaluates the ability to reconstruct individual 3D objects from reference images, while scene-level generation further requires multi-object construction, spatial arrangement, and overall scene organization. We evaluate object-level generation on **BlenderObject-250** and scene-level generation on **BlenderScene-40**.

Evaluation Protocol. Active camera exploration is used only during construction and Critic-based diagnosis. For quantitative evaluation, all methods are rendered from the same predefined evaluation viewpoints under an identical rendering configuration. This separation prevents adaptive camera selection from introducing evaluation bias.

We report three automatic evaluation metrics: **Photometric Loss (PL)**, **Negative CLIP Score (N-CLIP)**, and **VLM-Score**. PL measures pixel-level discrepancy between the rendered result and the target image, with lower values indicating better alignment. N-CLIP measures high-level visual-semantic discrepancy, where lower values are better. VLM-Score evaluates task completion, spatial accuracy, detail accuracy, and overall visual quality using a vision-language evaluator, with higher values indicating better performance.

Token Accounting. We report provider-reported token consumption over the complete generation trajectory as a proxy for inference workload. Token usage is accumulated across model calls involved in initial construction, visual inspection, diagnosis, code generation, and subsequent refinement. Input tokens, cached-input tokens, and output tokens are recorded separately whenever they are reported by the corresponding API, while their aggregate is used for the main token statistics.

Because different model providers use different tokenizers, caching mechanisms, and pricing policies, raw token counts should not be interpreted as a strictly model-independent measure of computational or monetary cost. Instead, they provide an approximate measure of inference workload and are particularly informative when comparing different strategies under the same backbone. Token consumption is reported in millions (**M**) for readability.

Component Ablation. Component ablations are conducted only on the **scene-level generation** task using **GPT-6-Astra** as the fixed backbone. We compare four configurations: (1) the Base Model, which directly generates Blender code; (2) Base Model + Critic, where direct generation is followed by iterative Critic-based diagnosis and refinement; (3) Base Model + Builder & Toolkit, where the Builder performs construction planning and tool use in a single construction pass without iterative Critic refinement; and (4) the complete BlenderBuilder.

We additionally separate token consumption into the initial construction stage and subsequent refinement stages. This analysis allows us to study whether investing additional inference in a higher-quality initial construction can reduce the cost of later iterative revisions.

Evolution Experiment. The evolution experiment is conducted on the **scene-level generation** task. We evaluate increasingly capable GPT-family backbones, including **GPT-5**, **GPT-5.6**, and **GPT-6-Astra**. For each backbone, we compare direct modeling with the corresponding BlenderBuilder configuration under the same evaluation protocol. We use PL and N-CLIP to analyze whether the benefit of BlenderBuilder persists and increases as the underlying foundation model becomes stronger.

Human Evaluation. We further conduct a human evaluation to assess the practical quality of the generated 3D assets. We recruit **30 participants** and divide them into **five groups of six**. Each group consists of **two graduate students with more than two years of Blender experience** and **four undergraduate students with approximately six months of Blender experience**.

Each group evaluates the complete **BlenderObject-250** and **BlenderScene-40**. Participants are allowed to freely rotate, zoom, and inspect each 3D asset from multiple viewpoints while referring to the corresponding target image. Each sample is independently rated on a 1–5 Likert scale along three dimensions: **Quality**, **Realism**, and **Fidelity**. Quality evaluates geometric completeness and modeling quality; Realism evaluates whether the generated shapes, proportions, and local structures appear natural and plausible; and Fidelity evaluates consistency with the target input.

Ratings are first averaged within each evaluation group and subsequently aggregated across groups. We additionally compute the **Intraclass Correlation Coefficient (ICC)** for each criterion to assess agreement across independent evaluator groups. Detailed scoring rubrics are provided in the following section. “‘

C Human Evaluation Protocol and Rubric

C.1 Participants and Evaluation Protocol

We recruit 30 participants and divide them into five independent evaluation groups, with six participants per group. Each group consists of two graduate students with more than two years of Blender experience and four undergraduate students with approximately six months of Blender experience. This setup allows us to include both experienced users and participants with basic familiarity with Blender.

Each evaluation group assesses the complete **BlenderScene-40** and **BlenderObject-250** benchmarks, including **40 scene-level assets** and **250 object-level assets**. For each sample, participants are allowed to freely rotate, zoom, and inspect the generated 3D asset from multiple viewpoints while referring to the corresponding input text or reference image. All samples are anonymized and presented in randomized order to reduce potential bias from method identity or presentation order.

Participants independently rate each asset on a **1–5 Likert scale** along three dimensions: **Quality**, **Realism**, and **Fidelity**. For each asset, we first average the ratings from the six participants within each evaluation group and then aggregate the results across the five groups to obtain the final human evaluation scores. We additionally compute the **Intraclass Correlation Coefficient (ICC)** for each criterion to assess the agreement across independent evaluator groups.

C.2 RUBRIC

Quality measures the overall modeling quality of the generated 3D asset, including geometric completeness, structural correctness, surface quality, and visible modeling artifacts.

Score	Criterion
1	Severe geometric errors, missing components, broken structures, or clearly unacceptable modeling defects.
2	The basic structure is recognizable, but substantial errors, rough regions, or incomplete structures remain.
3	The overall structure is reasonable, but several noticeable local defects or missing details remain.
4	The geometry is largely complete and structurally sound, with only minor visible defects.
5	The geometry is complete, clean, and detailed, with almost no noticeable modeling defects.

Table 3: Human evaluation rubric for **Quality**.

Realism measures whether the generated asset appears visually natural and plausible, particularly in terms of shape, proportion, and local geometric details.

Score	Criterion
1	The asset appears clearly unnatural, with severely distorted shapes or proportions.
2	The target object is recognizable, but obvious unnatural proportions, surfaces, or structures remain.
3	The asset is generally plausible, but some proportions, shapes, or local details still appear artificial.
4	Most shapes, proportions, and details appear natural and convincing, with only minor realism issues.
5	The asset appears highly natural and visually convincing, approaching the quality of professionally modeled 3D assets.

Table 4: Human evaluation rubric for **Realism**.

Fidelity measures the consistency between the generated 3D asset and the target input, including overall appearance, geometric structure, component composition, and key details.

Score	Criterion
1	The generated asset differs substantially from the target, with major structural or semantic mismatches.
2	Some target characteristics are preserved, but several important structures or key details are incorrect.
3	The asset is broadly consistent with the target, but noticeable differences remain in structure, proportion, or details.
4	The asset closely matches the target in overall structure and major details, with only minor local discrepancies.
5	The asset is highly consistent with the target in overall appearance, structure, proportions, and key details.

Table 5: Human evaluation rubric for **Fidelity**.

For each asset, the human evaluation is represented as

$$S = \left(S_{\text{Quality}}, S_{\text{Realism}}, S_{\text{Fidelity}}\right),$$

where

$$S_{\text{Quality}}, S_{\text{Realism}}, S_{\text{Fidelity}} \in \{1, 2, 3, 4, 5\}.$$

The three criteria capture complementary aspects of the generated assets: **Quality** evaluates whether the asset is well modeled, **Realism** evaluates whether it appears natural and plausible, and **Fidelity** evaluates whether it faithfully matches the target input. Final results are reported by averaging ratings across participants and evaluation groups, together with ICC scores to assess inter-group consistency.