HoloCode: A Code-Centric Multi-Agent Framework for Image-to-3D Scene Generation

Hanlin Chen[1]* **Chung-Ching Lin**[2] **Yuyang Zhao**[1] **Dongyue Lu**[1] **Zhengyuan Yang**[2]
Xiaofei Wang[2] **Kevin Lin**[2] **Zhendong Wang**[2] **Yu Yang**[1] **Lijuan Wang**[2]
[1]National University of Singapore [2]Microsoft

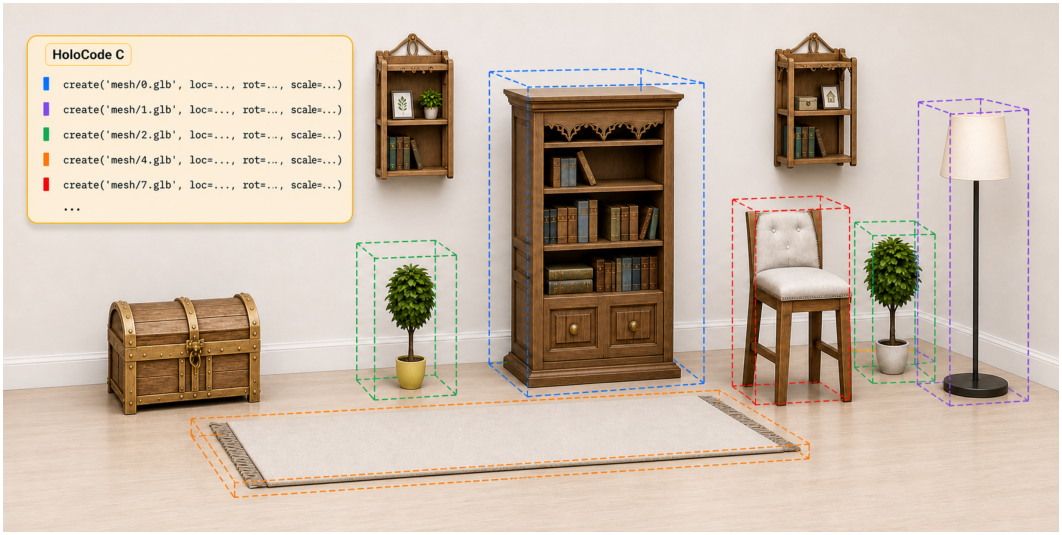

Figure 1: *HoloCode* generates 3D scenes from an image by predicting executable, object-centric Blender Python code. Each `create` command imports a mesh asset and sets its 3D transforms; colored markers and dashed cuboids indicate the correspondence between code and editable objects.

Abstract

Image-to-3D scene generation must recover both object geometry and plausible spatial relations. A common diffusion-based pipeline, exemplified by SAM 3D, generates each object separately and assembles the results. This preserves local details, but the lack of strong scene-level constraints often causes floating objects, interpenetrations, missing contacts, or implausible relative scales. Inspired by human designers who plan a layout before placing assets, we introduce HoloCode, a code-centric multi-agent framework. Given a single image and 2D localization cues, a vision-language model (VLM)-based Arranger Agent predicts executable Blender Python layout code with object identities and 3D transforms. A Modeler Agent binds each object to geometry through asset retrieval or image-to-3D object generation, and a Refiner Agent renders the scene and edits the code from visual feedback. The code representation exposes object names, positions, rotations, and scales, making generated scenes editable and interpretable. We train the code-generating agents using Blender Python datasets built from SAGE and 3D-FRONT, then further optimize them with GRPO rewards for executability and spatial alignment. Experiments in in-domain and cross-domain settings show that *HoloCode* improves scene-level plausibility while preserving object fidelity. Project page is available at https://holocode3d.github.io/.

*This work was done during Hanlin Chen's internship at Microsoft. Contact: hanlin.chen@u.nus.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Image-to-3D scene generation aims to lift a single RGB image into a complete 3D scene for AR/VR content creation, simulation, game and film production, and digital-twin construction [49, 11, 43, 9, 17, 28, 54]. Compared with single-object image-to-3D generation [45, 46, 22, 68, 39, 57, 60], the scene setting is substantially harder. A complete scene must be consistent at several coupled levels: object shape and appearance should match the image, while the layout must explain scale, depth, support, contact, and occlusion evidence in the same camera view. For example, a chair can look locally plausible as an isolated mesh, yet the scene still fails if the chair floats above the floor, penetrates a table, faces an inconsistent direction, or has a scale that contradicts nearby objects.

Recent progress *in single-image 3D object generation* has been largely driven by object-centric priors, including multiview diffusion, large reconstruction models, and 3D latent diffusion [45, 46, 22, 68, 39, 57, 60]. This success naturally suggests an object-wise route to scene generation: segment the image, generate each object independently, and assemble the resulting assets. SAM 3D [54] is representative of this strategy. It can preserve local geometry by optimizing each segmented object separately, but the cropped inputs weaken the evidence needed to infer relative depth, support, contact, and scale across objects, often leading to floating objects or interpenetrations as shown in Fig. 2. Such errors are not merely failures of isolated shape reconstruction; they arise from spatial variables that need to be reasoned about jointly at the scene level. This motivates a layout-first formulation that estimates object identities, positions, orientations, and scales before instantiating geometry. Making this layout explicit brings an additional advantage: after a scene is generated, users can directly adjust object placement, orientation, and scale, and the same variables provide a simple handle for fixing layout errors such as floating objects, interpenetrations, and incorrect relative scale. We therefore seek to *recover an explicit global layout over visible objects as editable scene variables*, making placement, support, contact, and relative scale directly controllable.

Following this formulation, we propose `HoloCode`, a **code-centric multi-agent framework** that first reasons about the 3D scene layout and then models each object within it. Unlike text-to-3D scene systems that expand prompts into objects, relations, assets, or tool calls and mainly target semantic consistency with language [14, 64, 71, 62, 58], our input is a single RGB image. The generated scene must therefore be *pixel-aligned*: the layout should match visible composition, camera-view geometry, and occlusion evidence, while object geometry and texture should follow image appearance. We use 2D cues accordingly, with bounding boxes for VLM-based layout reasoning and instance masks for object-centric retrieval and generation throughout the pipeline.

Given a single RGB image and ordered 2D boxes, our pipeline decouples the image-to-3D scene generation task into three stages: an **Arranger** Agent predicts an explicit 3D layout, a **Modeler** Agent instantiates each object within the predicted layout, and a **Refiner** Agent renders the scene and edits it to correct residual visual-spatial mismatches. The central representation in *HoloCode* is executable Blender Python code. The Arranger outputs a layout script whose core primitive is a `create(name, loc, rot, scale)` command. The semantic name defines an object placeholder, while `loc`, `rot`, and `scale` specify its 3D transform; this exposes scene layout as editable variables rather than as an implicit neural representation. In the second stage, the Modeler turns each placeholder into concrete geometry without changing the predicted layout. It first *queries* Objaverse [12] using object crops, multimodal retrieval, and re-ranking [38]; retrieval is preferred when the asset library contains a visually compatible object, but rare categories, unusual styles, or difficult viewpoints can make the best match unreliable. We therefore use retrieval confidence as a routing signal: low-confidence objects are passed to a *Diffuser* Agent that invokes image-to-3D object generators [60, 59, 54]. In both cases, the selected or generated mesh is inserted back into the same executable scene script under the Arranger-predicted transform, so object appearance can improve without relaxing layout constraints. In the final stage, the Refiner renders the current scene from the input viewpoint and edits the Blender Python code to correct residual placement, orientation, scale, support, and view-level mismatches. Unlike pipelines that treat generated geometry as a fixed output, *HoloCode* keeps the scene state editable throughout inference. This is especially useful because many scene errors are naturally code-local: lowering an object to contact the floor, rotating it toward the observed view, or resizing it to match nearby objects corresponds to small edits of `loc`, `rot`, and `scale`. Because object identities and transforms remain explicit, the generated scene can be parsed and edited directly.

Beyond standard SFT training of the code-generating agents, we further introduce GRPO with several rewards computed from parsed executable scripts. These **code-level reinforcement signals** directly



Figure 2: Object-wise SAM 3D optimization can yield floating objects and interpenetrations after assembly. *HoloCode* reasons over all objects in a shared layout, reducing such spatial conflicts.

reward valid programs and spatially accurate layouts through format executability, mean 3D IoU, and thresholded object accuracy during policy optimization.

To train and evaluate *HoloCode*, we construct **Blender Python datasets** from SAGE [58] and 3D-FRONT [15] annotations. We normalize scene annotations into a consistent camera-aligned Blender coordinate convention and serialize object layouts as deterministic sequences of `create` commands. The Arranger dataset supervises initial layout prediction, while the Refiner dataset starts from imperfect executable scripts, renders them from the input viewpoint, and supervises code edits that correct the visual and spatial mismatches. Experiments on in-domain and cross-domain settings show that *HoloCode* improves spatial plausibility while preserving object fidelity, especially when object-wise generation produces floating objects, interpenetrations, or inconsistent relative scales. In summary, our contributions are as follows:

- We introduce **HoloCode**, a **code-centric multi-agent framework** for image-to-3D scene generation that separates explicit VLM-based layout reasoning from object-level geometry instantiation, then uses render-and-edit refinement to improve scene consistency.
- We design an executable **Blender Python scene representation** shared by the Arranger, Modeler, and Refiner agents, exposing object identities, positions, rotations, and scales for inspection, rendering, scoring, and controllable editing.
- We propose **GRPO-based spatial and syntactic code alignment rewards** that optimize executable Blender Python code beyond token-level SFT.
- We also construct **two Blender Python code datasets** based on SAGE and 3D-FRONT, and demonstrate strong performance in both in-domain and cross-domain experiments.

2 Related Work

Image-to-3D Object and Scene Generation. Image-conditioned 3D scene generation spans several technical traditions. Supervised reconstruction methods recover room layout and object geometry from annotated indoor data [49, 11, 43, 9]. For multi-view surface reconstruction, NeuSG [4] uses Gaussian splatting guidance for neural implicit surfaces, while VCR-GauS [6] introduces view-consistent depth–normal regularization for Gaussian surfaces. CAD- and asset-based methods match image evidence to existing 3D models [29, 37, 19, 17]. More recent compositional pipelines combine reconstruction, retrieval, or generation modules to build scenes from object- or component-level predictions [70, 10, 21, 13, 65]. At the object level, multiview diffusion, large reconstruction models, and 3D latent diffusion provide increasingly strong generative priors [45, 46, 22, 68, 39, 57, 60], motivating recent scene systems that move beyond fixed CAD libraries. In this generative setting, scene-level pipelines can be roughly grouped into two strategies. One generates or reconstructs object instances separately and then assembles the resulting meshes [26, 54]; this leverages mature object generators, but makes the final layout depend on post-hoc assembly. The other optimizes multiple objects through a joint scene representation or shared latent space [10, 21, 65]. MIDI [28] and iScene [41] further use global scene tokens to coordinate object tokens, improving cross-object reasoning while still relying on compressed scene summaries. In action-controlled video generation, Robust Dreamer [7] maintains 3D consistency across autoregressive rollouts using latent Gaussian memory and deviation-aware training. For single-image 3D scene generation, *HoloCode* first predicts an explicit executable layout over detected objects, and then binds each object to a retrieved asset from Objaverse [12] or a generated mesh [60, 59, 54].

Structured 3D Scene Reasoning. 3D scene understanding research has shown the value of explicit object identities and spatial relations for language-grounded localization, question answering, detection, and layout recovery [3, 2, 56, 27, 42, 31, 23, 8, 72, 16]. ChatSplat [5] learns a Gaussian

language field for object-, view-, and scene-level conversation. Structured generative representations extend this idea to reconstruction and layout prediction. For example, SceneScript [1] formulates scene reconstruction as sequence prediction, and SpatialLM [48] predicts architectural elements and oriented object boxes from point clouds. *HoloCode* follows this structured direction in the single-image generation setting. Its Arranger Agent predicts Blender Python layout code in which each object is represented by an executable semantic name, position, rotation, and scale. Because this representation can be parsed, rendered, edited, and scored, spatial reasoning becomes an explicit part of the generation loop rather than an implicit byproduct of reconstruction.

LLM/VLM-Driven 3D Generation and Editing. LLMs have become useful planners, programmers, and tool controllers for 3D content creation. Shape-program and CAD-oriented methods generate executable or editable object programs [32, 35, 33, 34, 30, 36, 61]. At the scene level, systems such as LayoutGPT [14], Holodeck [64], GALA3D [71], SceneWeaver [62], SAGE [58], and LL3M [47] use LLMs or agents to convert text prompts into layouts, assets, tool calls, and iterative refinements. These systems primarily study text-conditioned scene creation, where success is measured by semantic consistency with the prompt. *HoloCode* instead uses VLM-based visual grounding from a single RGB image. Blender Python code serves as the shared state across agents, so code execution, visual verification, editability, and interpretability are built into the generation process from the beginning.

Code-Centric Agents and Reasoning Supervision. SceneCraft [25] synthesizes Blender scene programs from text, while Code-as-Room [63] uses top-down room images and CAD-Coder [18] generates individual CAD objects. Spatially contextualized VLMs [44] maintain a scene portrait, labeled point cloud, and hypergraph. VIGA [66] is a training-free image-to-program agent with interleaved rendering and inspection; we directly compare its adapted foreground-object reconstruction with our learned layout pipeline. Our CoT supervision follows precedents in scratchpads [50], rationale bootstrapping [67], and distillation [24]; ReFT [55] similarly combines CoT-SFT with subsequent reinforcement learning.

3 Methodology

This section describes **HoloCode**, shown in Fig. 3. Following the layout-first formulation in Sec. 1, our method uses an *executable Blender Python scene state* shared by three stages: the **Arranger** predicts editable global layout variables, the **Modeler** binds retrieved or generated geometry to each layout placeholder, and the **Refiner** corrects residual spatial mismatches through render-and-edit feedback. Sec. 3.1 describes the agent workflow, Sec. 3.2 introduces the Blender Python code datasets used for supervision, and Sec. 3.3 gives the SFT and GRPO objectives of our method in detail.

3.1 Code-Centric Scene Agents

We formulate the input as a single RGB image $\boldsymbol{I}$ with visible object instance masks $\boldsymbol{M} = \{\boldsymbol{m}_i\}_{i=1}^{N}$, following common image-to-3D scene generation settings [54, 28]. From each mask, we derive a 2D bounding box $\boldsymbol{b}_i$ and collect the ordered set $\boldsymbol{B} = \{\boldsymbol{b}_i\}_{i=1}^{N}$. The pipeline then maps $(\boldsymbol{I}, \boldsymbol{M}, \boldsymbol{B})$ to a sequence of *executable Blender Python scene scripts*: an initial layout script C_a, a mesh-bound script C_m, and a refined script C_r. Boxes are used by the Arranger because they provide compact object localization for VLM-based, pixel-aligned layout reasoning, while masks are retained to produce object crops for retrieval and generation in the Modeler.

3.1.1 `Arranger`: scene-level layout programming

In our framework, the first step is to establish a *shared 3D scaffold* before choosing any detailed meshes. This avoids committing to object geometry before the system has resolved global placement, relative scale, contact, and coarse orientation. The **Arranger** Agent therefore acts as a layout planner: it lifts the image and ordered 2D boxes into an executable scene-level placement script whose variables should match the visible composition, camera-view geometry, and occlusion evidence.

$$C_a = \mathcal{A}_{\text{arr}}(\boldsymbol{I}, \boldsymbol{B}), \tag{1}$$

where C_a is a Blender Python layout script. Rather than modeling detailed geometry, C_a specifies one placement command per object:

```
create(name="...", loc=..., rot=..., scale=...).
```
(2)

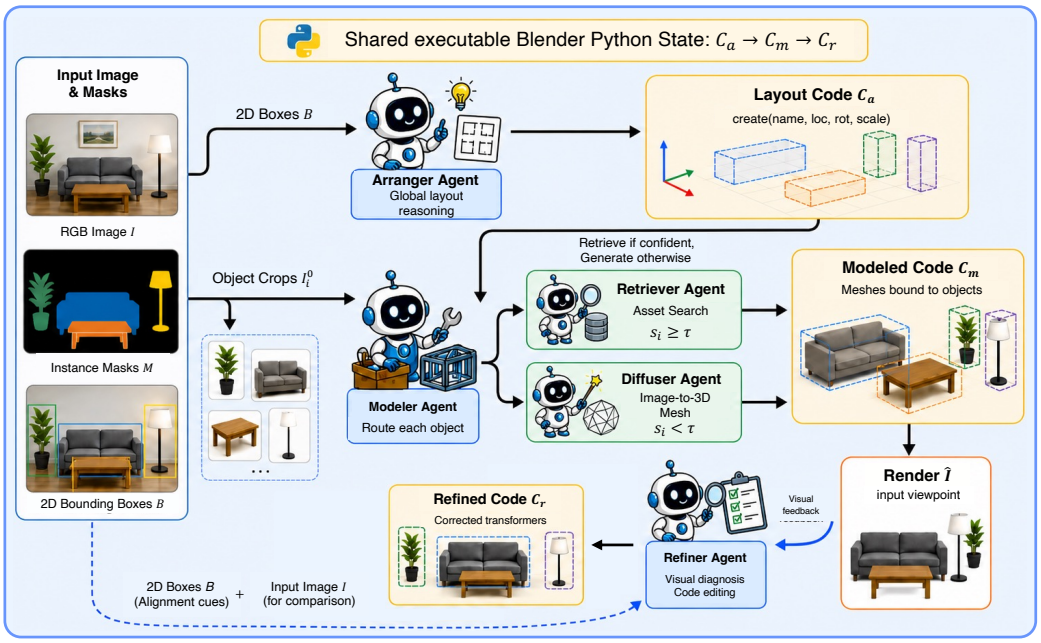

Figure 3: **Overview of the *HoloCode* multi-agent inference pipeline.** Given an image $\boldsymbol{I}$ and instance masks $\boldsymbol{M}$, the system derives boxes $\boldsymbol{B}$ for layout reasoning and object crops for modeling. The **Arranger** infers executable layout code C_a with object identities and 3D transforms; the **Modeler** binds each object to geometry, using *retrieval* for high-confidence asset matches and *diffusion* for low-confidence or out-of-library objects, yielding C_m. The **Refiner** renders C_m, compares it with the input view, and edits the Blender Python code into C_r to correct residual spatial mismatches.

Here, `name` defines the object placeholder, and `loc`, `rot`, and `scale` denote its 3D translation, orientation, and size in the shared scene coordinate frame. During layout prediction, these fields expose the scene layout as editable variables; after mesh binding, the predefined `create` interface resolves the attached object identifier or mesh path, imports the corresponding `.glb` asset, and applies the predicted transform. The same interface can also return parsed object attributes in an evaluation mode without modifying the scene, allowing generated code to be inspected or scored directly. Thus, C_a grounds each 2D object instance as an *executable 3D placement* while deliberately leaving mesh selection to the next stage. Appendix C describes the implementation details of this interface.

3.1.2 `Modeler`: layout-preserving geometry binding

Given C_a, the scene already specifies what objects should appear and where they should be placed, but it lacks concrete geometry. The **Modeler** Agent fills this gap by preserving the layout attributes in C_a while binding each object to either an existing asset or a newly generated mesh. We parse C_a into object specifications $\mathcal{O} = \{o_i\}_{i=1}^{N}$, where each o_i contains the semantic name and transform parameters from its `create` command. The output is a modeled script C_m in which every placement from C_a is associated with a concrete mesh. The Modeler serves as a *routing controller*: it first attempts retrieval for efficient high-fidelity asset reuse, and falls back to diffusion-based generation when retrieval confidence indicates that the asset library does not provide a reliable match.

Retriever: high-confidence asset reuse. The Retriever Agent queries Objaverse [12] to reuse curated 3D assets when possible. For each object, we crop an object-centric query image $\boldsymbol{I}_i^o$ from $\boldsymbol{I}$ using its instance mask in $\boldsymbol{M}$. Offline, each mesh asset is represented by visual embeddings of rendered or collected object views, computed with Qwen3-VL-Embedding [38]. At inference time, the same embedding model encodes $\boldsymbol{I}_i^o$. We first compute cosine similarity between the query embedding and all asset embeddings to obtain the top-K candidates, and then re-rank these candidates with a trained reranker [38]. The highest-ranked compatible mesh is inserted into the corresponding object specification o_i with the `loc`, `rot`, and `scale` predicted by the Arranger Agent. In this way, retrieval improves object fidelity without weakening the global layout constraints already encoded in C_a.

Diffuser: fallback mesh generation. Retrieval is reliable for common objects covered by the asset library, but it may fail for rare categories, unusual styles, or viewpoints. We therefore use

retrieval confidence as a routing signal. Let s_i be the highest cosine similarity between the query embedding of $\boldsymbol{I}_i^o$ and the asset database. If $s_i < \tau$, the retrieved candidates are treated as unreliable, and the object is routed to the Diffuser Agent. The Diffuser Agent invokes image-to-3D object generation models [60, 59, 54] conditioned on $\boldsymbol{I}_i^o$ to synthesize a mesh. The generated mesh is then handled identically to a retrieved asset: it is linked to o_i and written into the modeled script C_m using the Arranger-predicted transform. This *retrieval-before-generation* design keeps common objects efficient and high quality while covering objects outside the asset library.

3.1.3 `Refiner`: render-guided code refinement

Although C_m is executable, its rendered view may still disagree with the input image due to floating objects, missing support contacts, incorrect scale, or placement errors. The **Refiner** Agent addresses these residual visual-spatial failures by *closing the loop between code execution and visual feedback*. We render C_m from the same camera viewpoint as the input image to obtain $\hat{\boldsymbol{I}}$, and then predict

$$C_r = \mathcal{A}_{\text{ref}}(\boldsymbol{I}, \boldsymbol{B}, C_m, \hat{\boldsymbol{I}}), \tag{3}$$

where C_r is the refined Blender Python script. Guided by $\boldsymbol{B}$, the agent compares $\boldsymbol{I}$ with $\hat{\boldsymbol{I}}$ and edits object locations, rotations, scales, and scene parameters to correct support, contact, and view-level mismatches while preserving the bound meshes.

Prior multi-agent frameworks [47, 58] often use a critic to diagnose visual errors and a separate coding agent to apply edits. We merge these roles into the Refiner Agent with Chain-of-Thought (CoT) reasoning. Internally, the agent analyzes the current code and render, identifies visual mismatches, and derives *concrete code-level edits*; externally, it outputs only the final refined script C_r. This unified design reduces inter-agent communication while retaining explicit visual diagnosis.

3.2 Executable Code Supervision

Training code-generating agents requires target scripts in a *consistent spatial convention*. We therefore build **Blender Python code datasets** from SAGE [58] and 3D-FRONT [15] annotations and normalize all annotations before serialization. This dataset design makes executable code serve both as the intermediate representation introduced above and as the *direct supervision target*. Because the source datasets can differ in scene scale and coordinate systems, we run SAM 3D [54] and use its predicted scene scale as the common reference. After scale alignment, annotations are transformed into the camera coordinate system and then expressed in Blender coordinates, using a z-up, $-y$-forward, and x-left convention. This representation makes object transforms directly executable and keeps rendering, asset insertion, and refinement in the same coordinate frame. Based on the normalized annotations, we construct two datasets that correspond to the two learned code generators. The **Arranger dataset** supervises initial layout prediction from $(\boldsymbol{I}, \boldsymbol{B})$, while the **Refiner dataset** supervises code revision from an imperfect script and its rendered feedback.

Arranger Dataset: ordered layout code. For each scene, we convert every annotated object into the attributes required by `create`: position to `loc`, rotation to `rot`, and object size to `scale`. Objects are serialized into a *deterministic sequence* of `create` commands following the order of the input boxes, yielding the ground-truth layout code C_a^*. This ordering provides a direct correspondence between each 2D box and each generated command. Thus, the Arranger is supervised directly in the executable code format that it will produce at inference time, without any format conversion.

Refiner Dataset: render-conditioned code repair. The Refiner dataset adds *rendered feedback* and CoT supervision. For each training image, we first obtain a layout prediction from the trained Arranger Agent and bind meshes as in the Modeler stage. We then amplify its deviations from the reference layout to form a perturbed executable scene script $\tilde{C}_m$, which mimics imperfect inference-time code. After rendering $\tilde{C}_m$ from the input viewpoint to obtain $\tilde{\boldsymbol{I}}$, we use GPT-5 [53] to annotate a CoT trace conditioned on the input image, rendered image $\tilde{\boldsymbol{I}}$, bounding boxes, perturbed script, and correction hints from the script difference. The supervision target contains this trace, which links visual mismatches to edits of object parameters, followed by the corrected Blender Python code C_r^*.

3.3 Training Code Agents

The trainable components in *HoloCode* are the two code-generating agents: the **Arranger**, which predicts the initial layout script, and the **Refiner**, which edits an imperfect script from rendered feedback. The Retriever and Diffuser operate inside the Modeler as tool modules for asset reuse and fallback mesh generation, so they do not require code-generation supervision. This focuses learning on the stages where code quality directly controls spatial plausibility: *initial layout generation* and *render-conditioned correction*. We train these agents in two stages: SFT teaches the executable code format, and GRPO optimizes task-level spatial alignment.

SFT for supervised code imitation. We first train both agents as conditional code generators. The Arranger input is $x_a = (\boldsymbol{I}, \boldsymbol{B})$ with target C_a^*, while the Refiner input is $x_r = (\boldsymbol{I}, \boldsymbol{B}, \tilde{C}_m, \tilde{\boldsymbol{I}})$ with target C_r^*. Let x denote either input, and let $C^* = (c_1^*, \ldots, c_T^*)$ be its target code sequence. We apply supervised fine-tuning with the next-token objective:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(x,C^*)\sim\mathcal{D}} \left[\sum_{t=1}^{T} \log \pi_\theta(c_t^* \mid x, c_{<t}^*) \right]. \tag{4}$$

SFT teaches the target code format, but its token-level objective does not directly measure the spatial quality of the executed scene. We therefore further apply GRPO with rewards computed from parsed executable code and the resulting 3D layout.

GRPO for executable spatial alignment. GRPO turns the executable code representation into a direct scene-level training signal. To score a sampled script, we parse its `create` calls and match predicted objects to references using the deterministic order/name correspondence. We evaluate the executed layout with **spatial rewards**: [1]*mean 3D IoU* between oriented boxes derived from `loc`, `rot`, and `scale`, together with [2]*Accuracy@0.25* and [3]*Accuracy@0.50*, which measure the fraction of objects whose 3D IoU exceeds the corresponding threshold. Missing or invalid objects receive zero spatial score. To preserve syntactic code alignment and executability, we also use a **format reward**, defined as the fraction of candidate `create` calls that can be parsed and executed by our restricted evaluator. The final reward r_i combines spatial alignment and executability terms.

Starting from the SFT policy, GRPO optimizes the agents without an additional value model. For each input x, we sample G candidate code completions $\{C_i\}_{i=1}^G$ from the old policy $\pi_{\theta_{\mathrm{old}}}$, compute rewards $\{r_i\}_{i=1}^G$, and convert them into a *group-normalized advantage*:

$$A_i = \frac{r_i - \mathrm{mean}(\{r_1, \ldots, r_G\})}{\mathrm{std}(\{r_1, \ldots, r_G\}) + \delta}. \tag{5}$$

We maximize the clipped GRPO objective with a KL penalty to the SFT reference policy π_{ref}:

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) \quad = \quad \mathbb{E}\left[\frac{1}{G} \sum_{i=1}^{G} \min\Big(\rho_i A_i,\ \mathrm{clip}(\rho_i, 1-\epsilon, 1+\epsilon) A_i \Big) - \beta\, D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}}) \right], \tag{6}$$

and minimize $\mathcal{L}_{\mathrm{GRPO}}(\theta) = -\mathcal{J}_{\mathrm{GRPO}}(\theta)$. Here, $\rho_i = \pi_\theta(C_i \mid x)/\pi_{\theta_{\mathrm{old}}}(C_i \mid x)$ is the sequence-level probability ratio, while ϵ, β, and δ control clipping, KL regularization, and advantage normalization. This optimization encourages scripts that are both *executable* and *spatially aligned* with the annotated 3D scene, improving object transforms while preserving valid code structure.

4 Experiments

In this section, we conduct comprehensive experiments to evaluate the effectiveness of **HoloCode**. We first introduce the experimental settings in Sec. 4.1. Next, we present quantitative and qualitative comparisons with representative baselines in Sec. 4.2. Finally, we analyze the contribution of each component in Sec. 4.3. Together, these experiments examine both the spatial accuracy of the generated layouts and the quality of the resulting 3D scenes.

4.1 Experimental Settings

We summarize the evaluation metrics used throughout the paper here, and provide detailed metric and reward definitions, implementation details, and dataset settings in the supplementary material (Secs. E and A). For **layout estimation**, we parse the generated `create` commands and compare the resulting

Table 1: Quantitative comparison of *3D layout estimation* on **SAGE** and **3D-FRONT**.

Dataset	Method	RATE↓ (%)	AOE↓ (Deg)	RSE↓ (%)	mIoU↑	Acc@0.25↑ (%)	Acc@0.50↑ (%)
SAGE	*Qwen3.5*	40.83	107.00	46.74	0.116	18.29	3.81
	Gemini 3.1 Pro	31.96	106.40	43.06	0.161	26.80	6.99
	GPT-5	53.71	74.74	26.10	0.147	25.28	10.20
	SpatialLM	52.17	152.30	77.35	0.134	26.20	1.92
	VIGA	22.74	58.46	18.32	0.318	55.76	31.24
	Ours	**8.51**	**32.33**	**8.26**	**0.490**	**82.04**	**53.86**
3D-FRONT	*Qwen3.5*	21.25	107.30	29.43	0.203	36.44	12.26
	Gemini 3.1 Pro	17.08	107.70	25.61	0.296	54.86	24.51
	GPT-5	18.21	78.91	28.15	0.280	51.65	26.03
	SpatialLM	31.17	140.00	69.92	0.197	39.86	7.69
	VIGA	10.62	45.37	15.72	0.524	77.45	56.71
	Ours	**2.74**	**10.08**	**5.83**	**0.733**	**94.31**	**84.03**

Table 2: **Quantitative comparison on 3D-FRONT**. CD and F-Score are evaluated on the *scene geometry*, while mIoU measures *object bounding-box overlap*.

Method	CD↓	F-Score (%)↑	mIoU↑
PanoRecon	0.142	42.10	0.262
Total3D	0.252	34.25	0.257
InstPIFu	0.131	41.36	0.323
SSR	0.133	41.08	0.334
DiffCAD	0.117	45.06	0.414
Gen3DSR	0.121	42.02	0.386
REPARO	0.125	43.15	0.365
SAM 3D	0.088	60.55	0.466
MIDI	0.116	53.08	0.612
Ours	**0.071**	**63.67**	**0.733**

oriented 3D boxes against reference layouts. We report [1]RATE, which measures the relative position error between predicted and ground-truth object centers; [2]AOE, which reports the angular orientation error in degrees; [3]RSE, which measures the relative scale error; [4]mIoU, which computes the mean 3D IoU over matched objects; and [5]Acc@0.25 and [6]Acc@0.50, which measure the fraction of objects whose 3D IoU exceeds 0.25 and 0.50, respectively. For **3D scene generation**, following existing scene reconstruction protocols [49, 70, 28], we evaluate the generated scene geometry with [1]CD and [2]F-Score using the default threshold of 0.1. We further report [3]3D-mIoU between predicted and ground-truth object bounding boxes to assess the accuracy of the generated spatial arrangement.

4.2 Comparison to State of the Arts

3D Layout Estimation. We compare with LLM-based baselines, including Qwen3.5-397B [52], Gemini 3.1 Pro [51], GPT-5 [53], SpatialLM [48], and VIGA [66]; baseline preprocessing and alignment details are provided in Appendix A. Table 1 shows that *HoloCode* consistently achieves the best layout accuracy on both SAGE and 3D-FRONT. It obtains 0.490 mIoU and 53.86% Acc@0.50 on SAGE, and further reaches 0.733 mIoU and 84.03% Acc@0.50 on 3D-FRONT, with low RATE, AOE, and RSE. Qualitatively, Fig. 4 shows that baselines often recover object categories but scatter them with wrong depth, orientation, or scale, while *HoloCode* better preserves object grouping, support relations, and global scene structure.

VIGA receives the same RGB image and ordered textual boxes and is adapted to reconstruct only the specified target objects, excluding background geometry. We evaluate all 4,000 SAGE, 1,000 3D-FRONT, and 500 OOD test inputs. HoloCode improves on VIGA's 0.318/0.524/0.251 mIoU to 0.490/0.733/0.323, supporting task-specific supervision and spatial optimization over this training-free agent (Appendix F).

Image-to-3D Scene Generation. We compare with representative reconstruction/generation baselines, including PanoRecon [11], Total3D [49], MIDI [28], SAM 3D [54], and others. Table 2 shows

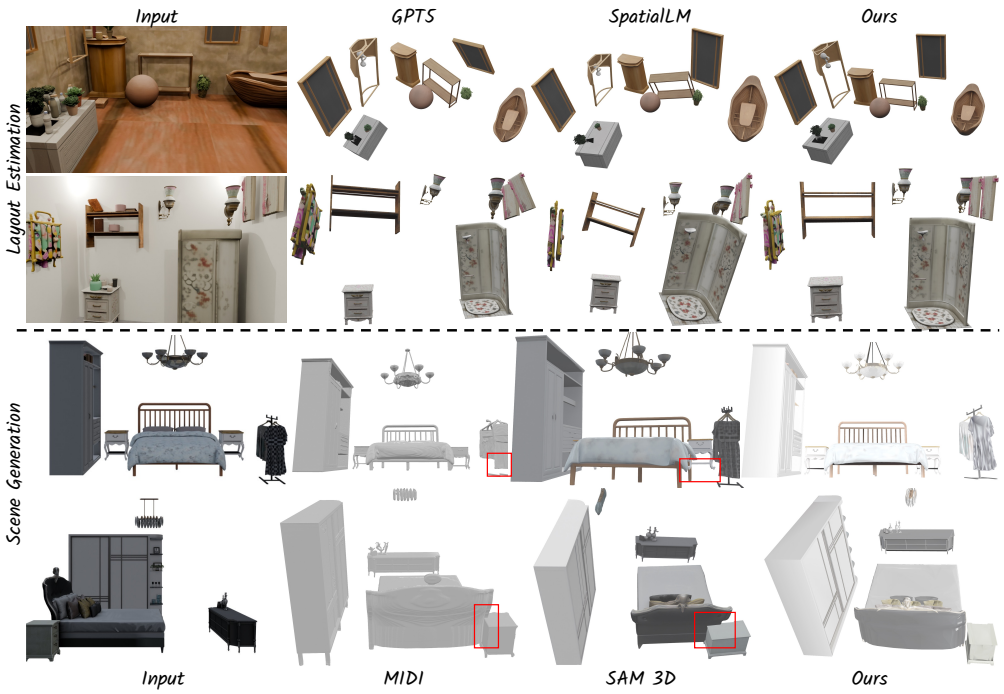

Figure 4: **Qualitative visualization on SAGE and 3D-FRONT.** We show results for both 3D layout estimation and image-to-3D scene generation, illustrating that *HoloCode* recovers coherent object layouts and produces assembled 3D scenes from a single input image.

Table 3: **Out-of-distribution (OOD) quantitative comparison.** The top block evaluates *3D layout estimation*, while the bottom block evaluates *3D scene generation*.

3D Layout Estimation

Method	RATE↓ (%)	AOE↓ (Deg)	RSE↓ (%)	mIoU↑	Acc@0.25↑ (%)	Acc@0.50↑ (%)
Qwen3.5	48.04	95.58	37.59	0.135	20.68	7.73
Gemini 3.1 Pro	37.20	115.8	48.16	0.178	29.96	9.28
GPT-5	45.97	95.60	36.80	0.153	23.95	9.21
SpatialLM	44.99	146.3	83.70	0.109	15.85	2.11
VIGA	24.26	93.71	30.42	0.251	47.62	13.76
Ours	**11.08**	**92.19**	**24.17**	**0.323**	**65.38**	**18.20**

3D Scene Generation

Metric	*Gen3DSR*	*DiffCAD*	*SAM 3D*	*MIDI*	**Ours**
CD↓	0.205	0.213	0.153	0.191	**0.137**
F-Score (%)↑	21.36	20.84	28.92	24.70	**33.26**
mIoU↑	0.176	0.169	0.217	0.251	**0.323**

HoloCode achieves the best CD, F-Score, and mIoU, reaching 0.071 CD and 0.733 mIoU. Under domain shift, Table 3 shows the same trend: on real-image OOD inputs, scene-generation mIoU improves from MIDI's 0.251 to 0.323 against the manually verified pseudo references. The best OOD layout metrics further indicate that executable layout code generalizes beyond the training domains under this evaluation protocol. Fig. 4 shows key failure modes of recent baselines. SAM 3D reconstructs objects independently before assembly, preserving local appearance but lacking scene-level constraints; objects can float, interpenetrate, or have inconsistent relative scales. MIDI performs joint scene optimization, but limited per-object resolution can degrade mesh details, such as the missing legs of the clothes rack. In contrast, *HoloCode* binds each object to an explicit executable layout, yielding more coherent spatial arrangements while retaining object-level geometry.

Code-Centric 3D Scene Editing. The executable representation enables localized 3D scene editing. Since each object is represented by a `create` command with explicit `loc`, `rot`, and `scale`, an

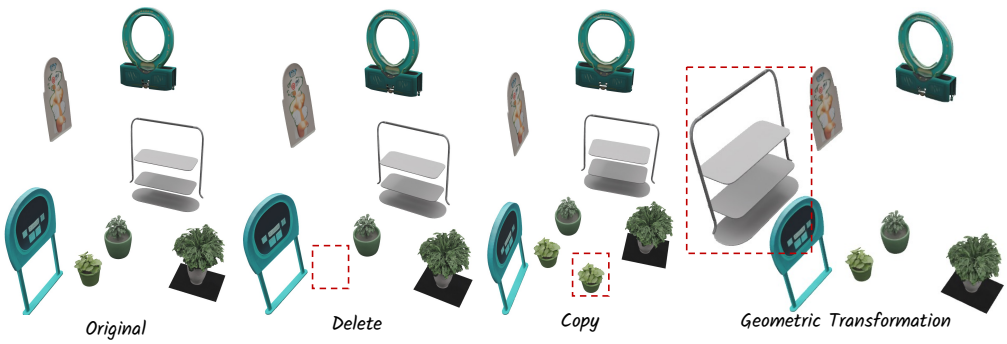

Figure 5: **Code-centric 3D scene editing visualization.** Starting from the original generated scene, *HoloCode* supports localized edits such as object deletion, object copying, and geometric transformation by directly modifying the executable Blender Python scene representation.

Table 4: **Ablation study on 3D layout estimation on SAGE and 3D-FRONT**. Rows within each dataset successively add GRPO optimization and render-conditioned refinement to the SFT-only Arranger.

Dataset	Stage	RATE↓	AOE↓	RSE↓	mIoU↑	Acc@.25↑	Acc@.50↑
SAGE	SFT	11.57	32.51	10.94	0.409	71.13	39.42
	+ GRPO	9.72	**32.24**	9.72	0.445	76.95	45.21
	+ Refiner	**8.51**	32.33	**8.26**	**0.490**	**82.04**	**53.86**
3D-FRONT	SFT	3.24	11.41	6.32	0.725	93.04	82.64
	+ GRPO	2.77	10.12	5.86	**0.733**	94.27	84.02
	+ Refiner	**2.74**	**10.08**	**5.83**	**0.733**	**94.31**	**84.03**

RATE, RSE, and Acc are percentages; AOE is in degrees.

LLM can delete, copy, move, rotate, or resize target objects by editing corresponding code lines while preserving the rest of the program. Fig. 5 shows representative delete, copy, and geometric-transformation edits that preserve unrelated objects.

4.3 Ablation Study

Training and refinement. Table 4 ablates the two learned components of *HoloCode*: **GRPO optimization for the Arranger Agent** and **render-conditioned correction by the Refiner Agent**. For each dataset, we compare an SFT-only Arranger, a GRPO-optimized Arranger, and the full pipeline with the Refiner enabled. We additionally isolate the input cue and output representation through controlled SAGE-only comparisons of boxes versus masks and Blender Python versus JSON (Appendix F.2).

GRPO optimization consistently improves the SFT-only Arranger across both datasets, reducing `RATE/RSE` and increasing `mIoU/Acc@0.50`; this shows that task-level spatial rewards are more effective than token-level imitation alone for executable layout code. **Refiner correction** further improves layout overlap and thresholded accuracy, especially on SAGE, by correcting render-visible spatial errors. The Refiner increases SAGE `mIoU` from 0.445 to 0.490. The gains on 3D-FRONT are smaller because the GRPO Arranger already reaches 0.733 `mIoU`, leaving limited room for further refinement.

5 Conclusion

We presented *HoloCode*, a code-centric multi-agent framework for image-to-3D scene generation. It uses Blender Python code to separate scene-level layout reasoning from object-level geometry instantiation. We constructed Blender Python datasets from SAGE and 3D-FRONT, and further optimized the agents with GRPO rewards for executable code and spatial alignment. Experiments show that *HoloCode* improves spatial plausibility while preserving object fidelity. These results show the value of executable code for controllable 3D scene generation.

References

[1] Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan P. Frost, Luke Holland, Campbell Orme, Jakob Engel, Edward Miller, Richard A. Newcombe, and Vasileios Balntas. SceneScript: Reconstructing scenes with an autoregressive structured language model. In *European Conference on Computer Vision*, pages 247–263, 2024.

[2] Daichi Azuma, Taiki Miyanishi, Shuhei Kurita, and Motoaki Kawanabe. ScanQA: 3d question answering for spatial scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19107–19117, 2022.

[3] Dave Zhenyu Chen, Angel X. Chang, and Matthias Nießner. ScanRefer: 3d object localization in rgb-d scans using natural language. In *European Conference on Computer Vision*, pages 202–221, 2020.

[4] Hanlin Chen, Chen Li, Yunsong Wang, and Gim Hee Lee. NeuSG: Neural implicit surface reconstruction with 3D gaussian splatting guidance. *arXiv preprint arXiv:2312.00846*, 2023.

[5] Hanlin Chen, Fangyin Wei, and Gim Hee Lee. ChatSplat: 3D conversational gaussian splatting. *arXiv preprint arXiv:2412.00734*, 2024.

[6] Hanlin Chen, Fangyin Wei, Chen Li, Tianxin Huang, Yunsong Wang, and Gim Hee Lee. VCR-GauS: View consistent depth-normal regularizer for gaussian surface reconstruction. In *Advances in Neural Information Processing Systems*, volume 37, pages 139725–139750, 2024.

[7] Hanlin Chen, Jiaxin Wei, Xibin Song, Yifu Wang, Steve Wang, Hongdong Li, Pan Ji, and Gim Hee Lee. Robust Dreamer: Deviation-aware latent gaussian memory for action-controlled AR video generation. In *Advances in Neural Information Processing Systems*, 2026.

[8] Sijin Chen, Xin Chen, Chi Zhang, Mingsheng Li, Gang Yu, Hao Fei, Hongyuan Zhu, Jiayuan Fan, and Tao Chen. LL3DA: Visual interactive instruction tuning for omni-3d understanding, reasoning, and planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26418–26428, 2024.

[9] Yixin Chen, Junfeng Ni, Nan Jiang, Yaowei Zhang, Yixin Zhu, and Siyuan Huang. Single-view 3d scene reconstruction with high-fidelity shape and texture. In *International Conference on 3D Vision*, pages 1456–1467, 2024.

[10] Yongwei Chen, Tengfei Wang, Tong Wu, Xingang Pan, Kui Jia, and Ziwei Liu. Comboverse: Compositional 3d assets creation using spatially-aware diffusion guidance. *arXiv preprint arXiv:2403.12409*, 2024.

[11] Manuel Dahnert, Ji Hou, Matthias Nießner, and Angela Dai. Panoptic 3d scene reconstruction from a single rgb image. *Advances in Neural Information Processing Systems*, 34:8282–8293, 2021.

[12] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.

[13] Andreea Dogaru, Mert Özer, and Bernhard Egger. Generalizable 3d scene reconstruction via divide and conquer from a single view. *arXiv preprint arXiv:2404.03421*, 2024.

[14] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36:18225–18250, 2023.

[15] Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942, 2021.

[16] Rao Fu, Jingyu Liu, Xilun Chen, Yixin Nie, and Wenhan Xiong. Scene-LLM: Extending language model for 3d visual reasoning. In *Winter Conference on Applications of Computer Vision*, pages 2195–2206, 2025.

[17] Daoyi Gao, Dávid Rozenberszki, Stefan Leutenegger, and Angela Dai. Diffcad: Weakly-supervised probabilistic cad model retrieval and alignment from an rgb image. *ACM Transactions on Graphics*, 43(4):1–15, 2024.

[18] Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-Coder: Text-to-CAD generation with chain-of-thought and geometric reward. In *Advances in Neural Information Processing Systems*, volume 38, pages 59765–59789, 2025.

[19] Can Gümeli, Angela Dai, and Matthias Nießner. Roca: Robust cad model retrieval and alignment from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4022–4031, 2022.

[20] Agrim Gupta, Piotr Dollár, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5356–5364, 2019.

[21] Haonan Han, Rui Yang, Huan Liao, Jiankai Xing, Zunnan Xu, Xiaoming Yu, Junwei Zha, Xiu Li, and Wanhua Li. Reparo: Compositional 3d assets generation with differentiable 3d layout alignment. *arXiv preprint arXiv:2405.18525*, 2024.

[22] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. Lrm: Large reconstruction model for single image to 3d. *arXiv preprint arXiv:2311.04400*, 2023.

[23] Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3D-LLM: Injecting the 3d world into large language models. In *Advances in Neural Information Processing Systems*, pages 20482–20494, 2023.

[24] Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017. Association for Computational Linguistics, 2023.

[25] 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 scene as blender code. *arXiv preprint arXiv:2403.01248*, 2024.

[26] Binbin Huang, Haobin Duan, Yiqun Zhao, Zibo Zhao, Yi Ma, and Shenghua Gao. Cupid: Generative 3d reconstruction via joint object and pose modeling. In *CVPR*, 2025.

[27] Haifeng Huang, Yilun Chen, Zehan Wang, Rongjie Huang, Runsen Xu, Tai Wang, Luping Liu, Xize Cheng, Yang Zhao, Jiangmiao Pang, and Zhou Zhao. Chat-Scene: Bridging 3d scene and large language models with object identifiers. In *Advances in Neural Information Processing Systems*, pages 113991–114017, 2024.

[28] Zehuan Huang, Yuan-Chen Guo, Xingqiao An, Yunhan Yang, Yangguang Li, Zi-Xin Zou, Ding Liang, Xihui Liu, Yan-Pei Cao, and Lu Sheng. Midi: Multi-instance diffusion for single image to 3d scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 23646–23657, 2025.

[29] Hamid Izadinia, Qi Shan, and Steven M Seitz. Im2cad. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5134–5143, 2017.

[30] Pradeep Kumar Jayaraman, J. Lambourne, Nishkrit Desai, Karl D. D. Willis, Aditya Sanghi, and Nigel Morris. Solidgen: An autoregressive model for direct b-rep synthesis. *arXiv preprint arXiv:2203.13944*, 2022.

[31] Baoxiong Jia, Yixin Chen, Huangyue Yu, Yan Wang, Xuesong Niu, Tengyu Liu, Qing Li, and Siyuan Huang. SceneVerse: Scaling 3d vision-language learning for grounded scene understanding. In *European Conference on Computer Vision*, pages 289–310, 2024.

[32] R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics*, 39(6):1–20, 2020.

[33] R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics*, 42(4):1–17, 2023.

[34] R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapelib: Designing a library of procedural 3d shape abstractions with large language models. *arXiv preprint arXiv:2502.08884*, 2025.

[35] R. Kenny Jones, Homer Walke, and Daniel Ritchie. Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9871–9880, 2022.

[36] Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4713–4722, 2024.

[37] Weicheng Kuo, Anelia Angelova, Tsung-Yi Lin, and Angela Dai. Mask2cad: 3d shape prediction by learning to segment and retrieve. In *European Conference on Computer Vision*, pages 260–277, 2020.

[38] Mingxin Li, Yanzhao Zhang, Dingkun Long, Keqin Chen, Sibo Song, Shuai Bai, Zhibo Yang, Pengjun Xie, An Yang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Qwen3-vl-embedding and qwen3-vl-reranker: A unified framework for state-of-the-art multimodal retrieval and ranking. *arXiv*, 2026.

[39] Weiyu Li, Jiarui Liu, Rui Chen, Yixun Liang, Xuelin Chen, Ping Tan, and Xiaoxiao Long. Craftsman: High-fidelity mesh generation with 3d native generation and interactive geometry refiner. *arXiv preprint arXiv:2405.14979*, 2024.

[40] Haotong Lin, Sili Chen, Jun Hao Liew, Donny Y. Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025.

[41] Lu Ling, Yunhao Ge, Yichen Sheng, and Aniket Bera. I-scene: 3d instance models are implicit generalizable spatial learners. *arXiv preprint arXiv:2512.13683*, 2025.

[42] Dingning Liu, Xiaoshui Huang, Yuenan Hou, Zhihui Wang, Zhenfei Yin, Yongshun Gong, Peng Gao, and Wanli Ouyang. Uni3D-LLM: Unifying point cloud perception, generation and editing with large language models. *arXiv preprint arXiv:2402.03327*, 2024.

[43] Haolin Liu, Yujian Zheng, Guanying Chen, Shuguang Cui, and Xiaoguang Han. Towards high-fidelity single-view holistic reconstruction of indoor scenes. In *European Conference on Computer Vision*, pages 429–446, 2022.

[44] Xinhang Liu, Yu-Wing Tai, and Chi-Keung Tang. Agentic 3D scene generation with spatially contextualized VLMs. *arXiv preprint arXiv:2505.20129*, 2025.

[45] Yuan Liu, Cheng Lin, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. Syncdreamer: Generating multiview-consistent images from a single-view image. *arXiv preprint arXiv:2309.03453*, 2023.

[46] Xiaoxiao Long, Yuan-Chen Guo, Cheng Lin, Yuan Liu, Zhiyang Dou, Lingjie Liu, Yuexin Ma, Song-Hai Zhang, Marc Habermann, Christian Theobalt, et al. Wonder3d: Single image to 3d using cross-domain diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9970–9980, 2024.

[47] 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.

[48] Yongsen Mao, Junhao Zhong, Chuan Fang, Jia Zheng, Rui Tang, Hao Zhu, Ping Tan, and Zihan Zhou. Spatiallm: Training large language models for structured indoor modeling. In *Advances in Neural Information Processing Systems*, 2025.

[49] Yinyu Nie, Xiaoguang Han, Shihui Guo, Yujian Zheng, Jian Chang, and Jian Jun Zhang. Total3dunderstanding: Joint layout, object pose and mesh reconstruction for indoor scenes from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 55–64, 2020.

[50] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[51] Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. A new era of intelligence with gemini 3, November 2025.

[52] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026.

[53] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[54] SAM 3D Team, Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, Aohan Lin, Jiawei Liu, Ziqi Ma, Anushka Sagar, Bowen Song, Xiaodong Wang, Jianing Yang, Bowen Zhang, Piotr Dollár, Georgia Gkioxari, Matt Feiszli, and Jitendra Malik. Sam 3d: 3dfy anything in images. *arXiv preprint arXiv:2511.16624*, 2025.

[55] Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. ReFT: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

[56] Zehan Wang, Haifeng Huang, Yang Zhao, Ziang Zhang, Tao Jin, and Zhou Zhao. Data-efficiently learn large language model for universal 3d scene perception. In *NAACL*, pages 313–333, 2025.

[57] Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Jingxi Xu, Philip Torr, Xun Cao, and Yao Yao. Direct3d: Scalable image-to-3d generation via 3d latent diffusion transformer. *arXiv preprint arXiv:2405.14832*, 2024.

[58] Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. *arXiv preprint arXiv:2602.10116*, 2026.

[59] Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and compact structured latents for 3d generation. *Tech report*, 2025.

[60] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. *arXiv preprint arXiv:2412.01506*, 2024.

[61] Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm. *arXiv preprint arXiv:2411.04954*, 2024.

[62] Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *arXiv preprint arXiv:2509.20414*, 2025.

[63] Yixuan Yang, Zhen Luo, Wanshui Gan, Jinkun Hao, Junru Lu, Jinghao Yan, Zhaoyang Lyu, and Xudong Xu. Code-as-Room: Generating 3D rooms from top-down view images via agentic code synthesis. *arXiv preprint arXiv:2605.18451*, 2026.

[64] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024.

[65] Kaixin Yao, Longwen Zhang, Xinhao Yan, Yan Zeng, Qixuan Zhang, Lan Xu, Wei Yang, Jiayuan Gu, and Jingyi Yu. Cast: Component-aligned 3d scene reconstruction from an rgb image. *ACM Transactions on Graphics*, 44(4):1–19, 2025.

[66] 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. *arXiv preprint arXiv:2601.11109*, 2026.

[67] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[68] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics*, 43(4):1–20, 2024.

[69] Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. Swift:a scalable lightweight infrastructure for fine-tuning, 2024.

[70] Junsheng Zhou, Yu-Shen Liu, and Zhizhong Han. Zero-shot scene reconstruction from single images with deep prior assembly. In *Advances in Neural Information Processing Systems*, 2024.

[71] Xiaoyu Zhou, Xingjian Ran, Yajiao Xiong, Jinlin He, Zhiwei Lin, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. Gala3d: Towards text-to-3d complex scene generation via layout-guided generative gaussian splatting. *arXiv preprint arXiv:2402.07207*, 2024.

[72] Chenming Zhu, Tai Wang, Wenwei Zhang, Kai Chen, and Xihui Liu. ScanReason: Empowering 3d visual grounding with reasoning capabilities. In *European Conference on Computer Vision*, pages 151–168, 2024.

A Implementation Details

We implement the trainable code-generating agents in `ms-swift` [69], using Qwen3.5-9B-Base [52] as the backbone. Training follows the two-stage optimization pipeline described in Sec. 3.3. Table 5 summarizes the main training, inference, and Modeler routing hyperparameters.

In the SFT stage, we perform full-parameter fine-tuning on the combined SAGE [58] and 3D-FRONT [15] code data. In the GRPO stage, we initialize from the SFT checkpoint and continue full-parameter training with the spatial-overlap and format rewards defined in Sec. E. For both stages, we freeze the vision encoder and multimodal aligner. All training experiments are conducted on a cluster of 32 NVIDIA H20 GPUs. Under this setup, training the Arranger Agent takes approximately 1.5 days, while training the Refiner Agent takes approximately 2.5 days. The longer training time for the Refiner Agent is mainly due to its render-conditioned input, which includes both the original image and the rendered feedback image and therefore contains more visual tokens than the single-image Arranger input. For the Modeler Agent routing in Sec. 3.1.2, objects with retrieval similarity below the threshold are routed to the Diffuser Agent. Unless otherwise specified, we use Trellis 2 [59] as the image-to-3D generation model for the Diffuser Agent.

Baseline Evaluation Protocol. We convert all layout-estimation baselines to the same parsed object representation used by *HoloCode*. The VLM baselines are prompted to output one `create`

Table 5: Implementation hyperparameters for the trainable code-generating agents. Unless otherwise specified, the Arranger and Refiner agents use the same settings.

Stage	Parameter	Value
General	Training framework	`ms-swift`
General	Backbone	Qwen3.5-9B-Base
General	Trainable modules	Full-parameter agent tuning with frozen vision encoder and multimodal aligner
General	Hardware	32 NVIDIA H20 GPUs
SFT	Training data	Combined SAGE and 3D-FRONT code data
SFT	Epochs	10
SFT	Precision and memory	bfloat16, FlashAttention, gradient checkpointing, group-by-length batching, DeepSpeed ZeRO-2
SFT	Sequence/image token budget	2048 / 2048
SFT	Per-device batch/grad. accum.	4 / 4
SFT	Learning rate/warmup	1×10^{-5} / 0.05
SFT	NEFTune noise	$\alpha = 5$
SFT	Effective batch size	512
GRPO	Initialization/epochs	SFT checkpoint / 1
GRPO	Completions per prompt	$G = 32$
GRPO	Generation batch size	32
GRPO	Max completion length	2048
GRPO	KL coefficient	$\beta = 0.04$
GRPO	Per-device batch/grad. accum.	1 / 1
GRPO	Learning rate	1×10^{-6}
GRPO	Reward weights	3D IoU: 1.0; Acc@0.25: 1.0; Acc@0.50: 1.0; format: 0.1
GRPO	Precision and memory	bfloat16, FlashAttention, gradient checkpointing, DeepSpeed ZeRO-2 offload
GRPO	Rollout engine	Co-located vLLM with prefix caching
Inference	Backend	vLLM
Inference	Max lengths	Model: 4096; generation: 2048
Inference	Repetition penalty/batch size	1.05 / 8
Modeler	Retrieval threshold	$\tau = 0.1$
Modeler	Diffuser model	Trellis 2



Figure 6: Out-of-domain visualization results demonstrating the generalization capability of the proposed method on unseen samples.

command per input box in the box order, and invalid or missing commands are scored as failures by the evaluator. SpatialLM [48] requires a point cloud input, so for each single-image test sample we first estimate dense depth with Depth Anything 3 [40], back-project the depth map with the dataset camera intrinsics into a camera-frame point cloud, and then run SpatialLM on the resulting point cloud. For images without calibrated intrinsics, we use the same centered pinhole-camera convention as the rendering and evaluation pipeline.

Since different baselines may use different global scale and coordinate conventions, we apply a scene-level alignment before computing layout metrics. For each baseline prediction, we first estimate a single isotropic scale from the matched object centers and sizes to match the reference scene extent. After scaling, we run ICP on matched object centers to estimate a rigid transform, and apply this transform to all predicted object centers and orientations; object sizes are affected only by the global scale. This alignment normalizes coordinate-frame differences but does not change object identities or relative layout structure. Missing, hallucinated, or unmatched objects remain in the denominator and are penalized as described in Sec. E.

B Datasets, Agent Prompts, and Refiner CoT Annotation

Datasets. Following the Blender Python code dataset construction in Sec. 3.2, we build two task-specific datasets for training the Arranger and Refiner agents. The Arranger dataset supervises layout-code prediction from the input image and 2D object boxes, while the Refiner dataset supervises render-conditioned code correction from imperfect scene scripts. Both datasets are constructed from SAGE [58] and 3D-FRONT [15]. For SAGE, we use its 10K generated scenes and render each scene from 10 different viewpoints, yielding 81K training views and 4K test views after filtering and splitting. For 3D-FRONT, we start from the MIDI [28]-processed image data and extract ground-truth object layouts from the official 3D-FRONT annotations, which are then serialized into Blender Python code. Following the MIDI evaluation protocol, we use the same 1K views for testing and use the remaining approximately 70K views for training. To further assess cross-domain generalization on real-world scenes, we evaluate on LVIS [20]. Since LVIS does not provide 3D scene ground truth, we first run SAM 3D [54] to obtain pseudo 3D layouts, then manually filter unreliable reconstructions and retain 500 scene images for testing.

Both trainable agents are supervised in a chat-style format with image inputs and a code response. The prompts are intentionally compact: the Arranger Agent receives only the original image and mask-derived 2D boxes, while the Refiner Agent additionally receives the rendered feedback image and the current executable script. In both cases, the target output is Blender Python code expressed through the shared `create` interface.

Arranger Agent Prompt. The Arranger Agent is trained to map the input image and ordered bounding boxes to an initial layout script. Its system prompt asks the model to act as a Blender Python scene-reconstruction expert and to return only executable code. The user prompt contains the image token, the number of detected objects, and one COCO-format box `[x,y,w,h]` per object. The object index in this box list defines the order of the corresponding `create` calls in the assistant target.

```
System:
You are an expert in 3D scene reconstruction using the Blender Python API.
Provide only the executable Python code block without any markdown formatting or explanation.

User:
Analyze the input image <image> and generate the corresponding Blender Python layout code.
Use the data below to position the {num_objects} objects detected in the scene:
Object 0 bbox: [x, y, w, h]
Object 1 bbox: [x, y, w, h]
...

Assistant:
create(name=..., loc=..., rot=..., scale=..., ...)
create(name=..., loc=..., rot=..., scale=..., ...)
...
```

This prompt makes 2D localization explicit without exposing 3D labels in the input. The assistant response is the serialized ground-truth layout code, where each object command specifies the semantic placeholder or object name together with its 3D location, rotation, and scale.

Refiner Agent Prompt. The Refiner Agent is trained as a render-and-edit code agent. Its input contains two images: Image 1 is the original scene image, and Image 2 is the render produced by the current script `synthetic.py`. The user prompt also includes the same mask-derived bounding boxes and the full current script. The system prompt asks the model to output the corrected Blender Python code after a thinking block, and the training assistant message is formatted as a CoT trace followed by the corrected code.

````
System:
You are an expert Blender Python refine agent.
Output the corrected Blender Python code after the thinking block.

User:
You are given two images and one Blender Python script.
Image 1 is the original input image for the code agent.
Image 2 is the render produced from synthetic.py.
Refine synthetic.py into the corrected Blender Python code.
Ignore color, texture, material, and lighting.
Only refine object position, rotation, and scale.
Use bbox order as object order.

Scene: {scene}
Mask-derived bboxes:
[
  {"index": 0, "bbox": [x, y, w, h]},
  ...
]

synthetic.py:
```python
...
```

Assistant:
<think>
Object 0: ...
Object 1: ...
...
</think>
```python
corrected create(...) calls
```
````

During parsing and evaluation, the reasoning block is removed and only the executable code is scored. Thus, the CoT is used as supervised intermediate reasoning for the Refiner Agent rather than as part of the final scene representation.

CoT Annotation for the Refiner Agent. We annotate Refiner CoT traces with GPT-5 [53]. The annotation prompt is different from the final SFT prompt: it provides GPT-5 with the agent-visible inputs above, plus teacher-only correction hints that are used only to make the label accurate. For each object, the visible payload contains its index, 2D box, name or description, and current `create` line. The teacher-only payload contains per-field actions for `loc`, `rot`, and `scale`, numeric deltas from the synthetic script to the corrected script, and direction hints in the shared coordinate convention. The default action rules mark `loc` as unchanged when RATE is below 0.05, `rot` as unchanged when AOE is below $5°$, and `scale` as unchanged when RSE is below 0.05; otherwise the corresponding field is labeled for modification. Numeric correction deltas are rounded to three decimals.

The labeling prompt requires GPT-5 to return strict JSON with a multi-line `thinking` string and a `per_object_actions` list. The generated thinking must not cite the teacher-only reference, ground-truth code, thresholds, or metrics. We also reject labels that leak teacher-only terms such as `rate`, `aoe`, `rse`, `ground truth`, `target`, or `answer key`, or that use shortcut answer-derived phrasing such as "from A to B" and "set the field to value". This filtering keeps the trace in the form of visual diagnosis rather than direct answer copying.

Refiner CoT Logic. The intended reasoning chain is object-wise and follows the same order as the boxes and `create` calls. For each object, the trace first compares the rendered object against the original image and its box, identifying whether the object appears too far left or right, too far forward or backward, too high or low, too large or small, or incorrectly oriented. It then states a keep-or-modify decision for `loc`, `rot`, and `scale`. Finally, it maps the visual mismatch to code-level edits: increasing `x` moves an object left, decreasing `x` moves it right, decreasing `y` moves it forward, increasing `y` moves it backward, and increasing or decreasing `z` moves it up or down. Rotation edits are described as component deltas tied to visible facing or tilt errors, and scale edits are described as size deltas tied to the mismatch between the render and the original box. Fields that remain unchanged must also be justified by visual agreement. The resulting SFT target therefore teaches the Refiner Agent to bridge three steps: visual comparison, field-level edit decision, and executable Blender Python correction.

C The `create` Interface

The `create` function is the executable primitive shared by the Arranger, Modeler, and Refiner agents. Its signature exposes only the variables that the code-generating agents need to control: `name` identifies the object or mesh file, `loc` gives the 3D translation, `rot` gives the object orientation, and `scale` gives the object size. Optional fields such as `obj_name` and `des` support object selection from a `.blend` file and description-level bookkeeping. This compact interface keeps the generated layout code close to the evaluation representation while still being directly executable in Blender after the Modeler Agent binds each placeholder to a concrete retrieved or generated asset.

In evaluation mode, `create(..., eval=True)` returns the parsed object tuple without modifying the Blender scene, which allows the same generated script to be used for rewards and metrics. In execution mode, the function first resolves the mesh path: absolute paths are used directly, explicit relative paths are interpreted from the current working directory, and bare asset names are resolved under the configured mesh directory. It then imports the asset with Blender's native importers for common formats such as `.obj`, `.fbx`, `.ply`, `.glb`, and `.gltf`. After import, it applies the predicted transform by setting `location`, broadcasting scalar `scale` values when necessary, converting either a 3D axis-angle vector or 4D quaternion into Blender's `(w,x,y,z)` quaternion convention, and assigning the result to `rotation_quaternion`. The function therefore serves as both a restricted parsing target for training and reward computation and a concrete scene-construction API for inference.

D More Experiments

D.1 Ablation Study of Reward Function Design

Table 6 compares different reward combinations where all variants are trained and evaluated only on the SAGE dataset. All GRPO reward variants substantially improve over the SFT baseline, confirming that task-level spatial rewards are important for layout-code optimization. The full reward

```python
import bpy
import os
from pathlib import Path

def create(name, loc=(0, 0, 0), rot=(0, 0, 0), scale=None,
           obj_name=None, des=None, eval=False):
    if eval:
        return name, loc, rot, scale, des

    path_obj = Path(name)
    if path_obj.is_absolute():
        filepath = str(path_obj)
    elif os.path.sep in name or "/" in name:
        filepath = str(path_obj)
    else:
        filepath = str(Path(base_mesh_dir) / name)

    bpy.ops.object.select_all(action="DESELECT")
    obj = None

    if filepath.endswith(".blend"):
        with bpy.data.libraries.load(filepath, link=False) as (data_from, data_to):
            if obj_name and obj_name in data_from.objects:
                data_to.objects = [obj_name]
            elif data_from.objects:
                data_to.objects = [data_from.objects[0]]
        if data_to.objects:
            obj = data_to.objects[0]
            bpy.context.scene.collection.objects.link(obj)
    elif filepath.endswith(".obj"):
        bpy.ops.wm.obj_import(filepath=filepath)
    elif filepath.endswith(".fbx"):
        bpy.ops.import_scene.fbx(filepath=filepath)
    elif filepath.endswith(".ply"):
        bpy.ops.wm.ply_import(filepath=filepath)
    elif filepath.endswith(".glb") or filepath.endswith(".gltf"):
        bpy.ops.import_scene.gltf(filepath=filepath)

    if obj is None and bpy.context.selected_objects:
        obj = bpy.context.selected_objects[0]
    if obj is None:
        return None

    obj.location = loc
    if scale is not None:
        if isinstance(scale, (int, float)):
            scale = (float(scale),) * 3
        obj.scale = scale
    obj.rotation_mode = "QUATERNION"
    obj.rotation_quaternion = _resolve_rotation_to_quat(rot)
    return obj
```

Figure 7: Condensed implementation of the `create` interface used by *HoloCode*. The function either returns parsed object attributes for evaluation or imports a mesh asset and applies the agent-predicted transform for executable scene construction.

combination achieves the best mIoU and thresholded accuracies, while remaining close to the best position, orientation, and scale errors. Figure 8 further shows a representative reward trace during GRPO training: despite high-variance batch-level rewards, the smoothed reward rises rapidly in the early stage and remains stable near a high value. We therefore use the combined reward of 3D IoU, Acc@0.25, Acc@0.50, and format correctness in the main experiments.

D.2 Qualitative Analysis of the Refiner Agent

Figure 9 illustrates how the Refiner Agent improves an initial executable scene script. The Arranger prediction already recovers the main objects and approximate scene structure, but several objects remain misaligned in position, depth, or scale. By conditioning on both the original image and the rendered feedback, the Refiner Agent identifies these object-wise discrepancies and converts them into targeted code edits. The accompanying CoT trace provides an interpretable diagnosis of which fields should be modified or preserved, showing that render-guided refinement helps close the gap between the initial layout and the observed scene.

Table 6: **Ablation study of reward-function design on SAGE**. Each variant is trained and evaluated only on SAGE. The first row is the SFT baseline without GRPO rewards.

3D IoU	Acc@0.25	Acc@0.50	Format	RATE (%)↓	AOE (Deg)↓	RSE (%)↓	mIoU↑	Acc@0.25 (%)↑	Acc@0.50 (%)↑
				18.87	47.94	19.22	0.471	75.45	54.17
	√			7.56	29.95	7.96	0.536	86.05	61.57
√				8.03	30.48	8.44	0.529	85.33	60.50
			√	7.76	30.16	8.14	0.534	85.60	61.10
		√		7.35	29.91	7.80	0.539	86.41	61.91
	√		√	7.23	29.50	7.62	0.538	86.36	61.64
√		√		7.39	29.48	7.72	0.537	86.07	61.65
√			√	7.59	29.91	8.00	0.536	86.03	61.07
	√	√		7.29	29.64	7.62	0.538	86.37	61.81
√	√			7.53	29.92	7.87	0.536	86.19	61.43
		√	√	7.44	29.68	7.75	0.535	86.00	61.43
√	√		√	8.05	29.73	7.74	0.538	86.25	61.68
	√	√	√	7.35	29.79	7.75	0.538	86.16	61.72
√	√	√	√	7.24	29.63	7.67	0.540	86.59	61.95

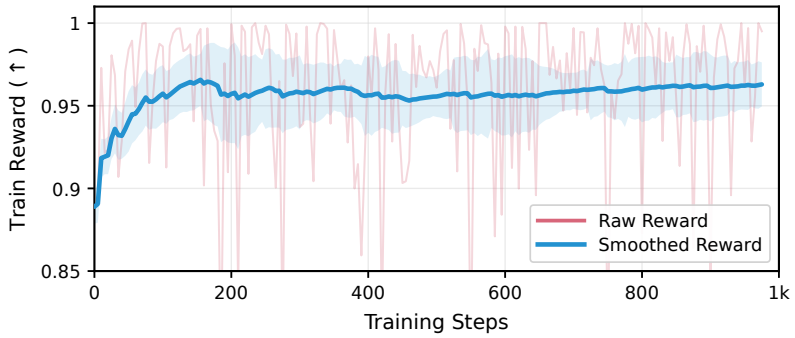

Figure 8: Training reward curve for the Acc@0.25 reward run. The translucent curve shows the raw logged reward values, and the solid curve shows the time-weighted EMA smoothed reward with smoothing set to 0.99.

D.3 Additional Out-of-Domain Qualitative Results

Figure 6 presents additional qualitative results on out-of-domain scenes. These examples include unfamiliar layouts, object appearances, textures, and imaging artifacts, testing whether the learned code agents transfer beyond the training distribution. Despite this domain shift, *HoloCode* preserves the overall scene arrangement and recovers plausible object geometry and placement, indicating that the executable scene representation remains effective under challenging visual conditions.

E Evaluation Metrics and Reward Functions

Both evaluation and reward computation operate on the parsed Blender Python layout code. We first remove reasoning blocks and keep executable `create(...)` calls. Each valid object is represented as $o_k = (n_k, \boldsymbol{t}_k, \boldsymbol{q}_k, \boldsymbol{s}_k, d_k)$, where n_k is the object matching key, $\boldsymbol{t}_k \in \mathbb{R}^3$ is the location, $\boldsymbol{q}_k$ is a unit quaternion, $\boldsymbol{s}_k$ is the scale, and d_k is the optional description. Rotations stored as Euler angles, axis-angle vectors, or 6D rotations are converted to quaternions before scoring. Scale values are broadcast to compatible dimensions and clipped to $[10^{-4}, 10]$ to avoid degenerate boxes. Let $K = \hat{K} \cup K^*$ be the union of predicted and ground-truth object keys. Missing objects, hallucinated extra objects, invalid rotations, or invalid scales remain in this denominator and receive failure scores.

3D Oriented Box IoU. For an object o_k, we convert $(\boldsymbol{t}_k, \boldsymbol{q}_k, \boldsymbol{s}_k)$ into an oriented 3D bounding box:

$$\mathcal{B}_k = \mathrm{conv}\left\{ \boldsymbol{t}_k + \boldsymbol{R}(\boldsymbol{q}_k)(\boldsymbol{s}_k \odot \boldsymbol{u}) \mid \boldsymbol{u} \in \left\{ -\frac{1}{2}, \frac{1}{2} \right\}^3 \right\}, \tag{7}$$



Figure 9: Qualitative visualization of the Refiner Agent. Given the original image and the initial Arranger prediction, the Refiner Agent compares the rendered feedback with the input scene, diagnoses object-level layout errors through CoT reasoning, and produces a corrected scene layout with improved spatial alignment.

where $\boldsymbol{R}(\boldsymbol{q}_k)$ is the rotation matrix induced by the quaternion. The per-object 3D IoU is

$$\mathrm{IoU}_k = \frac{\mathrm{Vol}(\hat{\mathcal{B}}_k \cap \mathcal{B}_k^*)}{\mathrm{Vol}(\hat{\mathcal{B}}_k \cup \mathcal{B}_k^*)}, \tag{8}$$

and is set to 0 for missing or invalid object pairs. We report mean 3D IoU and thresholded object accuracy as

$$\mathrm{mIoU} = \frac{1}{|K|} \sum_{k \in K} \mathrm{IoU}_k, \tag{9}$$

$$\mathrm{Acc@}\tau = \frac{1}{|K|} \sum_{k \in K} \mathbb{I}\left[\mathrm{IoU}_k \geq \tau\right], \quad \tau \in \{0.25, 0.50\}. \tag{10}$$

Translation, Rotation, and Scale Errors. We use relative translation error (RATE) to measure object-center accuracy:

$$\mathrm{RATE} = \frac{1}{|K|} \sum_{k \in K} \frac{\|\hat{\boldsymbol{t}}_k - \boldsymbol{t}_k^*\|_2}{d_k^t}, \quad d_k^t = \begin{cases} \|\boldsymbol{t}_k^*\|_2, & \|\boldsymbol{t}_k^*\|_2 > \epsilon, \\ 1, & \text{otherwise.} \end{cases} \tag{11}$$

The angular orientation error (AOE) is computed from the geodesic distance between unit quaternions:

$$e_k^{\mathrm{rot}} = \frac{180}{\pi} 2 \cos^{-1}\left(\left|\mathrm{clip}\left(\boldsymbol{q}_k^{*\top} \hat{\boldsymbol{q}}_k, -1, 1\right)\right|\right), \tag{12}$$

$$\mathrm{AOE} = \frac{1}{|K|} \sum_{k \in K} e_k^{\mathrm{rot}}. \tag{13}$$

The implementation also records an AOE histogram with 30° bins for diagnostic scene ranking. For scale, we report relative scale error (RSE):

$$\mathrm{RSE} = \frac{1}{|K|} \sum_{k \in K} \frac{\|\hat{\boldsymbol{s}}_k - \boldsymbol{s}_k^*\|_2}{d_k^s}, \quad d_k^s = \begin{cases} \|\boldsymbol{s}_k^*\|_2, & \|\boldsymbol{s}_k^*\|_2 > \epsilon, \\ 1, & \text{otherwise.} \end{cases} \tag{14}$$

For missing or invalid pairs, the evaluator assigns zero IoU and zero thresholded accuracy, 180° AOE, and failure values for relative translation and scale errors, so all parsing and object-coverage failures affect the final averages.

Scene-Level Diagnostic Score. The evaluation script also supports an optional GPT-5 scene score. Given a parsed predicted layout, the evaluator assembles and renders the scene from multiple views, provides the reference image and scene priors to GPT-5, and records a scalar scene-alignment score. This score is used only as an auxiliary diagnostic and scene-ranking metric; it is not used in the main quantitative tables or in GRPO training.

Reward Functions. Several reward functions reuse the metrics above, so we define them once and convert error metrics into bounded rewards. For any scalar error e, let $[x]_+ = \max(0, x)$. The increasing reward terms are

$$r_{\mathrm{IoU}} = \mathrm{mIoU}, \tag{15}$$

$$r_{\mathrm{Acc@}\tau} = \mathrm{Acc@}\tau, \quad \tau \in \{0.25, 0.50\}. \tag{16}$$

The error-based optional reward terms are

$$r_{\mathrm{RATE}} = [1 - \mathrm{RATE}]_+, \tag{17}$$

$$r_{\mathrm{RSE}} = [1 - \mathrm{RSE}]_+, \tag{18}$$

$$r_{\mathrm{AOE}} = \left[1 - \frac{\mathrm{AOE}}{180}\right]_+. \tag{19}$$

The plugin also implements a relative angular orientation reward. Let $\boldsymbol{q}_I = (1, 0, 0, 0)$ be the identity rotation and let

$$\theta_k^* = \frac{180}{\pi} 2 \cos^{-1}\left(\left|\boldsymbol{q}_k^{*\top} \boldsymbol{q}_I\right|\right). \tag{20}$$

The relative angular orientation error and reward are

$$\mathrm{RAOE} = \frac{1}{|K|} \sum_{k \in K} \frac{e_k^{\mathrm{rot}}}{d_k^\theta}, \quad d_k^\theta = \begin{cases} \theta_k^*, & \theta_k^* > \epsilon, \\ 180, & \text{otherwise}, \end{cases} \tag{21}$$

$$r_{\mathrm{RAOE}} = [1 - \mathrm{RAOE}]_+. \tag{22}$$

Format Reward and Final GRPO Reward. To encourage executable code, the format reward evaluates candidate `create*()` lines in a restricted environment:

$$r_{\mathrm{fmt}}(C) = \begin{cases} \frac{N_{\mathrm{succ}}(C)}{N_{\mathrm{cand}}(C)}, & N_{\mathrm{cand}}(C) > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{23}$$

where N_{cand} is the number of lines that look like `create*()` calls and N_{succ} is the number that parse and execute successfully. In the main GRPO experiments, we use the spatial rewards that directly match the reported overlap metrics plus the format reward:

$$r_{\mathrm{main}} = w_{\mathrm{IoU}} r_{\mathrm{IoU}} + w_{25} r_{\mathrm{Acc@0.25}} + w_{50} r_{\mathrm{Acc@0.50}} + w_{\mathrm{fmt}} r_{\mathrm{fmt}}, \tag{24}$$

with $w_{\mathrm{IoU}} = w_{25} = w_{50} = 1.0$ and $w_{\mathrm{fmt}} = 0.1$. The reward plugin further provides an optional normalized combined reward over IoU, RSE, RAOE, RATE, and format terms:

$$r_{\mathrm{comb}} = \frac{w_{\mathrm{IoU}} r_{\mathrm{IoU}} + w_{\mathrm{RSE}} r_{\mathrm{RSE}} + w_{\mathrm{RAOE}} r_{\mathrm{RAOE}} + w_{\mathrm{RATE}} r_{\mathrm{RATE}} + w_{\mathrm{fmt}} r_{\mathrm{fmt}}}{w_{\mathrm{IoU}} + w_{\mathrm{RSE}} + w_{\mathrm{RAOE}} + w_{\mathrm{RATE}} + w_{\mathrm{fmt}}}. \tag{25}$$

F Additional Evaluation and Controlled Ablations

F.1 VIGA Evaluation Protocol

The VIGA comparison uses all samples in the processed evaluation splits used throughout the paper: 4,000 SAGE test views, 1,000 3D-FRONT test views, and 500 manually verified LVIS-OOD images, totaling 5,500 samples without subsampling. VIGA and *HoloCode* receive the same RGB image and ordered textual 2D-box descriptions. We adapt VIGA's code and prompts to generate only the target objects specified by these boxes; background and environmental geometry are not generated or evaluated. VIGA is training-free, so this adaptation requires no retraining. Both methods use the same parsing, object matching, coordinate alignment, and layout metrics described in Secs. A and E. This comparison therefore evaluates foreground-object reconstruction and spatial arrangement. The results in Tables 1 and 3 assess the combined effects of image–code supervision, spatial GRPO, and learned refinement against a training-free agent; they do not isolate the output representation alone.

F.2 Controlled Cue and Representation Ablations

Input Cues and Output Representation. For the cue and representation ablations, we retrain on SAGE only, using the same backbone, data split, training schedule, spatial rewards, and evaluation protocol. The JSON variant contains identical object fields in the same order and uses a deterministic executor for the same Blender operations. These SAGE-only checkpoints are separate from the joint SAGE/3D-FRONT models in the main benchmark tables. Table 7 reports all six metrics.

Ordered 2D boxes improve mIoU from 0.521 to 0.540 compared with masks as layout cues. Their compact textual coordinates make object correspondence explicit in the output order; masks still

Table 7: Full cue and output-representation ablation. All variants are trained and evaluated on SAGE only.

Cue	Output	RATE↓ (%)	AOE↓ (Deg)	RSE↓ (%)	mIoU↑	Acc@0.25↑ (%)	Acc@0.50↑ (%)
Masks	Blender Python	8.12	30.71	8.48	0.521	84.73	59.26
2D boxes	JSON + executor	7.49	29.89	7.83	0.537	86.24	61.51
2D boxes	Blender Python	**7.24**	**29.63**	**7.67**	**0.540**	**86.59**	**61.95**

provide object crops for the Modeler. Blender Python is slightly better than JSON on all six metrics, including mIoU (0.537 to 0.540) and Acc@0.50 (61.51% to 61.95%). JSON can encode the same identities and transforms, so this result does not establish greater expressiveness for Python. Its practical benefit is a shared execution-aligned scene state across layout prediction, mesh binding, rendering, format/executability rewards, and visual refinement.

G Impact Statements

This work studies image-to-3D scene generation. It can support AR/VR content creation, simulation, game and film production, and digital-twin construction. By producing executable Blender Python code, the method also makes generated scenes easier to inspect and edit. Our training and evaluation are based on public datasets and 3D assets, and we do not collect private or personally identifiable data. However, generated 3D scenes may still reflect biases in the source images and asset databases. The system may also produce incorrect geometry, scale, or spatial relations, so its outputs should not be used directly in safety-critical applications without human review. Like other generative tools, it could be misused to create misleading synthetic environments or unauthorized replicas of real spaces. We encourage responsible use, careful dataset and asset licensing checks, and clear disclosure when generated 3D content is used.

H Limitation

Although *HoloCode* improves scene-level spatial reasoning and object fidelity, it still depends on the quality of object meshes. For common objects, the Modeler Agent retrieves meshes from an asset database. If the database does not contain a suitable asset, the retrieved mesh may have the wrong shape, style, or scale. For low-confidence cases, the pipeline uses image-to-3D object generation models to create meshes. These diffusion-based models can produce plausible geometry, but they may still miss fine details, generate artifacts, or fail on occluded and unusual objects. As a result, the final scene quality is limited by both asset coverage and object-level mesh generation quality. Future work can improve the asset retrieval process and use stronger object generation models.

Scene Complexity and Rotation Coverage. The framework has no fixed object-count bound, but practical capacity depends on VLM context and output-token budgets. Our empirical evaluation includes scenes with over 30 objects; more crowded scenes require longer outputs and more inter-object reasoning, increasing the risk of omissions, mismatches, and spatial errors. SAGE and 3D-FRONT mainly contain upright indoor objects. OOD evaluation uses full-3D AOE, but rotation-stratified results and broader orientation coverage remain future work.

Initialization and Single-View Ambiguity. Training on amplified Arranger errors exposes the Refiner to imperfect layouts, but does not establish recovery from arbitrary initializations. Supervised camera-frame transforms and scene context help resolve scale–depth ambiguity; truly indistinguishable single-view cases require additional cues.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Sec. 1 state the paper's scope, main contributions, and empirical claims for image-to-3D scene generation. These claims are supported by the method description in Sec. 3 and the experiments in Sec. 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix H discusses limitations from asset coverage and object-level mesh generation quality. It also identifies failure modes for rare, occluded, or unusual objects.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present formal theoretical results or theorem-proof claims. The mathematical content is used to define the training objective, evaluation metrics, and reward functions.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Secs. 3 and 4.1 describe the pipeline, datasets, training objectives, baselines, and evaluation metrics. Appendix A, Appendix B, and Appendix E provide implementation details, prompts, dataset splits, metric definitions, and reward definitions.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The current submission describes the public datasets, assets, prompts, implementation details, and evaluation protocol, but it does not yet provide an anonymized code and data release package. We plan to release code and derived assets when permitted by licensing and anonymity constraints.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Sec. 4.1 defines the experimental setting and metrics. Appendix A and Table 5 provide dataset settings, model backbone, training stages, optimizer-related hyperparameters, inference settings, and Modeler routing details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports deterministic benchmark results and ablations on fixed evaluation sets, but it does not report error bars, confidence intervals, or statistical significance tests. Repeated full training runs are computationally expensive for this setting.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A and Table 5 report the training hardware, memory-saving configuration, batch sizes, rollout backend, and training time for the trainable agents. These details cover the primary compute requirements needed to reproduce the reported experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The work uses public datasets, public or cited models, and 3D assets, and does not collect private or personally identifiable data. Ethical considerations and possible misuse are discussed in the Impact Statements.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The Impact Statements discuss positive applications in AR/VR, simulation, content creation, and digital twins. They also discuss potential bias, incorrect geometry, safety-critical misuse, misleading synthetic environments, and unauthorized replicas.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk scraped dataset, pre-trained language model, or general image generator. The Impact Statements describe responsible-use considerations for generated 3D scenes.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [No]

Justification: The paper credits the datasets, asset sources, models, and code framework through citations, including SAGE, 3D-FRONT, Objaverse, LVIS, Qwen, Trellis, SAM 3D,

and `ms-swift`. However, the current draft does not yet enumerate the license and terms of use for each asset source.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The current submission does not release the derived code datasets or trained agents as standalone new assets. Sec. 3.2, Appendix B, and Appendix A document their construction, prompts, training settings, and filtering/splitting choices.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or research with human subjects. Refiner CoT labels are generated with GPT-5 rather than collected from human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subject research. Therefore IRB approval or an equivalent human-subjects review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The core method uses VLM-based code-generating agents and GPT-5 for Refiner CoT annotation, as described in Sec. 3 and Appendix B. The experiments also identify LLM/VLM baselines and the Qwen3.5-9B-Base backbone in Sec. 4.2 and Appendix A.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.