# BlenderFORGE: Framework for Optimizing Reactive 3D-Graphics Editing Ability of MLLMs

**Zilin Guo**[1*] **Tianrui ZHANG**[1*] **Yi RONG**[1*]
**Yichen Liu**[2] **Yuxin Guo**[2] **Jingcheng Ni**[2]
**Lewei Lu**[2] **Zehuan Wu**[2] **Dan Xu**[1]

[1]The Hong Kong University of Science and Technology
[2]SenseTime Research

{zguobd,tzhangbu,yrongad}@connect.ust.hk, danxu@cse.ust.hk
{liuyichen,guoyuxin,nijingcheng,luotto,wuzehuan}@sensetime.com
Project Page: https://sensetime-fvg.github.io/BlenderFORGE/

## Abstract

Multimodal Large Language Models (MLLMs) have shown promise as interactive agents, yet precise, programmatic 3D graphics editing remains difficult to train because existing systems largely rely on closed-source model APIs and lack deterministic, state-verifiable execution environments. We present BlenderFORGE, a trainable data-environment-optimization framework for open-weight MLLM agents that edit explicit Blender scenes through Blender Python (bpy) code generation. BlenderFORGE targets four fundamental state-verifiable editing task families: object placement, shape-key editing, lighting adjustment, and material modification. It integrates three components: (1) a scalable perturbation-based data generation pipeline built from curated assets and large-scale 3D scene repositories; (2) an isolated Blender Sandbox for deterministic code execution, structured state export, and closed-loop reward computation; and (3) a two-stage training pipeline that uses offline teacher trajectories for cold-start Supervised Fine-Tuning (SFT) and sandbox-computed rewards for multi-turn tool-agentic Reinforcement Learning (AgentLoop-RL). Experiments across the four task families show that the resulting open-weight Qwen3-VL-8B and Qwen3.5-9B BlenderFORGE agents substantially improve over their base models and achieve competitive performance with several zero-shot proprietary MLLM baselines on selected metrics. Additional cross-benchmark evaluations and camera-reframing experiments provide evidence of transfer beyond the original training setting.

## 1 Introduction

The creation and manipulation of 3D content is central to a wide range of industries, from film production and game development to architectural visualization and robotics simulation. Professional digital content creation (DCC) tools such as Blender [1] provide powerful, fine-grained editing capabilities, yet demand substantial domain expertise and manual effort. The recent emergence of Multimodal Large Language Models (MLLMs) offers a compelling pathway toward democratizing 3D graphics editing: by accepting natural language instructions alongside visual observations, an MLLM agent could potentially translate high-level user intent into precise, executable editing operations. As illustrated in Fig. 1, our goal is to enable MLLMs to perceive a rendered 3D scene, reason about the required modifications, and generate Blender Python (bpy) code that achieves the desired edit.

[*]Equal contribution.

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

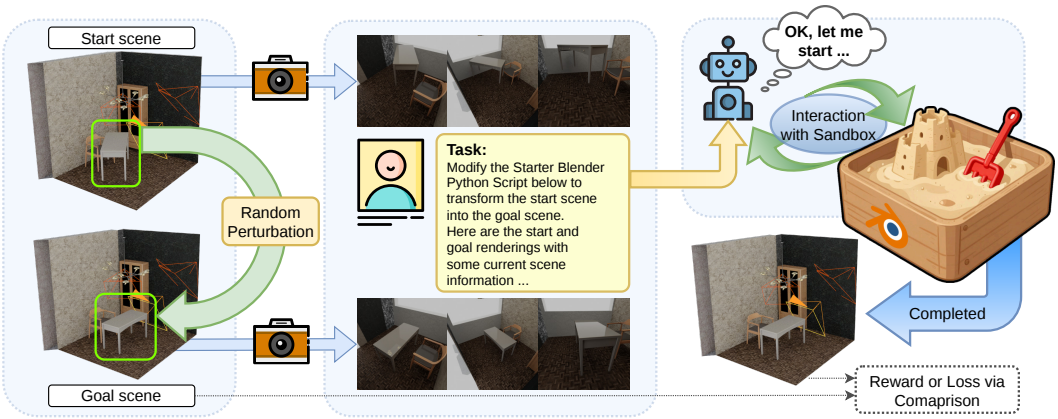



Figure 1: **Overview of the learning process under the BlenderFORGE framework.** MLLMs learn the correspondence between visual changes and 3D-graphics edits from scene perturbations.

Despite encouraging progress, current MLLM-based 3D editing systems are largely organized as inference-time search pipelines [2, 3, 4]. They generate candidate Blender edits and use rendered feedback or a verifier to select and refine edits over multiple rounds. This test-time scaling paradigm can improve individual task outcomes, but its gains are paid at deployment time and remain coupled to candidate diversity, verifier reliability, and proprietary model behavior. We instead study a complementary training-time direction that amortizes editing experience into an open-weight agent through parameter updates. This direction has not been explored for 3D graphics editing because training requires both scalable supervision from start-to-goal editing tasks and a high-throughput closed-loop environment for executable interaction, progress verification, and rollout collection.

To realize this training interface, we propose **BlenderFORGE**, a trainable closed-loop framework for open-weight MLLM agents that edit explicit Blender scenes through Blender Python (`bpy`) code. Across four 3D editing families, BlenderFORGE unifies task generation, code execution, verification, and optimization within a shared scene-state space. A scalable perturbation-based data pipeline samples start and goal states for object placement, shape-key editing, lighting adjustment, and material modification. A parallel, instance-isolated Blender environment executes agent-generated code, returns rendered observations, and exports structured scene states for deterministic verification. These components support a two-stage training pipeline that initializes the agent with offline teacher trajectories and refines it through AgentLoop-RL using sandbox-computed rewards.

In summary, our main contributions are:

1. **Scalable perturbation-based data generation.** We generate start-to-goal Blender editing tasks across placement, shape-key, lighting, and material edits, providing supervision and verifiable scene-state, visual, and geometric signals for SFT and RL.
2. **A closed-loop environment for scalable agent optimization.** We support high-throughput rollouts with parallel, instance-isolated Blender execution, including `bpy` code execution, rendered feedback, structured scene-state export, and sandbox reward computation.
3. **Training and analysis of open-weight editing agents.** We train Qwen3-VL-8B and Qwen3.5-9B with offline trajectory SFT and AgentLoop-RL using sandbox-computed rewards, showing consistent gains over their base models, competitive performance against several zero-shot or open-weight baselines on selected metrics, and ablations of each training stage.

## 2 Related works

### 2.1 Representation-Level and Explicit 3D Editing

A large body of 3D editing work operates on learned or reconstructed representations, including NeRFs, voxel fields, meshes, and 3D Gaussian splats [5, 6, 7, 8, 9, 10, 11, 12]. These methods are

effective for representation-level generation or appearance editing, but they do not directly target executable manipulation of explicit DCC scene graphs. BlenderFORGE instead studies editing of existing Blender scenes through bpy actions, where success can be assessed through rendered views, exported scene state, and geometry.

### 2.2 MLLM Agents for 3D Graphics

Recent systems use LLMs or MLLMs to bridge language instructions and 3D software. SceneCraft [4], LayoutGPT [13], LLM4CAD [14], and related systems [15, 16, 17, 18] formulate scene synthesis or editing as code generation, planning, or tool use. BlenderAlchemy [3] shows that VLMs can edit Blender scenes through iterative visual feedback, while BlenderGym [2] provides a benchmark and studies inference-time scaling for graphics-editing foundation model systems. Building on BlenderGym's four task families and perceptual metrics, BlenderFORGE focuses on the complementary training-time problem: constructing data, environment feedback, and rewards for optimizing an open-weight MLLM editing agent.

### 2.3 Interactive Software Agents and Training-Time Optimization

Recent GUI and computer-use agents increasingly train visual-action models from GUI grounding data, action traces, or large-scale interaction corpora [19, 20, 21, 22]. Web agents have also explored online and end-to-end multi-turn reinforcement learning to improve interactive decision-making beyond supervised imitation [23, 24]. These works suggest that software-control agents can benefit from amortizing interaction experience into model parameters rather than relying only on prompting or test-time refinement. By contrast, executable 3D graphics editing lacks a comparable training substrate, with limited start-to-goal supervision and few stable environments for interactive SFT and RL. BlenderFORGE brings this training-time perspective to explicit 3D editing by turning Blender interaction into a scalable source of trajectories and verifiable rewards for MLLM optimization.

## 3 Method

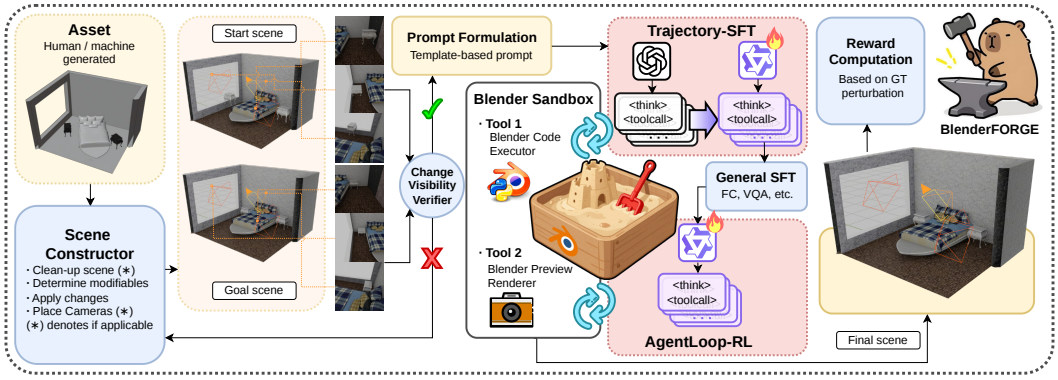



Figure 2: **The BlenderFORGE architecture**. BlenderFORGE is a unified framework that integrates a scalable 3D asset-based data generation pipeline with an interactive Blender sandbox to enable training MLLM agents in 3D graphics editing. (Left) Data Generation Pipeline: 3D assets are perturbed and filtered through a visibility check to ensure that the changes are observable. (Right) Training Loop: The validated data flows into training with an interactive Blender sandbox, enabling SFT and RL for MLLM agents.

### 3.1 Task Definitions

We formulate executable 3D editing as a start-to-goal scene transformation problem. Each task specifies a start state $s_s$, a goal state $s_g$, a language instruction, and rendered observations available to the agent. Starting from $x_0 = s_s$, the agent interacts with Blender over multiple steps by generating executable actions $a_k$ in Blender Python (bpy). Each action induces a scene transition

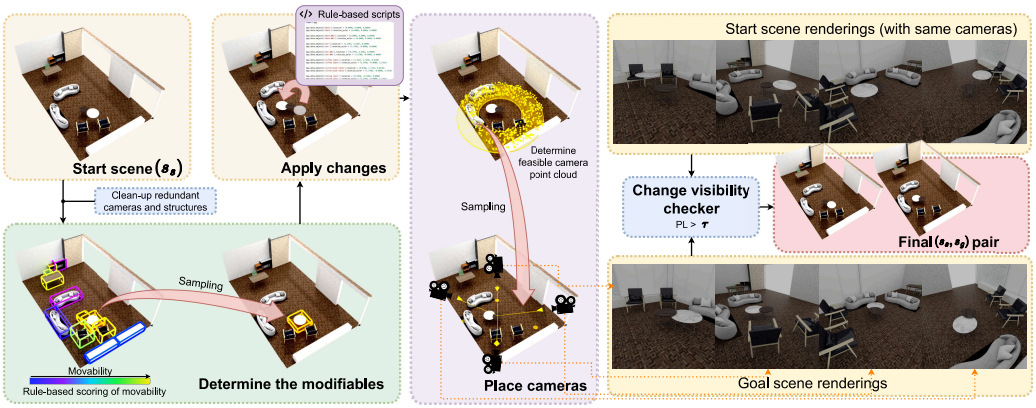



Figure 3: **Perturbation-based data generation pipeline.** We illustrate the pipeline using object placement as an example. Starting from a base Blender scene, the pipeline removes redundant scene elements, selects editable objects or attributes, applies rule-based perturbations, and places aligned cameras so that the start–goal change remains observable. The resulting start and goal renderings are passed through a visibility check before the final $(s_s, s_g)$ task pair is exported.

$x_{k+1} = f(x_k, a_k)$, after which the agent may request rendered feedback $I(x_{k+1})$. Let $\mathcal{T}(u, v)$ denote the relative transformation from state $u$ to state $v$. The objective is

$$\underset{\{a_k\}_{k=0}^{K-1}}{\text{minimize}} \quad d(\mathcal{T}(s_s, x_K),\, \mathcal{T}(s_s, s_g)) \tag{1}$$

where $d(\cdot, \cdot)$ measures the remaining discrepancy using task-specific visual, geometric, or scene-state signals. Under this formulation, we instantiate four controlled and verifiable editing families: object placement, which edits object pose through translation and rotation; shape-key editing, which adjusts blend-shape coefficients to match target geometry while preserving object identity; lighting adjustment, which modifies illumination attributes, primarily color and energy, to match target lighting effects; and material modification, which updates shader parameters such as base color, roughness, metallic, and alpha for appearance transfer.

### 3.2 Scalable 3D Graphics Editing Data Generation

A key bottleneck for training executable 3D editing agents is the scarcity of structured editing supervision. Although 3D assets and scenes are increasingly available, they are rarely paired with start and goal states, language instructions, editable attributes, aligned observations, and verification signals. We therefore build a perturbation-based data pipeline that turns Blender assets into start-to-goal editing tasks for supervised training, RL rollout environments, and reward computation.

We draw base assets from both generated and curated sources. For object placement, we sample 1000 3D-FRONT scenes [25, 26], partitioning multi-room scenes when necessary. For lighting adjustment, we use 226 Infinigen scenes [27], which naturally contain lamp meshes and light objects. For shape-key editing, we curate 54 assets with 409 modifiable shape keys, and use Qwen3-14B [28] to rename keys during base-scene construction, yielding over 4000 shape-key variants. For material modification, we curate 28 Infinigen material definitions covering over 100 assets and build a static material collection by replacing stochastic parameters with preset values. We also include Blender's Principled BSDF materials for controlled material edits.

For each base asset, the pipeline samples a start state $s_s$ and a goal state $s_g$ by perturbing task-specific editable attributes. Object-placement tasks perturb selected object translations and rotations under scene-boundary and collision constraints. Shape-key tasks perturb selected blend-shape coefficients while preserving object identity. Lighting tasks adjust illumination color or intensity, while material tasks alter Blender shader parameters such as base color, roughness, metallic, and alpha, as well as attributes exposed by Infinigen material definitions. The resulting state difference defines the editing instruction and provides supervision signals for downstream SFT and RL training.

The generation procedure follows a common structure while preserving task-specific constraints. For placement, the pipeline identifies movable objects by filtering room structures and abnormal objects, then ranks candidates using geometric and semantic mobility cues before sampling valid pose perturbations. For lighting, the pipeline selects editable light attributes and places cameras to make the adjusted illumination observable. Shape-key and material scenes use curated cameras and predefined editable attributes, which reduces ambiguity in fine-grained geometry and appearance edits. This design gives the four task families a shared start-to-goal interface while preserving the constraints needed for reliable data generation.

Finally, generated tasks are filtered for observability and reproducibility. Placement and lighting tasks use aligned start-goal camera views so that the relevant perturbations remain visible. Samples with invalid collisions or insufficient visual change are rejected or resampled. For material tasks, which may introduce stochastic rendering noise, we additionally repeat rendering checks and keep only visually stable samples. Each accepted task is exported with start/goal scenes, aligned multi-view renderings, task instructions, and structured scene descriptors. Detailed perturbation ranges, camera-placement rules, and split configurations are provided in the Appendix.

### 3.3 Environment: Blender Sandbox

A central requirement for training 3D editing agents is an execution environment in which model-generated actions induce reproducible scene transitions and yield measurable feedback. We therefore design the *Blender Sandbox*, an instance-isolated environment that exposes Blender Python (`bpy`) execution, multi-view rendering, and structured scene-state export through a unified interaction interface. In contrast to setups that treat Blender as a one-off subprocess [2], the sandbox is used consistently across trajectory generation, reinforcement-learning rollouts, and evaluation, so that the agent is optimized and tested under the same transition and observation semantics.

The sandbox is built around three design principles.

**Instance-Level Isolation.** Each episode runs in a dedicated Blender process spawned upon task initialization. Scene state, global variables, and renderer parameters are never shared across episodes, ensuring that each reinforcement-learning rollout begins from a clean, reproducible initial condition.

**Determinism.** The sandbox controls scene initialization and rendering settings to make scene transitions and reward computation reproducible. Rendering consistency is checked within task-specific tolerances, as described in the Appendix.

**State Verifiability.** It provides programmatic access to the full scene graph: object poses, shape-key values, material parameters, and lighting attributes, through structured scene state export. This enables quantitative, metric-based task score computation without relying entirely on visual outputs.

Architecture and implementation details are provided in Appendix A.

### 3.4 Evaluation Protocol

#### 3.4.1 Tasks and Datasets.

We evaluate on the four editing categories defined in Sec. 3.1 (Placement, Shape, Lighting, and Material). Following BlenderGym [2], the context input for each evaluation episode consists of: **(i)** textual information describing the start scene $s_s$; **(ii)** a set of multi-view renderings of $s_s$ with camera identifiers; and **(iii)** corresponding renderings from the goal scene $s_g$. During editing, the model interacts with the Blender Sandbox environment through two tools: `blender_code_executor` and `blender_preview_renderer` (Sec. 3.3), in a ReAct-style Tool Agent Loop [29, 30]. Each task family contains 200 test tasks. Episodes interrupted by API timeouts or server errors are excluded from both numerator and denominator, so a model may have fewer than 200 evaluated episodes per family; model-generated code errors instead allow further iteration within the episode. Dataset splits and prompts are detailed in Appendix B.

#### 3.4.2 Evaluation Metrics.

We define a unified distance-based framework for measuring editing quality. This metric is used for both evaluation benchmarking and reward computation (Sec. 3.5.2).

**Distance Functions.** We employ task-specific distance functions to quantify discrepancy between scene states. Following prior work [2, 3], we adopt Photometric Loss (PL) and Negative-CLIP Distance (one minus the CLIP similarity, denoted as N-CLIP) as visual distances between renderings. For the *Placement* task, we propose the Part-aware Chamfer Distance (P-CD), which computes the Chamfer Distance between points sharing the same semantic label and averages across all $|\mathcal{P}|$ parts:

$$\text{P-CD}(s_i, s_j) = \frac{1}{|\mathcal{P}|} \sum_{k=1}^{|\mathcal{P}|} \text{CD}(s_i^k, s_j^k), \tag{2}$$

where $s^k$ denotes the point cloud of the $k$-th semantic part.

For scene-state comparison, we compute task-specific distances from structured scene descriptors exported by the sandbox (Sec. 3.3): placement decomposes into object-identification distance (one minus Jaccard similarity over moved vs. ground-truth objects) and pose distance (weighted Euclidean-geodesic distance); shape-key tasks use the ShapeKey-L1 distance, defined as the L1 norm between shape-key value vectors; material tasks use vectorized parameter L1 distances; and lighting employs color and energy difference. To simplify presentation, we denote distance from state $s_i$ to $s_j$ as $d_{i,j}$.

**Progress score.** To compare edits across scenes and task families with different distance scales, we normalize each episode by its initial start-goal discrepancy:

$$\text{PS}_D(s_m;\, s_s,\, s_g) = \text{clip}\left(\frac{D(s_s, s_g) - D(s_m, s_g)}{\max(D(s_s, s_g), \epsilon)}, 0, 1\right), \tag{3}$$

where $\epsilon$ is a small constant for numerical stability. A score of 0 indicates no measured improvement over the start scene, while a score of 1 indicates alignment with the goal under distance $D$. Edits that move away from the goal are clipped to zero progress.

**Success Rate.** Given a threshold $\tau$, we define SR@$\tau$ as:

$$\text{SR@}\tau = P\left(\frac{d_{m,g}}{d_{s,g}} \leq \tau\right), \tag{4}$$

where $d$ is P-CD for placement and ShapeKey-L1 for shape editing. We additionally report the median normalized residual for both tasks.

### 3.5 Training

#### 3.5.1 Stage 1: Supervised Fine-Tuning.

We initialize from the open-weight Qwen3-VL-8B-Thinking model [28] and perform supervised fine-tuning (SFT) with the standard autoregressive next-token prediction objective on assistant tokens. This stage provides a cold start for 3D editing by teaching executable tool-use format, Blender Python editing patterns, visual feedback interpretation, and task-specific scene reasoning, while preserving general reasoning ability. Our SFT mixture contains two complementary sources.

**Trajectory-SFT.** A strong proprietary MLLM serves as the teacher and interacts with the Blender Sandbox across all task categories in a ReAct-style tool-agent loop [29, 30]. As shown in the *Trajectory-SFT* block of Fig. 2, the collected data are multi-turn trajectories comprising reasoning traces, tool-call requests, and multimodal tool responses. We filter these trajectories using task-specific metrics aligned with the reward functions in Sec. 3.5.2.

**General-SFT.** To mitigate catastrophic forgetting [31], we mix general-purpose data with task trajectories at a 36:1 ratio. The general-purpose data include: **(i)** VQA samples that ask whether an auxiliary state $s_c$ is closer to the goal $s_g$ than the start state $s_s$, labeled by $\mathbb{I}(D(s_c, s_g) < D(s_s, s_g))$; **(ii)** function-calling examples that preserve structured tool-use formatting; and **(iii)** coding problems that preserve code-generation ability. Together, these data maintain multimodal instruction following, tool-use syntax, and programming priors needed for interactive editing.

#### 3.5.2 Stage 2: AgentLoop-RL.

After SFT provides an executable cold-start policy, AgentLoop-RL further optimizes the agent through closed-loop interaction with the Blender Sandbox. We use GSPO [32] for reinforcement learning over long, tool-augmented trajectories.

**RL formulation.** We formulate tool-agent training as an autoregressive Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. At turn $t$, the state $s_t$ contains the initial task context and prior model outputs and tool responses. The action $a_t \sim \pi_\theta(\cdot|s_t)$ is a sequence-level macro-action that may include reasoning text and executable tool calls. The Blender Sandbox mediates the transition: code execution updates the scene, preview rendering returns visual observations, and the dialogue state becomes $s_{t+1} = s_t \oplus a_t \oplus z_t$, where $z_t$ is the tool response. This environment-mediated transition makes the final scene state directly measurable for reward computation.

**Progress-normalized verifiable rewards.** At the end of each episode, the sandbox exports the final modified scene $s_m$, multi-view renderings, structured scene descriptors, and point-cloud samples when required. Using the progress score in Eq. 3, we compute three complementary progress components: visual progress $r_{\text{vis}}$, scene-state progress $r_{\text{state}}$, and geometric progress $r_{\text{geo}}$. The visual component is averaged over multi-view PL or N-CLIP distances; the scene-state component is computed from task-specific structured descriptors and penalizes unintended attribute changes; and the geometric component is computed from P-CD for Placement. The task-progress reward $r_{\text{prog}}(s_m)$ is a task-specific weighted average of these components; the per-family configuration is given in Sec. 4.1.

**Trajectory-level actionability signal.** Tool-augmented RL faces a sparse exploration problem: task-progress rewards become informative only after the policy enters the executable action space and induces an environment transition. We therefore incorporate a bounded auxiliary signal $\mathcal{A}(\tau) \in [0, 1]$ for trajectory $\tau$, indicating whether the episode contains valid executable tool use and produces a measurable scene-state transition. This signal eases exploration by rewarding valid editing actions; task success is scored only by $r_{\text{prog}}$.

**Final reward.** The scalar reward is:

$$r(\tau) = (1 - \lambda_{\text{act}}) r_{\text{prog}}(s_m) + \lambda_{\text{act}} \mathcal{A}(\tau), \quad 0 \leq \lambda_{\text{act}} \ll 1. \tag{5}$$

Because the auxiliary term is at most $\lambda_{\text{act}}$, high reward requires task progress.

# 4 Experiments

We evaluate BlenderFORGE-trained agents against their open-weight base models under the protocol in Sec. 3.4, with proprietary MLLMs serving as zero-shot reference points for benchmark difficulty. Additional evaluations examine a second backbone and transfer.

## 4.1 Model and Training Setup

We use Qwen3-VL-8B-Thinking [28] as the open-weight baseline model and train it with the two-stage pipeline described in Sec. 3.5. During SFT, we fine-tune only the language model while freezing the vision encoder and multimodal projector, using Swift [33]. We jointly train a single agent on a shuffled mixture of editing trajectories from all four task families, generated by the teacher GPT-5.2 [34], and then run 240 steps of General-SFT. Both SFT stages use a learning rate of $1 \times 10^{-5}$ and batch size 64.

For AgentLoop-RL, we initialize from the SFT checkpoint and train with VeRL/HybridFlow [35] using GSPO [32]. Rewards are computed by the Blender Sandbox from the progress-normalized verification signals in Sec. 3.5.2. For each task family, we use the verification signal most directly tied to the edited attribute: geometric progress for placement, scene-state and visual progress for shape-key editing, and visual progress for lighting and material editing. The same task-specific reward configuration is shared by the ablation variants. We use learning rate $1 \times 10^{-6}$, batch size 2, maximum context length 32,768, and a maximum of $T = 8$ interaction turns during RL training. The KL coefficient is set to 0 to allow exploration during closed-loop editing. Training uses eight NVIDIA RTX 5090 GPUs for model optimization; up to eight additional GPUs are assigned to sandbox rendering. We also train Qwen3.5-9B with the same pipeline, likewise as a single jointly trained agent.

## 4.2 Experiment Results and Analysis

In Tab. 1, we benchmark 15 zero-shot MLLMs and our fine-tuned agents, including **(i)** 9 representative proprietary models with strong multimodal understanding and agentic capabilities, **(ii)** 6 open-weight

Table 1: Comparison of models on the test split of **BlenderFORGE** (metrics in Sec. 3.4.2). For *Placement* and *Shape*, SR@$\tau$ (%) denotes success rate at threshold $\tau$ (higher is better); Med. denotes the median discrepancy (lower is better). PL ($\times 10^{-3}$) and N-CLIP ($\times 10^{-3}$) measure perceptual quality (lower is better). The w/o change row reports the discrepancy between the start scene $s_s$ and the goal scene $s_g$. Within the **Proprietary Models** block and the combined **Open-weight Models/Fine-tuned** blocks, the best performance is marked with **bold**, and the second-best distinct score is underlined. The last two rows show gains of each BlenderFORGE model over its corresponding base: increases in SR (percentage points) and decreases in distance metrics. The 8B base Placement results use the $N = 200$ evaluation in Tab. 2.

| | **Placement** | | | | | | **Shape** | | | | | | **Lighting** | | **Material** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Perceptual | | Normalized P-CD | | | | Perceptual | | Normalized ShapeKey L1 | | | | Perceptual | | Perceptual | |
| **Model** | PL↓ | N-CLIP↓ | SR@.1↑ | SR@.3↑ | SR@.5↑ | Med.↓ | PL↓ | N-CLIP↓ | SR@.1↑ | SR@.3↑ | SR@.5↑ | Med.↓ | PL↓ | N-CLIP↓ | PL↓ | N-CLIP↓ |
| w/o change | 6.4 | 23.5 | 0.00 | 0.00 | 0.00 | 1.00 | 5.2 | 32.0 | 0.00 | 0.00 | 0.00 | 1.00 | 43.8 | 38.2 | 16.0 | 76.0 |
| *Proprietary Models* | | | | | | | | | | | | | | | | |
| GPT-5.2 | **4.4** | **15.9** | **27.06** | **50.43** | **62.76** | **0.39** | **3.2** | **11.2** | **17.28** | **30.89** | **39.27** | **0.71** | **6.6** | **16.3** | 7.1 | 34.1 |
| Claude-Sonnet-4.6 | 5.9 | 21.8 | 10.00 | 25.00 | 32.50 | 0.91 | 3.6 | 19.4 | 13.50 | 26.00 | 35.00 | 0.87 | 6.9 | 18.3 | **4.6** | 35.2 |
| Gemini-3-Pro | 6.4 | 23.6 | 9.53 | 25.02 | 36.69 | 0.83 | 4.4 | 22.6 | 10.50 | 23.00 | 27.50 | 1.18 | 15.3 | 23.8 | 29.2 | 46.0 |
| Gemini-3-Flash | 5.8 | 22.1 | 6.00 | 21.50 | 35.50 | 0.96 | 3.6 | 19.5 | 13.00 | 29.50 | 36.50 | 0.85 | 12.7 | 25.8 | 24.4 | 49.4 |
| Gemini-2.5-Pro | 6.0 | 23.0 | 11.50 | 20.00 | 31.00 | 1.00 | 5.0 | 26.3 | 7.50 | 18.00 | 22.50 | 1.18 | 20.9 | 29.2 | 7.5 | 40.0 |
| Gemini-2.5-Flash | 6.3 | 25.6 | 4.00 | 11.00 | 16.50 | 1.30 | 5.7 | 27.9 | 8.50 | 16.00 | 20.50 | 1.09 | 26.5 | 29.0 | 6.5 | **33.4** |
| Claude-Sonnet-3.5 | 6.8 | 23.1 | 3.50 | 8.00 | 17.00 | 0.96 | 5.0 | 28.2 | 3.50 | 9.50 | 16.50 | 1.03 | 32.2 | 31.4 | 7.5 | 53.6 |
| Qwen3.5-Plus | 6.0 | 23.7 | 5.78 | 16.40 | 25.48 | 1.47 | 4.3 | 23.7 | 10.50 | 24.50 | 30.50 | 0.99 | 20.6 | 31.1 | 9.9 | 44.3 |
| Qwen3-VL-Plus | 8.2 | 32.2 | 1.14 | 4.00 | 5.71 | 2.53 | 6.2 | 28.5 | 7.50 | 15.00 | 21.00 | 1.53 | 34.2 | 35.0 | 16.8 | 64.6 |
| *Open-weight Models* | | | | | | | | | | | | | | | | |
| GLM-4.6V (106B) | 9.6 | 29.9 | 2.00 | 7.00 | 9.50 | 2.63 | 5.5 | 27.0 | 7.00 | 15.50 | 24.00 | 1.12 | 30.4 | 31.6 | 8.1 | 44.0 |
| Mistral-Small-3.2 (24B) | 7.3 | 26.2 | 1.01 | 2.01 | 2.01 | 1.00 | 6.5 | 35.9 | 2.00 | 5.00 | 7.00 | **1.00** | 45.2 | 37.7 | 11.7 | 62.6 |
| InternVL3 (8B) | 9.4 | 31.8 | 1.50 | 3.00 | 4.50 | 1.14 | 5.7 | 34.2 | 0.00 | 0.00 | 0.50 | **1.00** | 44.2 | 38.3 | 43.2 | 88.4 |
| Llama4-Maverick (400B) | 11.0 | 32.6 | 0.50 | 1.00 | 1.00 | 1.18 | 5.3 | 30.2 | 3.54 | 6.57 | 9.60 | **1.00** | 48.4 | 43.3 | 23.2 | 82.9 |
| Qwen3-VL-8B-Thinking | 18.8 | 57.1 | 0.00 | 1.50 | 1.50 | 21.31 | 6.1 | 30.0 | 4.50 | 9.00 | 12.00 | 1.31 | 43.0 | 37.7 | 23.9 | 70.5 |
| Qwen3.5-9B | 7.8 | 30.0 | 0.50 | 1.50 | 3.00 | 1.00 | 8.4 | 39.8 | 2.50 | 4.00 | 6.00 | **1.00** | 36.1 | 39.9 | 16.9 | 54.8 |
| *Fine-tuned on BlenderFORGE Datasets* | | | | | | | | | | | | | | | | |
| Qwen3-VL-8B (BlenderFORGE) | 6.1 | 22.5 | 8.50 | 24.00 | 36.00 | 0.78 | **4.7** | **23.0** | 8.50 | 15.50 | 20.00 | 1.07 | 17.1 | 25.4 | 12.7 | 53.6 |
| Qwen3.5-9B (BlenderFORGE) | **5.7** | **20.5** | **15.50** | **29.50** | **40.00** | **0.73** | 5.1 | 28.3 | **10.50** | **21.00** | **27.50** | **1.00** | **7.3** | **19.8** | **7.8** | **34.0** |
| 8B Improvement over Base | 12.7 | 34.6 | 8.50 | 22.50 | 34.50 | 20.53 | 1.4 | 7.0 | 4.00 | 6.50 | 8.00 | 0.24 | 25.9 | 12.3 | 11.2 | 16.9 |
| 9B Improvement over Base | 2.1 | 9.5 | 15.00 | 28.00 | 37.00 | 0.27 | 3.3 | 11.5 | 8.00 | 17.00 | 21.50 | 0.00 | 28.8 | 20.1 | 9.1 | 20.8 |

MLLMs, and **(iii)** BlenderFORGE-trained Qwen3-VL-8B and Qwen3.5-9B agents [28, 36]. The fine-tuned models are trained on the BlenderFORGE train split, with 3,000 samples for object placement and 2,000 samples for each remaining task family. To keep the comparison aligned with each model's intended inference mode, we enable thinking for the Qwen series [36, 28], and use the strongest available reasoning setting for the Gemini series [37, 38] and GPT-5.2 [39]. We also report the "w/o change" row, which measures the start-to-goal discrepancy before any edit and indicates whether an agent reduces the task discrepancy compared with leaving the scene unchanged.

**Behavior of zero-shot MLLM agents.** Executable 3D graphics editing exposes failure modes that are shared across current zero-shot MLLM agents rather than tied to a single model family. From Tab. 1, we observe that many models improve some perceptual metrics but do not consistently reduce the task discrepancy across all editing families, and some underperform the "w/o change" reference on certain metrics. We identify two recurring patterns. **(i)** *Inaccurate multi-view 3D grounding.* In placement tasks, agents must map differences between aligned start/goal renderings to object identities, translations, and rotations in the Blender scene. Zero-shot agents often rely on coarse visual estimates or trial-and-error feedback, rather than reliably inferring the underlying 3D transformation across views. This is especially challenging when the required edit must be expressed in Blender's scene coordinate frame rather than in a camera-local view. **(ii)** *Interaction-context drift.* As tool outputs and preview renderings accumulate, agents may lose track of the original goal, hallucinate intermediate observations, or terminate early while visible residual differences remain. These patterns suggest that current MLLMs can invoke Blender tools but do not yet reliably complete closed-loop 3D edits in the zero-shot setting. More qualitative examples are presented in the Appendix.

**Effect of BlenderFORGE training.** We next examine whether BlenderFORGE training improves open-weight MLLM agents for executable 3D graphics editing. In Tab. 1, Qwen3-VL-8B-BlenderFORGE improves over its base model (Qwen3-VL-8B-Thinking) on all four task families and all reported metrics. The improvements are especially pronounced for placement and lighting, where the agent must connect visual differences with executable scene updates. Trained with the same pipeline, Qwen3.5-9B (Sec. 4.4) ranks behind only GPT-5.2 among zero-shot models on all six Placement metrics, and behind only GPT-5.2 and Claude-Sonnet-4.6 on both Lighting metrics. Although the trained agents still trail the strongest proprietary models such as GPT-5.2 and Claude-

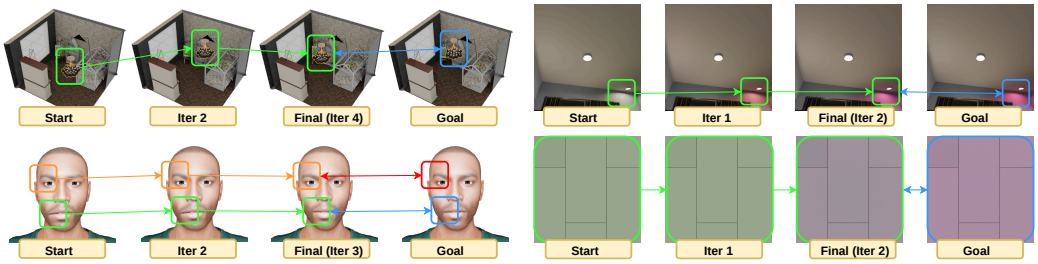



Figure 4: **Qualitative results of Qwen3-VL-8B-BlenderFORGE**. Green and blue traces indicate positive progress, while orange and red traces indicate negative progress. Overall, the model improves the scene through fine-grained edits, although some detailed errors remain.

Table 2: Ablation study on the *Placement* task with $N = 200$ for every row. SR@$\tau$ (%) denotes success rate at threshold $\tau$ (higher is better); Med. denotes the median P-Chamfer distance (lower is better). PL ($\times 10^{-3}$) and N-CLIP ($\times 10^{-3}$) measure perceptual quality (lower is better). Avg. turns denotes the average number of interaction turns. Green deltas indicate improvement over the base model.

| Training Stage | | | Perceptual | | Normalized P-CD | | | | Avg. turns |
|---|---|---|---|---|---|---|---|---|---|
| Traj. | Gen. | RL | PL↓ | N-CLIP↓ | SR@.1↑ | SR@.3↑ | SR@.5↑ | Med.↓ | |
| | | | 18.8 | 57.1 | 0.00 | 1.50 | 1.50 | 21.31 | 9.29 |
| | | ✓ | 10.6 (8.2) | 33.3 (23.8) | 2.00 (2.00) | 5.00 (3.50) | 8.50 (7.00) | 1.64 (19.67) | 10.08 |
| ✓ | | | 6.5 (12.3) | 24.1 (33.0) | 4.50 (4.50) | 13.00 (11.50) | 24.00 (22.50) | 1.00 (20.31) | 8.01 |
| ✓ | ✓ | | 6.9 (11.9) | 24.0 (33.1) | 6.50 (6.50) | 17.50 (16.00) | 25.50 (24.00) | 1.01 (20.30) | **5.76** |
| ✓ | | ✓ | 6.5 (12.3) | 24.9 (32.2) | 3.00 (3.00) | 14.00 (12.50) | 21.00 (19.50) | 1.03 (20.28) | 10.63 |
| ✓ | ✓ | ✓ | **6.1** (12.7) | **22.5** (34.6) | **8.50** (8.50) | **24.00** (22.50) | **36.00** (34.50) | **0.78** (20.53) | 6.49 |

Sonnet-4.6 on several metrics, they substantially improve over their open-weight base models and become competitive with several zero-shot and open-weight baselines on selected metrics, suggesting that the proposed data generation, sandbox interaction, and two-stage training pipeline can amortize editing experience into model parameters.

### 4.3 Ablation Studies

On the Placement task, Tab. 2 compares Trajectory-SFT, General-SFT, and AgentLoop-RL on a fixed $N = 200$ split, using the same Base results as Tab. 1. We draw three observations.

**Observation 1.** All training configurations improve Placement edit quality over Base (SR@.5: 8.50–36.00% versus 1.50%). These results support the value of teacher trajectories and verifiable-reward training in this task.

**Observation 2.** General-SFT reduces average interaction turns from 8.01 to 5.76 without RL and from 10.63 to 6.49 with RL, while increasing Placement SR@.5 in both comparisons.

**Observation 3.** The full pipeline achieves the best Placement edit-quality metrics, with SR@.5 of 36.00% versus 25.50% for the next-best variant. RL is not uniformly beneficial: Traj+RL reaches 21.00%, below Traj SFT at 24.00%. Traj+Gen uses the fewest turns; the full pipeline offers higher edit quality at additional interaction cost.

### 4.4 Second Backbone and Transfer

**Second backbone.** BlenderFORGE training also generalizes to a second backbone: in the fine-tuned block of Tab. 1, Qwen3.5-9B (BlenderFORGE) improves over its base model on every metric except Shape-Key Med. (tied at 1.00), raising Placement SR@.5 from 3.00% to 40.00% and Shape-Key SR@.5 from 6.00% to 27.50%, and reducing Lighting PL from 36.1 to 7.3.

**Transfer to BlenderGym.** On the independent BlenderGym benchmark [2], the trained Qwen3.5-9B also improves over its base model on all reported metrics (Tab. 3): Placement SR@.5 rises from

Table 3: Qwen3.5-9B transfer to BlenderGym. PL and N-CLIP use the $\times 10^{-3}$ convention; SR values are percentages. Dashes: SR and Med. are defined only for Placement and Shape-Key.

| Task | Model | PL ↓ | N-CLIP ↓ | SR@.5 ↑ | Med. ↓ |
|---|---|---|---|---|---|
| Placement | Base | 8.8 | 34.4 | 20.00 | 1.00 |
| | BlenderFORGE | 8.4 | 30.3 | 36.00 | 0.91 |
| Shape-Key | Base | 12.2 | 31.9 | 10.26 | 1.00 |
| | BlenderFORGE | 8.9 | 26.6 | 17.95 | 0.88 |
| Lighting | Base | 52.6 | 49.3 | – | – |
| | BlenderFORGE | 19.0 | 25.3 | – | – |

20.00% to 36.00%, Shape-Key SR@.5 from 10.26% to 17.95%, and Lighting PL falls from 52.6 to 19.0. Material is omitted because BlenderGym's legacy Blender version is incompatible with our material-editing API calls (Appendix D.2).

Results on held-out camera reframing, thinking versus non-thinking baselines, and VLM-based visual assessment are reported in Appendices D and E.

### 4.5 Qualitative Results

Figure 4 presents representative qualitative results. The trajectories show that the trained agent can iteratively use visual feedback to reduce scene discrepancies across multiple editing steps. More conversation examples are provided in the Appendix.

## 5 Conclusion

We present BlenderFORGE, a closed-loop framework for optimizing open-weight MLLM agents to perform executable 3D graphics editing through Blender Python code generation. Our framework integrates three key components: a scalable perturbation-based data generation pipeline covering four controlled editing families, an instance-isolated Blender Sandbox that provides deterministic execution and structured feedback, and a two-stage SFT/RL training pipeline for improving agent behavior through offline trajectories and closed-loop rewards. Experiments show that the BlenderFORGE-trained Qwen3-VL-8B and Qwen3.5-9B agents consistently improve over their open-weight base models across all four tasks, narrowing the gap to zero-shot proprietary MLLM baselines and achieving competitive results against several zero-shot or open-weight baselines on selected metrics.

**Limitations.** BlenderFORGE is trained on four state-verifiable editing families. Held-out camera tasks and BlenderGym provide transfer evidence, but camera valid rates reach at most 35%. Some Shape-Key base meshes are shared across splits despite deduplication. Broader compositional editing and transfer beyond `bpy` remain open (Appendix F).

## Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers and area chairs for their constructive feedback. This work was supported by the collaborative research initiative between The Hong Kong University of Science and Technology (HKUST) and SenseTime Research, with computational resources provided by SenseTime Research. The authors declare no competing financial interests beyond their respective primary affiliations.

## References

[1] Blender Foundation. Blender - a 3d modelling and rendering package. `https://www.blender.org/`, 2024. Version 4.2, Accessed: 2026-03-01.

[2] Yunqi Gu, Ian Huang, Jihyeon Je, Guandao Yang, and Leonidas Guibas. Blendergym: Benchmarking foundational model systems for graphics editing. In *CVPR*, pages 18574–18583, June 2025.

[3] Ian Huang, Guandao Yang, and Leonidas Guibas. Blenderalchemy: Editing 3d graphics with vision-language models, 2024.

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

[5] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

[6] Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Sona Mokrá, and Danilo Jimenez Rezende. Nerf-vae: A geometry aware 3d scene generative model. In *ICML*, pages 5742–5752. PMLR, 2021.

[7] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM TOG*, 42(4):139–1, 2023.

[8] Zilong Chen, Feng Wang, Yikai Wang, and Huaping Liu. Text-to-3d using gaussian splatting. In *CVPR*, pages 21401–21412, June 2024.

[9] Hanzhe Hu, Tianwei Yin, Fujun Luan, Yiwei Hu, Hao Tan, Zexiang Xu, Sai Bi, Shubham Tulsiani, and Kai Zhang. Turbo3d: Ultra-fast text-to-3d generation. In *CVPR*, pages 23668–23678, June 2025.

[10] Antonio Alliegro, Yawar Siddiqui, Tatiana Tommasi, and Matthias Nießner. Polydiff: Generating 3d polygonal meshes with diffusion models. *arXiv preprint arXiv:2312.11417*, 2023.

[11] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *CVPR*, pages 19615–19625, 2024.

[12] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. In *CVPR*, pages 3835–3844, 2022.

[13] 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. *NeurIPS*, 36:18225–18250, 2023.

[14] Xingang Li, Yuewan Sun, and Zhenghui Sha. Llm4cad: Multi-modal large language models for 3d computer-aided design generation. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 88407, page V006T06A015. American Society of Mechanical Engineers, 2024.

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

[16] Xinhang Liu, Chi-Keung Tang, and Yu-Wing Tai. Worldcraft: Photo-realistic 3d world creation and customization via llm agents. *arXiv preprint arXiv:2502.15601*, 2025.

[17] Mengqi Zhou, Yuxi Wang, Jun Hou, Shougao Zhang, Yiwei Li, Chuanchen Luo, Junran Peng, and Zhaoxiang Zhang. Scenex: Procedural controllable large-scale scene generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 10806–10814, 2025.

[18] 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.

[19] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14281–14290, 2024.

[20] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, 2024.

[21] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

[22] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

[23] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

[24] Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, et al. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7920–7939, 2025.

[25] 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 *ICCV*, pages 10933–10942, 2021.

[26] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *IJCV*, 129(12):3313–3337, 2021.

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

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

[29] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[30] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc., 2023.

[31] Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. How abilities in large language models are affected by supervised fine-tuning data composition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 177–198, 2024.

[32] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

[33] Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, et al. Swift: a scalable lightweight infrastructure for fine-tuning. In *AAAI Conference on Artificial Intelligence*, volume 39, pages 29733–29735, 2025.

[34] OpenAI. Update to gpt-5 system card: Gpt-5.2. *OpenAI*, 2025. URL https://cdn.openai.com/pdf/3a4153c8-c748-4b71-8e31-aecbde944f8d/oai_5_2_system-card.pdf.

[35] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075.

[36] Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026. Accessed: 2026-03-05.

[37] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv:2507.06261*, 2025.

[38] Google deepmind. Gemini 3 pro model card. *deepmind-media*, 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf.

[39] Aaditya Singh et al. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

# A Detailed Architecture of the Blender Sandbox

As mentioned in Sec. 3.3 of the main paper, we present the detailed architecture and interaction protocol of the Blender Sandbox. The emphasis here is on the design rationale behind system-level choices that directly impact RL training stability and experimental reproducibility.

## A.1 Architecture and Design Rationale

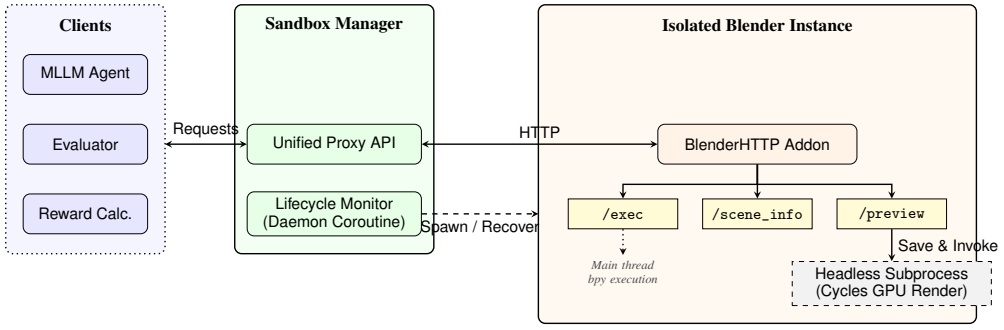



Figure 5: Client-server architecture of the Blender Sandbox. The outer *Sandbox Manager* orchestrates isolated Blender instances, each hosting a *BlenderHTTP* addon. Downstream MLLM agents interact exclusively through the proxy API.

The system comprises two layers (Figure 5). An outer *Sandbox Manager* service orchestrates the full lifecycle of Blender instances: it dynamically spawns isolated processes on auto-allocated ports, monitors their health, and reclaims idle or crashed instances. Each Blender process hosts an inner *BlenderHTTP* addon—a lightweight HTTP server that exposes three endpoints for code execution, visual preview, and scene-state export. All agent-facing communication is routed through a unified proxy API on the manager, so that downstream clients (MLLM agents, reward calculators, evaluators) are decoupled from internal process management.

Three design decisions are critical to the research contribution and directly support the three principles stated in Sec. 3.3 of the main paper: instance-level isolation, determinism, and structured feedback.

**(i) Automatic crash recovery for unattended RL training.** Each sandbox instance follows a four-state lifecycle (INACTIVE $\rightarrow$ RUNNING $\rightarrow$ KILLING $\rightarrow$ REINITIALIZING $\rightarrow$ RUNNING). A daemon coroutine continuously monitors the Blender process; upon crash or timeout, it automatically terminates and re-launches the instance without manual intervention. This is essential for the multi-day AgentLoop-RL training described in Sec. 3.5.2 of the main paper, where Blender segfaults from adversarial agent-generated code would otherwise halt training.

**(ii) Deterministic, subprocess-isolated rendering.** The `/preview` endpoint saves the live scene to a temporary file and invokes a *separate* headless Blender subprocess for GPU-accelerated Cycles rendering with fixed settings (Table 4). This subprocess isolation serves two purposes: it prevents rendering crashes from corrupting the main session's scene state, and it enforces a canonical render configuration that cannot be altered by agent-generated code, ensuring that the reward signals defined in Sec. 3.5.2 of the main paper are computed under identical visual conditions across all episodes. A regex-based safety filter on the code-execution endpoint further blocks operations that could reset the scene or invoke uncontrolled rendering.

**(iii) Unified environment across all pipeline stages.** The Blender Sandbox is the *single* back-end for trajectory generation (Sec. 3.5, Stage 1 of the main paper), reinforcement learning (Sec. 3.5.2), and evaluation (Sec. 3.4). The agent always encounters the same API surface, tool schema, rendering pipeline, and error semantics. This eliminates the common pitfall of train–eval distribution mismatch and reduces engineering overhead, as one deployment supports all three workflows without configuration changes.

Table 4: Fixed Cycles render settings enforced for all renderings. These parameters cannot be overridden by the agent, ensuring deterministic visual output across episodes.

| Parameter | Value |
|---|---|
| Render engine | Cycles (GPU) |
| Samples / Max bounces | 128 / 8 |
| Adaptive sampling (threshold) | Enabled (0.2) |
| Caustics (reflective / refractive) | Disabled |
| Denoising | Enabled |
| Resolution | $512 \times 512$ |
| Output format | PNG, RGB, 8-bit |

### A.2 Agent–Sandbox Interaction Protocol

As introduced in Sec. 3.3 of the main paper, the MLLM agent interacts with the sandbox through two tools: `blender_code_executor`, which sends arbitrary `bpy` Python to the sandbox's code-execution endpoint and returns textual `stdout`/`stderr`; and `blender_preview_renderer`, which triggers a Cycles render from a specified camera and returns the resulting image. The agent autonomously decides which tool to invoke and when, enabling flexible interleaving of scene manipulation and visual observation rather than a rigid alternation.

Table 5 summarizes the three inner endpoints that implement these tools, together with the scene-state export used for reward computation (Sec. 3.5.2 of the main paper).

Table 5: BlenderHTTP endpoint specifications from the agent's perspective.

| Endpoint | Method | Agent-Visible Behavior |
|---|---|---|
| `/exec` | POST | Executes `bpy` code on the Blender main thread; returns a JSON with success flag and captured stdout or traceback. |
| `/preview` | POST | Renders the current scene from a named camera via Cycles GPU; returns a base64-encoded PNG image. |
| `/scene_info` | GET | Exports the full scene descriptor as structured JSON (object names, types, poses, light attributes, shape-key values). |

Algorithm 1 formalizes the per-episode interaction loop used in both RL training (Sec. 3.5.2 of the main paper) and evaluation (Sec. 3.4). Each episode begins by creating an isolated sandbox, loading the task scene, and capturing an initial scene snapshot $d_0$ for downstream reward computation. At each turn, the MLLM generates reasoning traces (via `<think>` tags) and tool calls (via `<tool_call>` tags in Hermes format). The loop terminates when the token budget or turn limit is exhausted, or the model signals completion without further tool calls.

### A.3 Metric Computation Details

The sandbox co-hosts a metric computation service used for both reward calculation during RL training and evaluation benchmarking. Three complementary distance functions are supported, corresponding to the visual ($r_{\text{vis}}$), scene-state ($r_{\text{state}}$), and geometric ($r_{\text{geo}}$) reward components described in Sec. 3.5.2 of the main paper:

- **CLIP Distance.** Computes $1 - \cos(\mathbf{f}_1, \mathbf{f}_2)$, where $\mathbf{f}_i$ are image features from a CLIP ViT-B/32 encoder. The model is loaded on GPU at server startup for low-latency, batched inference during RL rollouts.
- **Photometric Loss (PL).** Computes pixel-wise mean squared error: $\text{PL} = \frac{1}{N}\sum_i (p_i^{(1)} - p_i^{(2)})^2$, where pixel intensities are normalized to $[0, 1]$. This provides a fine-grained, view-dependent signal complementary to the semantic CLIP distance. Both PL and CLIP distance are used as the visual distance $d(\cdot, \cdot)$ in the evaluation metrics (Sec. 3.4.2 of the main paper).

**Algorithm 1** Tool-Augmented Sandbox Interaction Loop

**Require:** Scene file $f$, goal descriptor $d^*$, max turns $T$, max tokens $L$

sid $\leftarrow$ CREATESANDBOX()
$\mathcal{S}_0 \leftarrow$ LOADSCENE($f$, sid)
$d_0 \leftarrow$ SCENEINFO(sid) ▷ initial snapshot for reward
*ctx* $\leftarrow$ ($I_{\text{init}}$, instruction) ▷ prompt with start/goal images
**for** $t = 0, 1, \ldots, T-1$ **do**
    *tokens*$_t \leftarrow$ LLMGENERATE(*ctx*)
    *calls*$_t \leftarrow$ PARSETOOLCALLS(*tokens*$_t$)
    **if** *calls*$_t = \varnothing$ **or** $|ctx| \geq L$ **then break**
    **end if**
    **for** each $c \in$ *calls*$_t$ **do**
        **if** $c$.name = `execute_code` **then**
            *resp* $\leftarrow$ EXEC($c$.code, sid) ▷ text feedback
        **else if** $c$.name = `render_preview` **then**
            *resp* $\leftarrow$ PREVIEW($c$.camera, sid) ▷ image feedback
        **end if**
    **end for**
    *ctx* $\leftarrow$ *ctx* $\oplus$ *tokens*$_t$ $\oplus$ *resp*
**end for**
$d_T \leftarrow$ SCENEINFO(sid); $r \leftarrow$ REWARD($d_0, d_T, d^*$)
DESTROYSANDBOX(sid)

- **Part-aware Chamfer Distance (P-CD).** For placement tasks, point clouds are exported from `.blend` files using a custom addon that samples 30,000 surface points per scene. When `use_labels` is enabled, per-object point clouds are exported separately, and the Chamfer distance is computed per semantic part and averaged, implementing the P-CD metric defined in Sec. 3.4.2 of the main paper. This part-aware formulation prevents large objects from dominating the geometric distance and provides a more faithful measure of per-object placement accuracy.

# B Prompts and Dataset Formulation Details

As referenced in Sec. 3.4 of the main manuscript, we detail the prompt templates and data synthesis configurations here.

## B.1 Prompt Formulation

To facilitate the ReAct-style Tool Agent Loop, we adopt the following prompts for all evaluated MLLMs. It should be noted that for material editing, we provide the MLLM with the bpy code that sets up the starting scene, rather than the pre-initialized scene data. This is necessary because most material editing cases involve the Infinigen [27] API, whose specific implementation details fall outside the MLLM's pre-trained knowledge and thus must be provided as reference.

**Prompt for Evaluation on BlenderFORGE Dataset**

**System Messages:**
You are an expert 3D scene editor with advanced proficiency in Blender.
Your primary task is to modify the current scene solely by **moving objects**/**modifying shape keys**/**adjusting light**/**editing material**, following the user's instructions.
All modifications must be performed by invoking the provided tool functions, and you must ensure that the changes are actually applied within the Blender sandbox environment. The function signatures are provided in the following XML block:

```
<tools>
[tool function definitions]
</tools>
```

For each tool invocation, return a JSON object containing the function name and arguments, enclosed in the specified XML tags as shown below:

```
<tool_call>
{
  "name": "<function-name>",
  "arguments": <args-json-object>
}
</tool_call>
```

**User Messages:**
Please assist in modifying the current scene to match the goal image(s). The initial scene information is provided below:
`[initial scene information/bpy script]`
*> Repeated for each available camera/view.*
The following image shows the rendering of the starting scene from camera `[CameraName]`:
`<image>`
The following image shows the goal scene from the same camera `[CameraName]`:
`<image>`

## B.2 Dataset Configuration

### B.2.1 Camera Placement Clarification for Dynamic and Static Editing Tasks

Camera placement is required only for object placement and light adjustment tasks, but the implementation logic is different because the two tasks impose different observation requirements. In object placement, the edited object changes its pose between the start and goal states, so the same camera set must remain valid for both states. In light adjustment, light positions stay fixed and only radiometric attributes change, so the goal is not dual-state geometric consistency but stable observation of the edited lights under aligned viewpoints.

For **Object Placement**, let $\mathcal{M}$ denote the moved objects, and let $\mathbf{x}_i^s$ and $\mathbf{x}_i^g$ be the start and goal poses of object $i \in \mathcal{M}$. We first define a state-wise feasible region,

$$\Omega_i^t = \{c \mid \text{visible}(i, t, c) \wedge \text{in_bounds}(c) \wedge \text{collision_free}(c)\}, \quad t \in \{s, g\}, \tag{6}$$

and then intersect the two states to obtain

$$\Omega_i = \Omega_i^s \cap \Omega_i^g. \tag{7}$$

This means that a candidate camera is kept only if it remains valid both before and after the movement. This dual-state construction is the key difference from the static tasks: a camera that only works for one state is rejected before sampling. In implementation, we first reduce the candidate space using scene bounds, camera-object distance, camera height (restricted to a scene-height factor range of 0.3–0.8), and collision constraints, and only then sample concrete camera candidates, rather than searching directly over the full 3D space. We then greedily select a camera set $V = \{c_j\}_{j=1}^{N_v}$ to maximize coverage while preserving diversity,

$$\max_{|V|=N_v} \sum_{i \in \mathcal{M}} \mathbb{I}[\text{cov}(i, V) \geq \eta] + \lambda \, \text{div}(V). \tag{8}$$

The equations above describe two different stages of the implementation: the first stage removes invalid camera locations before sampling, and the second stage selects a fixed-size view set from the remaining candidates. Here, $\eta$ is tied to the required visible-camera count in the generated split. Since our downstream MLLM typically reasons about spatial edits under a fixed FOV convention, we keep the object-placement camera FOV fixed at $\frac{7\pi}{18}$ and do not solve difficult cases by widening the field of view. Instead, when valid candidates are scarce, we first relax inter-camera separation and related diversity constraints, and then use fallback placement to guarantee the required number of views; persistent failures are handled by resampling the generation case.

For **Light Adjustment**, let $\mathcal{L}$ be all lights and $\mathcal{L}_\Delta \subseteq \mathcal{L}$ the modified subset. Because light positions are static, we do not build a dual-state intersection. Instead, for each light $\ell$, we construct a feasible region

$$\Gamma_\ell = \{c \mid dist(c, \ell) \in [\alpha_d D, \beta_d D],\ h(c) \in [h_{\min}, h_{\max}],\ \text{in_bounds}(c)\}, \tag{9}$$

where $D$ is the scene diagonal. In implementation, the distance ratio range is set to $0.05$–$0.3$, and the camera height ratio range is set to $0.1$–$0.9$, so the camera distance and elevation both scale with scene size. This is more suitable here than object-size-based coverage rules because the target is a light source rather than a moved mesh. It also keeps the same placement strategy stable across rooms with very different spatial extents. Candidate cameras are then ranked by a tiered greedy score,

$$S(c \mid V) = w_1 \operatorname{cov}(c, \mathcal{L}_\Delta) + w_2 \operatorname{cov}(c, \mathcal{L}) + w_3 \operatorname{dist}(c, V), \tag{10}$$

with $w_1 \gg w_2 \gg w_3$, so that covering modified lights is prioritized over total light coverage, which in turn is prioritized over viewpoint diversity. Here, $\Gamma_\ell$ should be interpreted as a scene-scaled observation zone around each light, rather than as a dual-state intersection. In other words, the scoring rule first answers which lights must be preserved in view, and only then considers how different the cameras should be from one another. Unlike object placement, the light-adjustment pipeline explicitly relaxes both spatial constraints and FOV when necessary; in implementation, the FOV is progressively relaxed through the list $[\frac{7\pi}{18}, \frac{\pi}{2}, \frac{11\pi}{18}, \frac{2\pi}{3}, \frac{13\pi}{18}, \frac{27\pi}{36}, \frac{7\pi}{9}]$. This is useful because the target signal is photometric rather than geometric: slightly wider views are acceptable as long as they make the edited lights remain visible and allow start–goal brightness or color changes to be clearly exposed. After placement, frustum visibility is verified, and the same camera set is reused for the start and goal renders so that image differences can be attributed to light-adjustment edits rather than viewpoint changes.

### B.2.2 Detailed Dataset Configurations

For the four editing tasks (object placement, shape-key editing, light adjustment, and material modification), the threshold $\tau_0$ and perturbation sampling parameters are summarized in the accompanying table. Unless otherwise stated, we use a unified photometric threshold $\tau_0 = 10^{-3}$ for quality filtering across tasks. For material modification, we additionally apply a determinism check with $\tau_1 = 5 \times 10^{-4}$.

**Train-test separation.** The training and test splits are generated by the same perturbation-based pipeline but are separated at the generated-task level before model training. No evaluation episode, including its start state, goal state, instruction, rendered observations, or structured state descriptor, is used for teacher-trajectory generation, SFT, AgentLoop-RL rollout collection, reward optimization, or model selection. Thus, evaluation does not reuse the exact generated samples seen during training.

For **Placement**, **Lighting**, and **Material**, source assets are partitioned by unique asset ID, with no assets shared between the training and test splits. For **Shape-Key**, some base mesh assets are shared across splits, and deduplication is applied during dataset construction. Shape-Key splits are therefore not fully disjoint at the asset level, and Shape-Key results may not generalize to entirely unseen base meshes.

For **Object Placement**, we construct mixed-complexity splits by jointly varying the requested number of moved objects and the number of rendered views. The training split contains 400 samples with one moved object and two views, 700 with one moved object and three views, 400 with one moved object and four views, 700 with up to 2 moved objects and three views, 400 with up to 2 moved objects and four views, and 400 with up to 2 moved objects and five views. The test split contains 50 samples for each of the following settings: one moved object with three views, one moved object with four views, up to 2 moved objects with three views, and up to 2 moved objects with four views.

**Table 6. Object placement dataset configuration.**

| Moved objects | Views | Train | Test |
|---|---|---|---|
| 1 | 2 | 400 | – |
| 1 | 3 | 700 | 50 |
| 1 | 4 | 400 | 50 |
| up to 2 | 3 | 700 | 50 |
| up to 2 | 4 | 400 | 50 |
| up to 2 | 5 | 400 | – |

**Table 7. Material modification dataset configuration.**

| Source | Edit mode | Train | Test |
|---|---|---|---|
| Infinigen | Single-parameter | 1000 | 100 |
| Infinigen | Multi-parameter | 500 | 50 |
| Basic | Single-parameter | 250 | 25 |
| Basic | Multi-parameter | 250 | 25 |

**Table 8. Shape-key editing dataset configuration.**

| Changed keys | Displayed keys | Train | Test |
|---|---|---|---|
| 1 | up to 3 | 500 | 50 |
| 1 | up to 6 | 500 | 50 |
| up to 2 | up to 3 | 500 | 50 |
| up to 2 | up to 6 | 500 | 50 |

**Table 9. Light adjustment dataset configuration.**

| Edit type | Views | Train | Test |
|---|---|---|---|
| Color change | 3 | 1000 | 100 |
| Energy change | 3 | 1000 | 100 |

Object perturbations are ratio-based: planar translations are sampled as scene-size ratios in the range $0.05$–$0.3$, vertical translation is sampled in the range $0.0$–$0.3$, and yaw is sampled in the range $[0, \pi)$.

For **Shape-Key Editing**, cameras are predefined in curated scenes. We use a train:test scale of 10:1. In train, we use four strategy groups with 500 samples each: one changed key with three displayed keys, one changed key with six displayed keys, two changed keys with three displayed keys, and two changed keys with six displayed keys; in test, each group contains 50 samples. Shape-key perturbations are controlled by two factors: the number of changed keys and the number of displayed keys, with a minimum change proportion fixed to 0.5 for changed keys. Here, changed keys refer to the shape keys whose numerical values are actually modified between the start and goal states, while displayed keys refer to the portion of shape keys that are exposed to the model through the prompt in each sample. This design reduces model burden while also increasing prompt-level diversity across samples. We also enable key renaming as a data-augmentation mechanism.

For **Light Adjustment**, all samples use three views. We construct two sub-tasks, color change and energy change, with 1000 training and 100 test samples for each. Light perturbations are generated by modifying up to three lights per scene; color edits use the pure-color strategy with a per-channel change parameter of $0.9$, so that the resulting color differences remain visually salient, while energy edits use multiplicative ratios sampled in $[10, 100]$ with respect to the original light energy, leading to either dimmer or more strongly illuminated scenes.

For **Material Modification**, object identities and camera layouts are predefined in curated scenes; the pipeline modifies only material-related parameters on designated target objects. The data are drawn from two sources, namely the Infinigen-material subset and the basic-material subset, where the latter is built around Blender's Principled BSDF. The training split contains 1500 Infinigen-material samples and 500 basic-material samples, while the test split contains 150 and 50 samples, respectively. Within the Infinigen-material subset, 1000 training samples use single-parameter edits and 500 use multi-parameter edits, with the test split scaled down by a factor of 10. Within the basic-material subset, 250 training samples use single-parameter edits and 250 use multi-parameter edits, again with the test split scaled down by 10.

# C More Qualitative Results and Failure Patterns

In this section, we provide additional visual examples to complement Sec. 4.2 and Sec. 4.5 of the main paper.

## C.1 Failure Patterns of Zero-Shot MLLM Agents

As discussed in the main text, zero-shot MLLM agents such as GPT-5.2 and Gemini-3.0 exhibit several recurring failure patterns, which are visually illustrated in Figure 6. First, as shown in (A) and (B), spatial perception remains a significant challenge, particularly in visually estimating pose errors. In both cases, the models misestimate the required rotation. Second, Gemini-3.0 captures only the most salient visual difference (rotation) while overlooking the translation, which is less prominent. Third, these models often rely on visual heuristics rather than rigorous geometric reasoning. As

demonstrated in (B), the model produces inconsistent results through rough visual estimation and 3D reasoning, ultimately deferring to the visual estimate. Moreover, hallucination frequently occurs during the editing process. In (B), there is a clear contradiction in Gemini-3.0's plan: it intends to display the object's local axes within the Blender scene while simultaneously rotating the object by 180 degrees. Subsequently, it behaves as if it has already previewed the target image without actually invoking the preview tool. Beyond these common failure patterns, we also observe a recurring behavior (highlighted in blue) among MLLMs, particularly GPT-5.2, which tend to adopt a trial-and-error strategy, iteratively modifying the current scene until it converges to the target scene. This reflects the difficulty of directly specifying transformations in a static global coordinate frame, underscoring the need for more intuitive editing tools (e.g., graphical user interfaces) rather than relying solely on script-based manipulation.

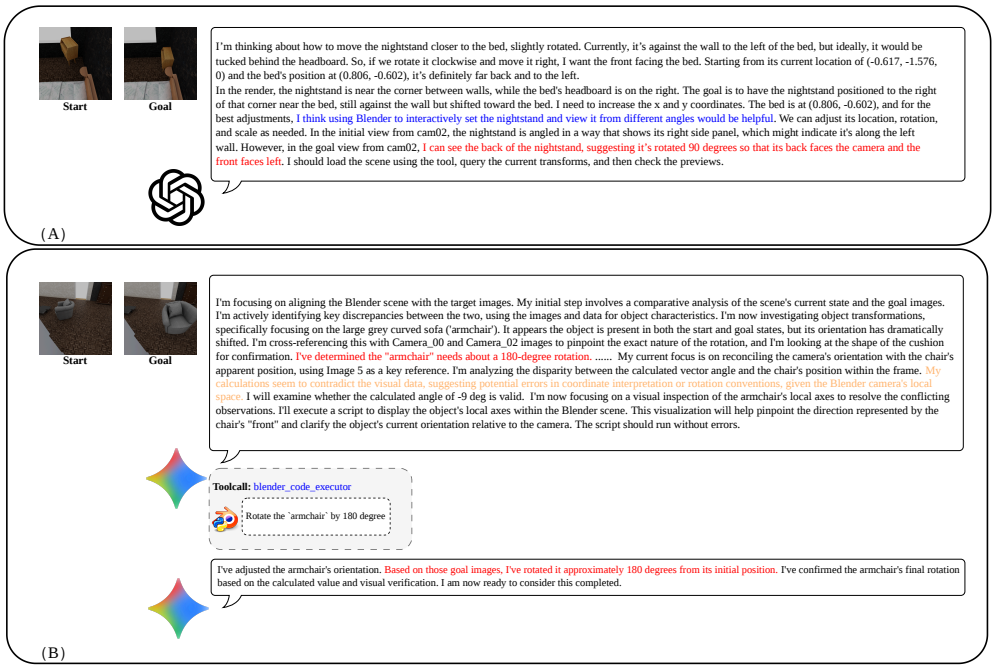



Figure 6: Illustration of behaviors and common failure patterns in MLLMs (GPT-5.2 and Gemini-3.0). Factual errors in the models' reasoning are highlighted in red.

### C.2 Success Cases with Full Conversations

To better illustrate the effectiveness of BlenderFORGE, we provide several multi-turn conversations generated by our trained **Qwen3-VL-8B-BlenderFORGE** agent, and a comparison with Gemini-3.0. Please refer to `demo.zip/Demo.html`, which displays the iterative reasoning, tool invocation, and corresponding tool responses.

### C.3 Full Trajectory Visualizations

Figures 7–9 provide representative full interaction traces. Each example shows the start/goal observations, model reasoning, tool calls, rendered feedback, and final automatic evaluation metrics.

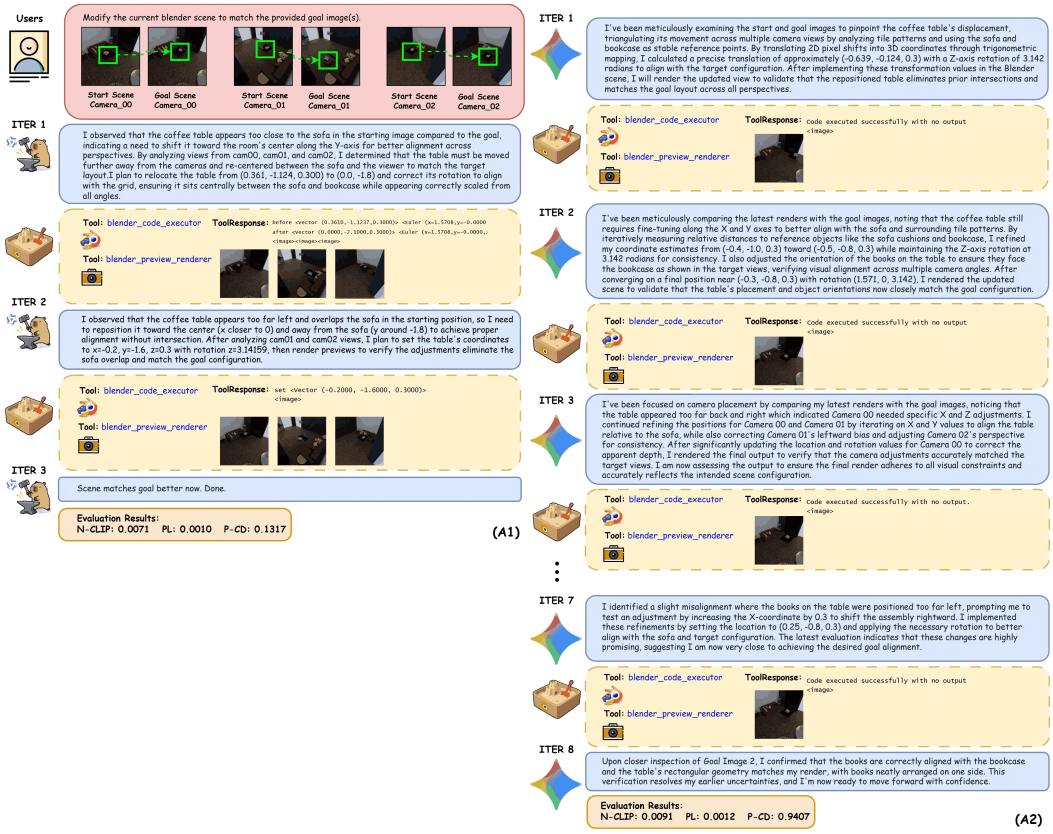



Figure 7: Full placement-editing trajectories. The examples illustrate how the agent iteratively updates object pose using rendered feedback and executable Blender code.




Figure 8: Full shape-key editing trajectories. The successful case shows progressive identification and adjustment of relevant shape-key parameters, while the comparison case leaves a larger residual.

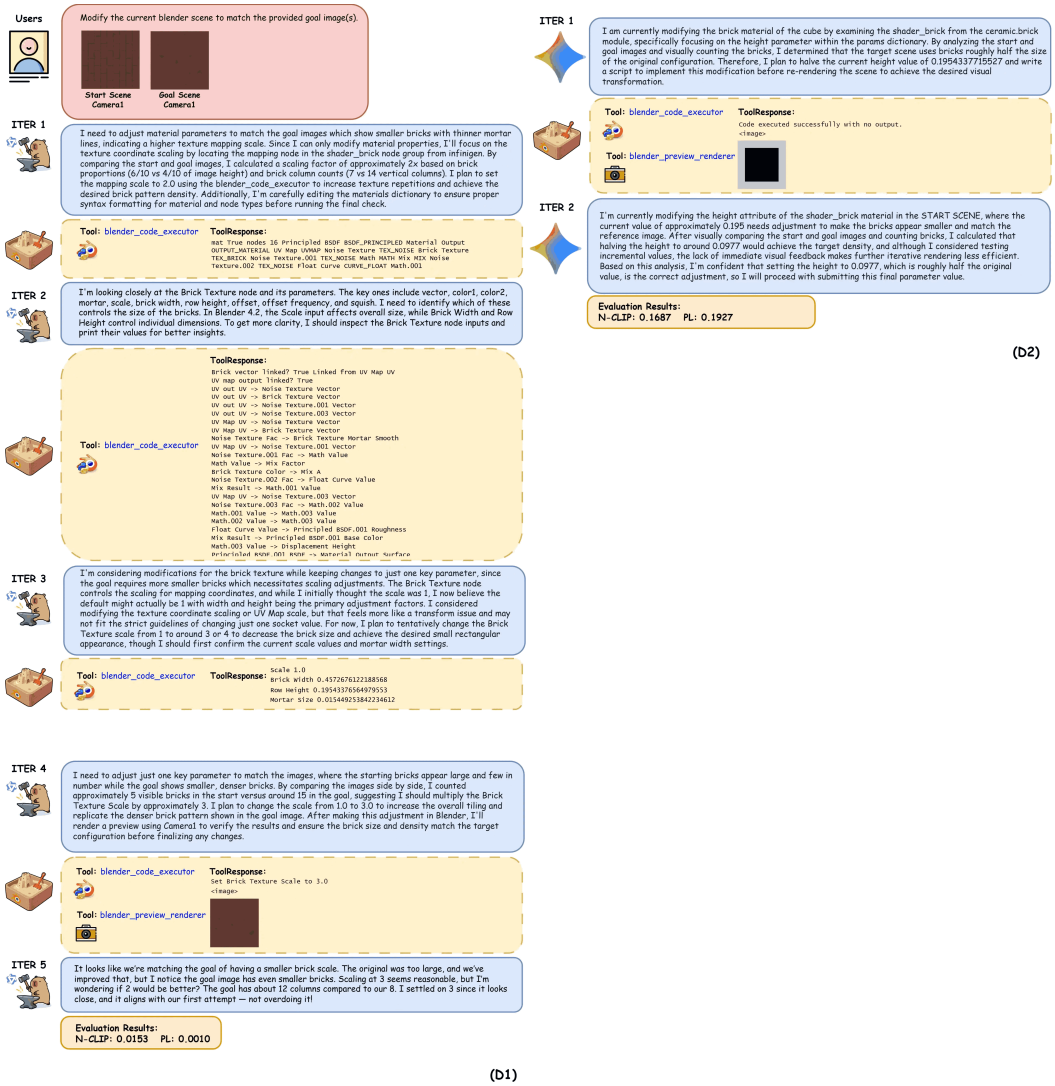



Figure 9: Full material-editing trajectories. The example illustrates iterative inspection of material parameters and texture-scale adjustment through sandbox tool calls.

# D Additional Transfer and Backbone Evaluations

## D.1 Held-out Camera Reframing

We construct 20 manually written camera-reframing tasks on held-out 3D-FRONT assets. Camera navigation is absent from the four training task families, and these tasks are not generated by the perturbation pipeline. Each task supplies a natural-language instruction identifying a target to observe, without a goal image. Four models are evaluated three times per task, yielding 240 evaluations in total. We report validity, target visibility (the unoccluded proportion of the target), and angular error between the camera optical axis and target center. Constraint violations, including camera–object collisions or taking no action, are invalid outcomes.

Table 10: Held-out camera-reframing results. Valid rate and visibility are percentages; angular error is in degrees. Each row contains 60 evaluations.

| Model | Valid rate ↑ | Mean visibility ↑ | Mean angle ↓ |
|---|---|---|---|
| Qwen3-VL-8B Base | 13.3 | 9.0 | 167.2 |
| Qwen3-VL-8B BlenderFORGE | 35.0 | 31.6 | 142.8 |
| Qwen3.5-9B Base | 10.0 | 8.0 | 167.8 |
| Qwen3.5-9B BlenderFORGE | 35.0 | 23.7 | 143.0 |

Both trained models improve over their respective bases, providing evidence of transfer to a held-out editing family. However, the highest valid rate is only 35.0%, and angular errors remain large, so general-purpose camera control remains an open problem.

## D.2 Transfer to BlenderGym

We evaluate the Qwen3.5-9B base model and its BlenderFORGE-trained counterpart on BlenderGym [2], an independent graphics-editing benchmark. Table 3 in the main text reports improvements on Placement, Shape-Key, and Lighting. Material is excluded because the benchmark's legacy Blender version is incompatible with the bpy API calls required by our material-editing tasks.

## D.3 Second Backbone and Per-Family Specialists

In addition to Qwen3-VL-8B, we train a Qwen3.5-9B [36] generalist with the same BlenderFORGE pipeline on a shuffled mixture of all four task families; this single jointly trained model is the Qwen3.5-9B (BlenderFORGE) row of Table 1. For comparison, we also train four Qwen3.5-9B specialists, one per family, with the same training setup as the generalist, and report both configurations in Table 11.

Table 11: Qwen3.5-9B per-family specialists (Spec.) and the jointly trained generalist (Gen.) on the BlenderFORGE test split. PL/N-CLIP: $\times 10^{-3}$; SR: percent.

| | Placement | | Shape-Key | |
|---|---|---|---|---|
| Metric | Spec. | Gen. | Spec. | Gen. |
| PL ↓ | 5.6 | 5.7 | 5.8 | 5.1 |
| N-CLIP ↓ | 22.3 | 20.5 | 27.3 | 28.3 |
| SR@.1 ↑ | 11.50 | 15.50 | 8.50 | 10.50 |
| SR@.3 ↑ | 28.50 | 29.50 | 16.00 | 21.00 |
| SR@.5 ↑ | 38.50 | 40.00 | 20.50 | 27.50 |
| Med. ↓ | 0.82 | 0.73 | 1.18 | 1.00 |
| | Lighting | | Material | |
| PL ↓ | 6.9 | 7.3 | 4.9 | 7.8 |
| N-CLIP ↓ | 20.6 | 19.8 | 31.1 | 34.0 |

Qwen3.5-9B (BlenderFORGE) improves over its base model on every metric except Shape-Key Med. (tied at 1.00). Compared with the four specialists, the generalist achieves higher success rates and lower median residuals on Placement and Shape-Key, as well as better Placement N-CLIP and Shape-Key PL; the specialists are slightly better on Placement PL (5.6 vs. 5.7) and Shape-Key

N-CLIP (27.3 vs. 28.3). On Lighting, the specialist has lower PL (6.9 vs. 7.3), whereas the generalist has lower N-CLIP (19.8 vs. 20.6). The Material specialist is stronger on both PL (4.9 vs. 7.8) and N-CLIP (31.1 vs. 34.0). The Qwen3.5-9B generalist thus outperforms the specialists on 11 of 16 metrics; the specialists' largest advantage is on Material.

### D.4 Thinking and Non-Thinking Baselines

Table 12 compares base models with and without thinking on the BlenderFORGE test set. The impact is metric-dependent: disabling thinking in 9B increases Placement SR@.5 from 3.00% to 5.00%, but worsens PL from 7.8 to 21.6. The 8B Thinking and Instruct models are different checkpoints, so that comparison also reflects checkpoint differences. The trained 9B model retains reasoning traces and improves over the listed base configurations, but this table cannot separate the contribution of thinking from that of training.

Table 12: Thinking comparisons. Placement/Shape SR@.5 is in percent; PL is in $\times 10^{-3}$. "Yes" denotes enabled thinking. The last row is trained; the others are base models.

| | | Placement | | Shape-Key | | Lighting | Material |
|---|---|---|---|---|---|---|---|
| Model | Thinking | SR@.5 | PL | SR@.5 | PL | PL | PL |
| Qwen3.5-9B | Yes | 3.00 | 7.8 | 6.00 | 8.4 | 36.1 | 16.9 |
| Qwen3.5-9B | No | 5.00 | 21.6 | 11.00 | 11.0 | 51.1 | 17.1 |
| Qwen3-VL-8B Thinking | Yes | 1.50 | 18.8 | 12.00 | 6.1 | 43.0 | 23.9 |
| Qwen3-VL-8B Instruct | No | 9.00 | 7.3 | 13.50 | 6.3 | 38.7 | 25.0 |
| Qwen3.5-9B BlenderFORGE | Yes | 40.00 | 5.7 | 27.50 | 5.1 | 7.3 | 7.8 |

## E Visual Assessment and Metric Validity

### E.1 Blinded Preference on Imagine-OOD

**Tasks and purpose.** Imagine-OOD evaluates semantic fulfillment on 80 manually written editing tasks, with 20 tasks in each of Placement, Shape-Key, Lighting, and Material. Each task pairs a human-written natural-language instruction with a Gemini-generated goal image. The four task families are the same as those used for training; this evaluation examines manually specified editing requests with generated visual goals, whereas the camera-reframing evaluation in Appendix D.1 tests a task family absent from training.

**Model comparison and judge inputs.** For each task, GPT-5.5 compares the edited outputs of Qwen3.5-9B Base and BlenderFORGE SFT+RL. Model identities are hidden, and left–right presentation positions are randomized and swapped. The judge rates semantic fulfillment on a 1–5 Likert scale. We use two judging conditions:

- **T:** the judge receives the start image, the human-written instruction, and the two edited output images. The instruction specifies the desired edit, and no goal image is supplied to the judge.
- **T+I:** the judge receives the same inputs plus the Gemini-generated goal image as a visual reference for the desired edit.

T and T+I thus differ only in the reference available to the judge; both include visual observations.

**Counting and reporting.** Each of the 80 tasks is assessed under both judging conditions, yielding 160 task–condition pairs. Table 13 records Base wins, SFT+RL wins, and ties. The Overall row pools the T and T+I rows, giving 160 comparisons over the same 80 tasks. The four per-family rows break down the 80 T+I comparisons.

The trained model wins 108 comparisons versus 18 for Base, a 6:1 ratio among non-tied outcomes; 34 comparisons are ties. N-CLIP preference agreement with GPT-5.5 is reported as 77.8% on decisive comparisons. These are auxiliary vision-model judgments and may miss unintended changes to non-target attributes.

Table 13: GPT-5.5 blinded pairwise preferences on Imagine-OOD. The overall row pools both conditions for the same 80 tasks.

| Group | Pairs | Base wins | SFT+RL wins | Ties |
|---|---|---|---|---|
| Overall | 160 | 18 | 108 | 34 |
| T | 80 | 10 | 48 | 22 |
| T+I | 80 | 8 | 60 | 12 |
| Placement / T+I | 20 | 0 | 17 | 3 |
| Shape-Key / T+I | 20 | 2 | 16 | 2 |
| Material / T+I | 20 | 2 | 15 | 3 |
| Lighting / T+I | 20 | 4 | 12 | 4 |

### E.2 Correlation with Automatic Metrics

We assess 800 historical edit cases, 200 per task family, sampled without conditioning on metric values. The cases span 120 task-specific run directories: 48 Placement, 27 Shape-Key, 23 Lighting, and 22 Material runs. Each case receives one pointwise GPT-5.5 rating. We compute Spearman correlations with raw metric distances and 95% bootstrap confidence intervals using 3,000 resamples. For pooled analyses, metric values are converted to within-task ranks before concatenation.

Table 14: Spearman correlations between GPT-5.5 ratings and automatic distances. Negative values indicate that higher visual ratings associate with lower distances.

| Task | Metric | $N$ | $\rho$ | 95% CI | $p$-value |
|---|---|---|---|---|---|
| Placement | P-Chamfer | 200 | $-0.504$ | $[-0.627, -0.368]$ | $2.8\times10^{-14}$ |
| | N-CLIP | 200 | $-0.366$ | $[-0.489, -0.237]$ | $1.0\times10^{-7}$ |
| | PL | 200 | $-0.186$ | $[-0.323, -0.049]$ | 0.008 |
| Shape-Key | ShapeKey-L1 | 200 | $-0.391$ | $[-0.501, -0.271]$ | $1.0\times10^{-8}$ |
| | N-CLIP | 200 | $-0.605$ | $[-0.696, -0.498]$ | $2.2\times10^{-21}$ |
| | PL | 200 | $-0.477$ | $[-0.587, -0.359]$ | $9.8\times10^{-13}$ |
| Lighting | N-CLIP | 200 | $-0.491$ | $[-0.600, -0.365]$ | $1.5\times10^{-13}$ |
| | PL | 200 | $-0.439$ | $[-0.564, -0.307]$ | $7.6\times10^{-11}$ |
| Material | N-CLIP | 200 | $-0.669$ | $[-0.745, -0.582]$ | $2.4\times10^{-27}$ |
| | PL | 200 | $-0.657$ | $[-0.739, -0.558]$ | $4.2\times10^{-26}$ |
| Pooled state | P-Chamfer/ShapeKey-L1 | 400 | $-0.396$ | $[-0.478, -0.310]$ | $1.7\times10^{-16}$ |
| Pooled N-CLIP | All four tasks | 800 | $-0.498$ | $[-0.554, -0.439]$ | $3.0\times10^{-51}$ |
| Pooled PL | All four tasks | 800 | $-0.413$ | $[-0.476, -0.350]$ | $2.6\times10^{-34}$ |

All ten task–metric correlations are negative with $p < 0.01$, but their magnitudes vary. P-Chamfer has the strongest association on Placement ($\rho = -0.504$), while N-CLIP has the strongest association on Shape-Key ($\rho = -0.605$). Placement PL is weakly correlated ($\rho = -0.186$). These associations support metric usefulness within the sampled cases, which are not guaranteed to be independent. The metrics remain proxies for semantic correctness, and these correlations do not test the significance of the main training comparisons.

## F Extended Limitations and Future Work

BlenderFORGE is designed as a controlled training substrate for executable 3D graphics editing, and its current scope leaves several important limitations.

**Task scope.** The benchmark focuses on four state-verifiable editing families: object placement, shape-key editing, lighting adjustment, and material modification. These tasks cover fundamental geometric and appearance edits, but they do not yet include more compositional workflows such as object insertion or deletion, procedural modeling, animation, physics-aware editing, or long-horizon scene redesign involving many coordinated changes.

**Goal-conditioned setting.** The four original in-distribution task families provide goal renderings together with the start-scene observations. This design enables deterministic measurement of editing progress and supports closed-loop reward computation, but it differs from open-ended natural-language editing where the desired final scene may be specified only textually or through partial constraints. The text-only camera-reframing evaluation begins to test underspecified goals, but its highest valid rate is only 35%. Imagine-OOD adds a visual assessment of manually specified edits. Unrestricted natural-language editing remains future work.

**Benchmark-induced specialization.** Training on perturbation-generated tasks may specialize the agent to the benchmark's task distribution; the camera-reframing and BlenderGym evaluations provide only preliminary evidence beyond it. The data pipeline itself is asset-agnostic and applies to domains with state-verifiable attributes; the current four-family scope reflects asset availability rather than a methodological ceiling.

**Environment specificity.** The current framework is built around Blender scenes and the `bpy` API. This makes the environment executable, inspectable, and reproducible, but it also means that the trained agent may learn Blender-specific conventions. Transfer to other DCC tools, CAD systems, game engines, or implicit 3D representations remains unexplored.

**Model and interaction scaling.** Our experiments train 8B and 9B open-weight backbones with bounded interaction horizons. Improvements on the second backbone support applicability beyond one model, while generalist–specialist differences remain task-dependent, most notably on Material. Longer trajectories and richer action spaces also increase rollout cost.

## G Compute Resources

Table 15 summarizes the resource configuration for the original 8B experiments. All GPUs are NVIDIA GeForce RTX 5090 cards with 32GB memory. CPU and RAM allocations follow a provisioning rule of approximately eight CPU cores per GPU and 16GB RAM per CPU core; upper bounds reflect peak provisioning rather than measured utilization.

Table 15: Resource estimates for the original 8B experiments. Training GPU-hours use allocated cards times elapsed time; sandbox usage is bounded by its maximum allocation. Evaluation is reported per run, not as a total over all models or additional studies.

| Stage | GPU usage | CPU/RAM | Time | Total compute |
|---|---|---|---|---|
| Dataset generation | 2 RTX 5090 GPUs for scene generation and rendering | 16 cores / 256GB | 24 hours | 48 GPU-hours |
| SFT | 8 RTX 5090 GPUs for unified mixed-task training | 64 cores / 1TB | 8 hours | 64 GPU-hours |
| AgentLoop-RL | 8 RTX 5090 GPUs for training; up to 8 for sandbox rendering | Up to 128 cores / 2TB | 10 days | 1920 training; at most 1920 sandbox GPU-hours |
| Evaluation | 2 RTX 5090 GPUs per evaluation run | 16 cores / 256GB per run | 6 hours per run | 12 GPU-hours per run |

The dataset generation stage used approximately 2TB of storage for intermediate scenes, renderings, metadata, and exported descriptors. Additional GPUs can be assigned to the Blender Sandbox during

RL and evaluation to increase rollout and rendering throughput without changing the model-side training configuration.

## H Release Policy and Asset Licenses

The test set is publicly available, and training data can be generated with the Dataset Pipeline. The release at `https://github.com/SenseTime-FVG/BlenderFORGE` provides data-generation, training, sandbox, and evaluation code, with links to the test set and fine-tuned model weights. It includes dataset metadata, split definitions, environment setup, evaluation configurations, and commands for verifying the main benchmark results with the released checkpoints. Restricted raw assets from upstream datasets such as 3D-FRONT or 3D-FUTURE are not redistributed; the data-access instructions explain how to obtain them from their original providers under the applicable terms and generate training tasks with the pipeline. Full dataset and model documentation covering licensing, intended use, and limitations will accompany the assets.

Table 16: Primary existing assets, models, and software used by BlenderFORGE. We cite the original sources in the main paper and will include license and terms-of-use notes in the release package.

| Asset / software | Use in BlenderFORGE | License / terms treatment |
|---|---|---|
| 3D-FRONT / 3D-FUTURE | Base scenes and object assets for placement | Used under provider terms; restricted raw assets are not redistributed. |
| Infinigen | Lighting scenes and material definitions | Code: BSD 3-Clause; preserve upstream notices and attribution. |
| Blender / `bpy` | Scene representation, editing API, rendering, and sandbox execution | Used under Blender GNU GPL terms; integration code retains notices. |
| Qwen3-14B / Qwen3-VL-8B / Qwen3.5-9B | Shape-key renaming and open-weight agent initialization | Apache 2.0 for these checkpoints; retain license and attribution notices. |
| GPT-5.2 / GPT-5.5 | Teacher trajectories and visual assessment | Provider API terms; no proprietary weights are redistributed. |
| Gemini | Imagine-OOD goal images and zero-shot baselines | Provider service terms; no proprietary weights are redistributed. |
| Swift | SFT framework | Apache 2.0; retain notices and attribution. |
| VeRL / HybridFlow | RL training infrastructure | Apache 2.0; retain notices and attribution. |
| CLIP | Visual distance for rewards and evaluation | MIT; retain notices and attribution. |

Shape-key and material assets described as prepared by human researchers are author-curated assets rather than crowdsourced data or participant-generated content. BlenderKit was used only as a design reference for author-created assets and is not redistributed as an asset source in the planned release.

## I Broader Impact and Release Safeguards

BlenderFORGE is intended to make programmatic 3D editing more accessible and reproducible. Potential positive impacts include reducing manual effort in film, game, architectural visualization, robotics simulation, educational content creation, and rapid prototyping workflows, especially for users who can describe desired edits more easily than they can implement detailed Blender Python scripts.

The same capability can also be misused to automate deceptive, infringing, or otherwise harmful 3D content workflows, or to modify assets in ways that violate ownership or platform terms. Our planned release will therefore include an intended-use and limitations statement, license and attribution requirements, and a usage policy that prohibits illegal, deceptive, privacy-invasive, or copyright-infringing uses. The release will not include personal data and will not redistribute restricted upstream raw assets; users must acquire such assets from the original providers under the applicable terms.

## 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 Introduction state the data-generation, sandbox, and SFT/RL training contributions, and Secs. 3–4 provide the corresponding methodology and empirical evidence. The claims distinguish the four training task families from additional transfer evaluations and comparisons of two open-weight backbones.

   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: Sec. 5 and Appendix F discuss task scope, partial base-mesh sharing for Shape-Key, camera-reframing validity, and model and interaction scaling. Appendices E.1 and E.2 discuss the scope and limitations of the VLM-based evaluations.

   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 theoretical theorems or proof-based claims. The equations in Sec. 3 define the task objective, evaluation metrics, and reward computations.

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: The public test set, released model checkpoints, and evaluation code, configurations, and commands provide a path to verify the main benchmark results without retraining. Training data can be generated with the Dataset Pipeline. Secs. 3.4 and 4.1 describe the evaluation protocol and model settings; Appendix H provides the release entry point and reproduction resources.

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: [Yes]

   Justification: We provide the test set, model weights, data-generation, training, sandbox, and evaluation method, together with environment setup, evaluation configurations. Training data can be generated with the released pipeline using upstream assets obtained under their respective terms. Appendix H provides the release entry point and data-access instructions.

   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. 3.4 defines the evaluation protocol and metrics, Sec. 4.1 reports model, SFT, RL, and inference settings, and the Appendix provides prompt templates, render settings, and dataset split configurations.

   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: Appendix E.2 reports 95% bootstrap confidence intervals and p-values for the VLM–metric correlations, and Appendix D.1 reports three camera-reframing repeats per task and model. The main model comparisons do not report error bars, confidence intervals, or significance tests.

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: [No]

   Justification: Sec. 4.1 and Appendix G report hardware and resource estimates for the original 8B experiments. A complete compute accounting for the additional backbone, transfer, and VLM evaluations is not reported.

   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 does not involve human subjects or personal data, and the release policy in Appendix H is designed to respect upstream asset licenses and redistribution restrictions. Appendix I describes intended use, risks, and safeguards for responsible release.

   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: Appendix I discusses positive impacts for 3D content creation, simulation, education, and accessibility, as well as potential misuse in deceptive or infringing 3D content workflows.

    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: [Yes]

    Justification: Appendix I discusses intended use and misuse risks, and describes planned safeguards for the dataset and model release, including usage restrictions and exclusion of personal data and restricted upstream raw assets.

    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: [Yes]

    Justification: Appendix H lists the upstream assets, models, and software, their licenses or provider terms, and redistribution constraints. It includes both trained backbones and the models used for teacher trajectories, generated goal images, and visual assessment; restricted upstream raw assets are excluded from the planned release.

    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: [No]

    Justification: The paper documents the dataset formulation, sandbox, training pipeline, and evaluation protocol. The test set, code, model weights, and reproduction instructions are available, and training data can be generated with the Dataset Pipeline. Full dataset and model documentation covering licensing, intended use, and limitations remains planned (Appendix H).

    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 work does not involve crowdsourcing or human-subject experiments. References to human researchers in the paper refer only to the authors' internal curation and predefined asset preparation.

    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: No crowdsourcing or human-subject study was conducted, so IRB or equivalent approval 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 method and evaluations use Qwen3-14B for shape-key renaming, GPT-5.2 for teacher trajectories, Qwen3-VL-8B and Qwen3.5-9B as editing agents, Gemini for generating Imagine-OOD goal images, and GPT-5.5 for visual preference and metric-validity assessments, as described in Secs. 3.2, 3.5, 4.1, and Appendix E.

    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.