MUSE: AGENTIC 3D SCENE AUTHORING VIA MEMORY-GROUNDED INCREMENTAL REQUIREMENT SATISFACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-driven 3D scene generation is a promising technique for digital content creation, embodied AI simulation, and interactive design, yet practical workflows often require refining, extending, or correcting existing scenes while preserving non-target content. Existing methods can produce realistic and structurally plausible scenes, but they generally lack editability with requirement-level state tracking, so part-level failures often lead to full-scene regeneration or manual intervention. To tackle this challenge, we formulate controllable 3D scene authoring as incremental requirement satisfaction, unifying construction and editing. We present MUSE, a memory-grounded multi-agent framework in which an Architect compiles instructions into structured requirements, a Sculptor executes local scene operations, and an Inspector verifies each step while updating Working, Scene, and Skill Memory. To evaluate requirement-level controllability and preservation-aware editing, we introduce AuthorBench, comprising 145 constrained construction cases and a 1,584-case editing pool with external structured checks. Against the strongest baseline, MUSE improves construction All-Goal success from 37.9% to 80.7% and surface-constraint fulfillment from 35.0% to 93.3%; on a stratified 240-case editing split, it achieves 76.7% All-Goal success, a 99.9% preservation rate, and a 0.6% unintended change rate. Human evaluations and navigation-proxy tests further support stronger intent alignment and spatial stability, while ablations validate the memory design. Together, these results establish MUSE as an effective framework for controllable 3D scene authoring.

1 INTRODUCTION

Text-driven 3D scene generation enables users to create complex three-dimensional environments from natural language, with broad applications in simulation and game design (Yang et al., 2024a;c; Fu et al., 2024; Gu et al., 2025; Xia et al., 2026). However, many practical settings, such as embodied AI training and interactive design, require more than one-shot synthesis: users often need to add, remove, or adjust local content in an existing scene while keeping non-target regions structurally and semantically stable. Supporting this workflow requires controllable generation, where a system maintains an explicit, persistent representation of the scene state and generation intent to guide subsequent local operations.

Despite substantial progress in spatial modeling, existing text-driven 3D scene generation methods still fall short of controllable generation. They can be grouped into three families. Data-driven methods (Paschalidou et al., 2021; Wang et al., 2021; Dhamo et al., 2021; Tang et al., 2024; Lin and Mu, 2024) rely on implicit generation processes, making it difficult to explicitly control local structures. LLM-based planning methods (Feng et al., 2023; Yang et al., 2024c; Sun et al., 2025) improve open-vocabulary flexibility, but rely on one-shot context without persistently tracking execution history or unmet constraints. Agentic methods introduce tool use and closed-loop reflection (Yang et al., 2025; Pfaff et al., 2026; Xia et al., 2026), yet most of them still optimize within a single generation round rather than manage scene state across interactions. Consequently, existing systems generally lack explicit modeling and binding of requirement-level states, making it difficult to repair local failures without disturbing non-target content. In practice, when a local result is unsatisfactory, they often resort to global regeneration instead of truly controllable local revision.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

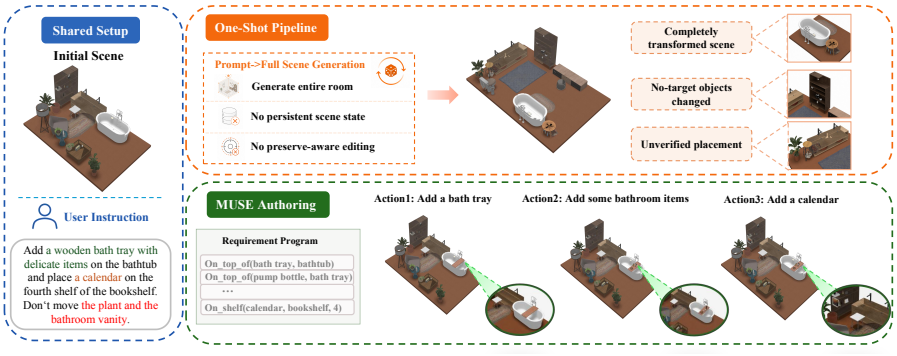


Figure 1: **From one-shot regeneration to persistent 3D scene authoring.** One-shot pipelines regenerate the whole scene and may disturb non-target regions, whereas MUSE performs verified local actions over persistent memory to preserve satisfied content.

Therefore, scene editability is central to controllable 3D generation. Practical workflows often refine, extend, or correct an existing scene rather than start from an empty one. An authoring system should reuse semantic relations, satisfied bindings, and intermediate scene state, while applying atomic authoring actions only to the target region and preserving non-target content. We therefore formulate 3D scene synthesis as a persistent authoring process that unifies initial construction and fine-grained editing through requirement decomposition, local execution, verification, and memory.

Realizing this incremental authoring process requires structured memory. *Working Memory* tracks requirement status, diagnostics, and protected bindings to preserve satisfied content; *Scene Memory* maintains a persistent 3D hierarchy for stepwise spatial reasoning; and *Skill Memory* stores reusable decomposition patterns for recurring scene motifs, reducing repeated decomposition errors.

Based on this design, we present MUSE, a multi-agent framework for unified 3D scene authoring. The *Architect* compiles instructions into structured requirements, the *Sculptor* executes local operations, and the *Inspector* verifies their outcomes. They coordinate through a shared requirement-state protocol that tracks completion, binds satisfied requirements to concrete scene objects, constrains subsequent operations through protection sets, and re-verifies requirements affected by edits. Updated Scene Memory then carries scene context across instructions.

We introduce AuthorBench, comprising 145 construction and 1,584 editing cases with structured spatial and semantic checks. MUSE improves construction All-Goal success from 37.9% to **80.7%** and surface fulfillment from 35.0% to **93.3%**; in editing, it achieves **76.7%** All-Goal success, **99.9%** preservation, and **0.6%** unintended changes. Human evaluations, navigation-proxy tests, and ablations support its alignment, spatial stability, and memory design. Our contributions are:

- We formulate controllable 3D scene authoring as incremental requirement satisfaction over persistent scene state, unifying construction and preservation-aware editing at the requirement level.
- We present MUSE, a multi-agent framework in which Architect, Sculptor, and Inspector coordinate through a shared requirement-state protocol and Working, Scene, and Skill Memory to track progress, protect bindings, schedule dependencies, and re-verify affected requirements.
- We introduce AuthorBench to evaluate construction and preservation-aware editing at the requirement level, and validate MUSE against recent baselines through automated metrics, human studies, navigation-proxy tests, and ablations.

Interactive demo. Examples of incremental scene editing are available at `https://anonymousauthor608.github.io/muse_demo/`.

2 Related Work

Text-driven 3D scene synthesis. Data-driven methods (Paschalidou et al., 2021; Wang et al., 2021; Dhamo et al., 2021; Tang et al., 2024; Lin and Mu, 2024) learn object distributions and relations from indoor scene datasets such as 3D-FRONT (Fu et al., 2021a), while LLM-based planners improve open-vocabulary construction through spatial programs or layout optimization (Feng et al., 2023;

Yang et al., 2024c; Sun et al., 2025). Agentic frameworks further introduce tool use and iterative refinement (Yang et al., 2025; Pfaff et al., 2026; Xia et al., 2026). However, these methods generally operate within a single generation episode and lack persistent requirement-level state, making it difficult to isolate failed constraints, revise only the affected regions, and preserve previous results during continuous authoring. To address this issue, we introduce controllable 3D scene authoring, which unifies construction and editing at the requirement level.

Language-guided 3D scene editing. Interactive and generative editors support dialogue-based object manipulation, visual-feedback refinement, or autoregressive scene revision (Yang et al., 2024b; Wang et al., 2024a; Bucher and Armeni, 2025). Structured approaches instead update scene graphs or plan symbolic actions to improve spatial coherence and physical feasibility (Zhen et al., 2026; Noh et al., 2026). These formulations focus on the current dialogue, revised layout, or edit goal. MUSE treats construction and editing as incremental requirement satisfaction, tracking progress and protection within each instruction while carrying scene structure forward to subsequent instructions.

Memory and structured state in LLM agents. Memory mechanisms support long-horizon reasoning through spatial maps and scene graphs, episodic or verbal traces, and reusable skill libraries (Chaplot et al., 2020; Gu et al., 2024; Wang et al., 2024b; Park et al., 2023; Shinn et al., 2023; Yao et al., 2023; Huang et al., 2023; Sumers et al., 2024). Recent scene-synthesis agents also support iterative feedback or multi-turn interaction during generation (Yang et al., 2025; Pfaff et al., 2026; Xia et al., 2026; Zhao et al., 2026). However, although systems such as SceneReVis incorporate such feedback, their persistent state is still largely represented through dialogue history or model context rather than task-structured memory. This limits explicit tracking of satisfied requirements, protected content, and safe local modifications. We therefore introduce task-structured memory for controllable 3D scene authoring, combining persistent scene structure with explicit tracking of requirement progress and protected content during each instruction's execution.

3 METHOD

We present MUSE, a multi-agent framework that performs requirement-level planning and verification over persistent scene state. Given a complex construction prompt or targeted edit instruction, it compiles the input into a structured requirement program, executes requirements locally, verifies each step, and updates memory to protect satisfied content. Figure 2 presents the full pipeline. We then formalize the task and describe the Architect, Sculptor, and Inspector modules.

3.1 TASK FORMULATION

We formulate *controllable 3D scene authoring* as incremental requirement satisfaction over a persistent scene state. At interaction t, the system receives the current scene $\mathcal{S}_t$, a natural-language input I_t—a construction prompt or targeted edit instruction—and memory state $\mathcal{M}_t$. The input is compiled into a structured requirement program $\mathcal{B}_t$ containing atomic requirements $\{r_i\}_{i=1}^{n}$, which drives a sequence of atomic operations

$$\Delta_t = (\delta_1, \ldots, \delta_m), \qquad \mathcal{S}_{t+1} = \mathcal{S}_t \oplus \Delta_t. \tag{1}$$

Here, $\oplus$ denotes their sequential execution, and each δ_j represents object insertion, removal, transformation, or support-aware placement. Rather than jointly solving the dense requirement program, the system incrementally schedules and verifies requirements until all r_i are satisfied.

The incremental loop uses a three-layer memory state $\mathcal{M}_t = (\mathcal{W}_t, \mathcal{G}_t, \mathcal{K})$. Working Memory $\mathcal{W}_t$ tracks the current focus, requirement status, role bindings, and protection sets; Scene Memory $\mathcal{G}_t$ stores the persistent room shell, support surfaces, functional zones, and object graph; and Skill Memory $\mathcal{K}$ provides reusable motif priors, decomposition cues, and failure warnings. Scene Memory persists with the updated scene across instructions, whereas Working Memory and its protection sets are initialized for each instruction. Earlier requirements must be carried into the current program to remain explicitly enforced during subsequent edits.

Construction begins from an empty or partially specified room shell and incrementally realizes a new scene. Editing begins from an existing scene and adds preservation constraints to protect non-target content. Both modes share requirement compilation, local execution, and verification; Appendix A.1 provides the full pseudocode.

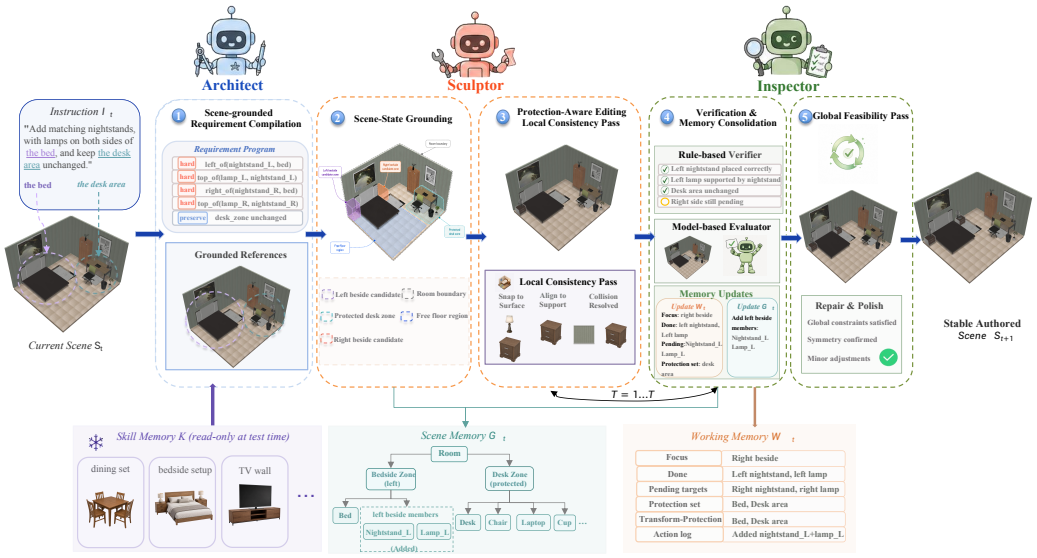


Figure 2: **Overview of MUSE.** At interaction t, **(1)** the Architect compiles I_t using Scene and Skill Memory; **(2)** the Sculptor grounds $\mathcal{S}_t$ and **(3)** executes protection-aware local edits; **(4)** the Inspector verifies requirements and updates memory; and **(5)** a global feasibility pass resolves residual conflicts, yielding $\mathcal{S}_{t+1}$.

3.2 Architect: Scene-Grounded Compilation

Dense spatial instructions can overload LLM reasoning, leading to dropped, hallucinated, or unverified constraints. The Architect combines I_t with Scene Memory $\mathcal{G}_t$ and decomposition priors retrieved from Skill Memory $\mathcal{K}$ to compile a structured requirement program $\mathcal{B}_t$. Its typed and independently trackable requirements provide explicit targets for execution, retry, verification, and preservation.

Requirement program structure. The requirement program $\mathcal{B}_t = (I_t, \textit{room_type}, \{r_1, \ldots, r_n\}, \mathbf{g}, \mathbf{s})$ contains atomic requirements r_i, scene-aware constraints $\mathbf{g}$, and a style profile $\mathbf{s}$. Each requirement records a type in $\{\texttt{hard}, \texttt{edit}, \texttt{preserve}, \texttt{forbid}\}$, a priority, and an optional checkable predicate ϕ_i. Requirements with ϕ_i are verified through deterministic spatial or counting checks, while residual semantic and appearance aspects receive model-based judgment. Using the objects and spatial relations stored in Scene Memory $\mathcal{G}_t$, the Architect resolves contextual references and adds prerequisite requirements needed for execution; Appendix A.2.1 and Appendix A.2.3 provide the full schema and enrichment protocol.

Skill memory $\mathcal{K}$: motif-guided decomposition. Compositional instructions often contain recurring motifs, such as a bedside setup or chairs arranged around a dining table. Inspired by HSM (Pun et al., 2026), Skill Memory stores ten manually authored cards for common indoor motifs, containing trigger phrases, expected anchors, and decomposition or role-binding cues. These cards encode category-level priors and contain no AuthorBench case identifiers, reference layouts, or case-specific target values. Retrieved cards guide the Architect in decomposing recurring patterns into separate, checkable requirements. After AuthorBench was finalized, Skill Memory was frozen together with MUSE and remained unchanged during evaluation. Outside evaluation, the library can be expanded offline without model parameter updates; details are provided in Appendix A.2.2.

3.3 Sculptor: Incremental Requirement Execution

Incremental execution risks regressing previously satisfied content. The Sculptor executes one requirement at a time over an explicit room state while protecting satisfied role bindings, enabling preservation-aware local revision.

Scene-state grounding. Spatial tools require explicit geometric domains, such as support surfaces, wall domains, and free-floor regions that are not directly available from a raw object list. Before planning begins, the Sculptor constructs an initialized room state $\mathcal{R}_0$ from $\mathcal{S}_t$ and $\mathcal{G}_t$ to register these necessary structural anchors. This grounding step allows later declarative spatial intent to be mapped stably into concrete placements and transformations.

Requirement-grounded planning. At each inner turn τ, Working Memory $\mathcal{W}_\tau$ records the current focus, requirement status, role bindings, and diagnostics. The protected set P_τ forbids deletion or replacement of anchors and satisfied role bindings, while the transform-protected set $P_\tau^T \supseteq P_\tau$ additionally forbids movement, rotation, and resizing. Verification updates both sets with newly satisfied role bindings so that subsequent actions preserve established progress. Appendix A.2.4 provides the full Working Memory schema.

Execution and local consistency pass. The Sculptor equips the planner with shared atomic authoring actions, encompassing both *declarative* tools for expressing spatial relations and *imperative* tools for direct object manipulation. The executor grounds declarative calls into geometric operations over the registered domains, allowing the planner to express spatial intent in a coordinate-free manner. Crucially, if the planner proposes an action that violates the protection sets P_τ or P_τ^T, the executor preemptively blocks it before any scene modification occurs.

After execution, an automated *local consistency pass* resolves newly introduced collisions, support-placement offsets, and minor interpenetrations within the edited neighborhood. Because both preemptive blocking and local repair are constrained by P_τ and P_τ^T, modifications remain highly localized, yielding a stable geometry for subsequent verification. Tool definitions and fallback behaviors are detailed in Appendix A.3.2; re-verification rules are in Appendix A.3.3.

3.4 Inspector: verification and memory consolidation

The Inspector verifies each requirement and consolidates the results into persistent memory, providing retry, scheduling, and termination signals for subsequent execution.

Compositional verification. For requirements with a checkable predicate ϕ_i, a *rule-based verifier* evaluates object counts, spatial relations, and support or wall placement. A *model-based feedback agent* instead judges residual perceptual or semantic aspects from the repaired scene state and rendered views. Deterministic results take precedence, so model-based feedback complements rather than overrides verifiable constraints. Feedback structures and reconciliation rules are detailed in Appendix A.3.3.

Memory consolidation and scheduling. Upon verification, the Inspector concurrently updates two memory layers. First, it updates Working Memory $\mathcal{W}_\tau$ with newly satisfied targets, expanding the nested protection sets (P_τ, P_τ^T) accordingly. Simultaneously, it consolidates the validated layout into Scene Memory $\mathcal{G}_\tau$, establishing a persistent structural graph (e.g., support hierarchies, zones) for future interactions. This dual-update ensures that later edits inherit a structured state rather than a raw object list. The scheduler then selects the next pending requirement, dynamically adjusting the execution order to prevent stalling on difficult constraints. Appendices A.3.1 and A.2.5 provide the scheduling and Scene Memory details.

Loop termination and global feasibility pass. The inner loop concludes once all scheduled requirements are verified or when further execution can no longer yield productive progress. Finally, the Inspector executes a *preservation-aware global feasibility pass* that resolves any residual scene-level collisions and applies lightweight stability fixes. This yields a clean, structurally sound scene state ready for the next user instruction without compromising protected non-target content.

4 AuthorBench

Existing 3D scene-synthesis benchmarks mainly evaluate holistic scene quality, such as physical plausibility, visual realism, and semantic alignment (Yang et al., 2025), but they do not reveal whether each user constraint is satisfied and rarely test realistic local editing with preservation constraints. To

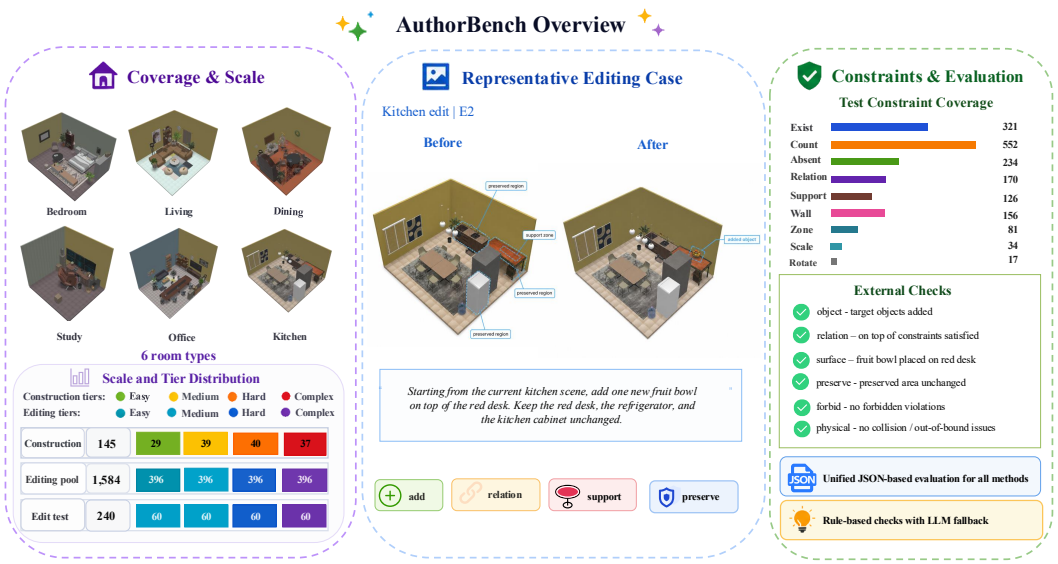


Figure 3: **AuthorBench overview.** AuthorBench unifies scene construction and preservation-aware editing under requirement-level evaluation. **(Left)** Dataset coverage across six room types and complexity tiers. **(Middle)** A representative editing case decomposing a user instruction into verifiable atomic constraints. **(Right)** The evaluation taxonomy, pairing each case with typed external checks.

fill this gap, we introduce **AuthorBench**. As illustrated in Figure 3, AuthorBench shifts the evaluation paradigm from holistic scene scoring to individual requirement verification. AuthorBench was co-developed with MUSE. Construction checks derive from fixed natural-language specifications, and editing checks are generated from frozen source-scene metadata and target identities. AuthorBench was finalized first; MUSE and Skill Memory were then frozen. Final scenes are evaluated externally using deterministic predicates for programmable constraints and an independent visual-semantic judge for semantic constraints, without relying on MUSE's internal Inspector.

Subsets. AuthorBench contains 145 construction cases and 1584 single-instruction editing cases across six room types: bedroom, living room, dining room, study, office, and kitchen. Construction starts from an empty room; editing starts from source scenes drawn from Edit-As-Act (Noh et al., 2026) and Imaginarium (Zhu et al., 2025), and specifically requires preservation constraints to test explicit local modifications rather than full-scene regeneration. All AuthorBench editing experiments use a fixed 240-case stratified split balanced by room type and Easy/Medium/Hard/Complex tiers; Appendix B.1 gives the sampling rules and tier definitions.

Annotation and metrics. Annotation compiles natural-language requirements into typed requirement checks and then applies human review. Checks cover object, relation, surface, zone, preserve, and forbid types, enabling AuthorBench to evaluate goal completion, local preservation, and physical validity. Main experiments report family-level *Goal Fulfillment (GF)*, their unweighted mean (*Macro*), strict *All-Goal* success, complexity retention (*Ret.* = Complex/Easy GF), editing locality via *Preservation Rate (PR)* and *Unintended Change Rate (UCR)*, and physical validity via average *Out-Of-Bounds (OOB)* and *Collision (Col.)* counts. For construction, *All-Goal (no Surf.)* recomputes strict success after excluding surface-family checks; Appendix B.2.1 provides full definitions.

5 Experiments

Our evaluation assesses *scene construction* (§5.1), *preservation-aware editing* (§5.2), and *additional analyses* covering external validation, cross-backend transfer, human preference, component ablations, and downstream utility (§5.3).

Setup & Baselines. We evaluate released systems under the unified AuthorBench protocol (§4). Construction uses 145 cases starting from empty room shells; editing uses the fixed 240-case stratified

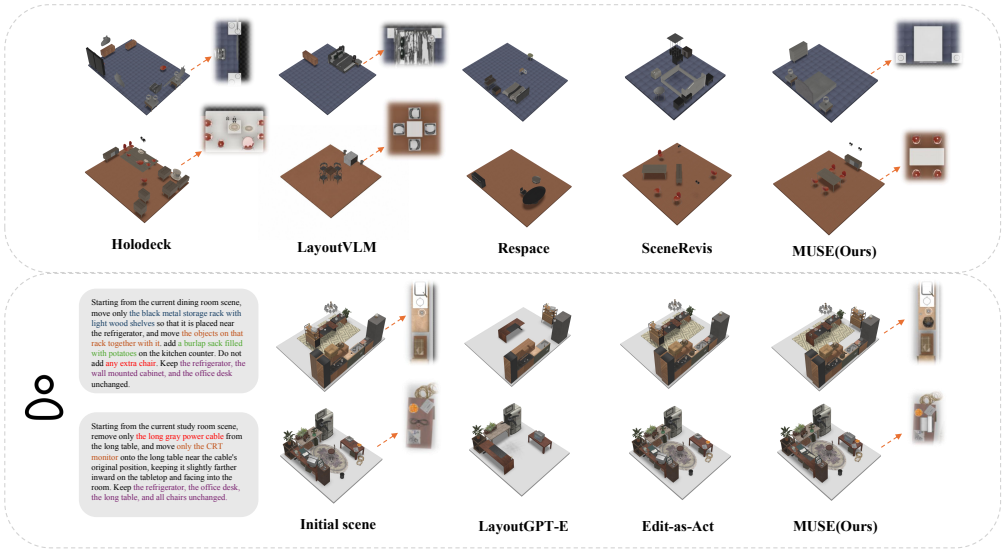


Figure 4: **Qualitative comparisons.** Top: MUSE improves scene layout; bottom: it completes local edits while preserving non-target content.

split. We compare MUSE against LayoutGPT (Feng et al., 2023), Holodeck (Yang et al., 2024c), LayoutVLM (Sun et al., 2025), ReSpace (Bucher and Armeni, 2025), and SceneReVis (Zhao et al., 2026) for construction, and against LayoutGPT-E, Edit-As-Act (Noh et al., 2026), SceneReVis, and ReSpace-E for editing. **Metrics** strictly follow §4; results are averaged across their respective evaluation subsets.

Implementation details. All methods using a large language or vision-language model, including MUSE and the baselines, use MiMo-V2.5 (Xiaomi MiMo, 2026) as the common backend in the main AuthorBench comparisons. Native outputs are converted once into the common AuthorBench format for identical rendering and rule-based evaluation. Explicit cross-backend experiments are reported separately. We adapt LayoutGPT and ReSpace to editing as LayoutGPT-E and ReSpace-E: LayoutGPT-E conditions on the serialized source scene, while ReSpace-E uses its native structured-scene edit pipeline. Implementation, asset, and prompt details are in Appendices B.3, A.4.2, and A.4.3.

5.1 Scene Construction

MUSE achieves 80.7% All-Goal success, more than doubling the 37.9% achieved by the strongest baseline, LayoutGPT (Table 1). Its largest gain occurs on surface constraints (93.3% vs. 35.0%), yet the advantage persists after removing all surface-family checks (87.6% vs. 80.0% All-Goal), showing that the improvement extends beyond surface placement. MUSE also retains 96.6% of its Easy-tier performance on dense Complex cases while maintaining near-zero physical errors (OOB 0.0, Col. 0.1). Qualitatively, the upper panel of Figure 4 shows that baselines often retrieve the requested objects but fail to maintain support relations and room-level groupings, whereas MUSE produces coherent arrangements and correctly grounds objects on their intended surfaces.

5.2 Preservation-Aware Scene Editing

Table 2 reveals a trade-off between task completion and editing locality. LayoutGPT-E attains high object fulfillment (86.0%) but largely overwrites the source scene (PR 25.1%, UCR 95.7%); local editing baselines preserve more context but remain below 12% *All-Goal* success. In contrast, MUSE achieves the highest completion (Macro 81.2, All-Goal 76.7) with near-perfect preservation (PR 99.9%, UCR 0.6%) and near-zero physical errors. Qualitatively, the lower panel of Figure 4 shows that baselines either alter protected content or leave requested operations incomplete, whereas MUSE completes the edits while restricting changes to the specified targets.

Table 1: **Main construction results on AuthorBench.** All-Goal (no Surf.) excludes surface-family checks. Bold denotes the best value in each column, including ties.

Method	Goal Fulfillment ↑					All-Goal↑	All-Goal (no Surf.)	Ret. ↑	Phys. Validity ↓	
	Obj.	Rel.	Surf.	Zone	Macro				OOB	Col.
LayoutGPT	98.4	83.8	35.0	79.0	74.1	37.9	80.0	82.7	1.4	2.3
Holodeck	83.3	57.6	21.5	30.9	48.3	14.5	24.8	63.9	**0.0**	**0.0**
LayoutVLM	98.0	69.7	22.1	59.3	62.3	26.9	62.8	81.7	**0.0**	**0.0**
ReSpace	71.9	36.4	8.6	32.1	37.3	11.0	17.2	57.5	0.4	0.5
SceneReVis	46.9	17.2	9.2	8.6	20.5	2.1	2.1	65.6	0.3	0.9
MUSE	**99.6**	**96.0**	**93.3**	**85.2**	**93.5**	**80.7**	**87.6**	**96.6**	**0.0**	0.1

Table 2: **Main preservation-aware editing results on AuthorBench.** Bold denotes the best value in each column, including ties.

Method	Goal Fulfillment ↑				All-Goal↑	Ret. ↑	Edit Locality		Physical Validity ↓	
	Obj.	Rel.	Surf.	Macro			PR↑	UCR↓	OOB	Col.
LayoutGPT-E	**86.0**	21.1	19.3	42.2	36.2	77.5	25.1	95.7	1.8	5.2
Edit-As-Act	30.6	14.1	5.0	16.6	5.4	102.3	84.9	17.8	0.2	0.5
SceneReVis	33.6	28.2	4.2	22.0	10.4	80.1	92.6	5.9	0.8	2.1
ReSpace-E	46.3	19.7	5.9	24.0	11.2	94.6	**100.0**	1.9	1.2	3.5
MUSE	79.0	**74.7**	**89.9**	**81.2**	**76.7**	**104.8**	99.9	**0.6**	**0.1**	**0.3**

5.3 Additional Analyses

Human evaluation. To complement the quantitative evaluation, we invited 50 participants to compare MUSE with Edit-As-Act, ReSpace-E, and SceneReVis (Table 5). Participants assessed instruction satisfaction, preservation, physical plausibility, and overall preference. MUSE achieves the highest mean ratings across all three dimensions and the highest overall preference rate (63.4%). Appendix B.4 provides the participant information, evaluation protocol, rating criteria, consent procedure, and ethics classification.

External benchmark validation. Table 3 evaluates transfer beyond AuthorBench for both editing and construction. On all 63 E2A-Bench editing tasks (Noh et al., 2026), MUSE improves room-macro Instruction Fulfillment (IF), Scene Coherence (SC), and Physical Plausibility (PP) over Edit-As-Act by 1.3, 0.7, and 2.4 percentage points, respectively. Under the adapted SceneEval-94 construction protocol (Tam et al., 2026), MUSE achieves the best Object Count (CNT), Support (SUP), and scene-level Collision (COLsc) scores, while Holodeck remains stronger on Accessibility (ACC). Accessibility is not explicitly optimized by the current requirement and verification loop; however, MUSE can continue editing the persistent scene to remove obstructions or adjust layouts when accessibility is requested. Full metrics and protocols are provided in Appendix C.2.

Cross-backend generalization. Replacing MiMo-V2.5 with five VLMs under fixed prompts, interfaces, memory, budgets, and scoring yields 86.9–92.7 construction Macro and stable editing locality (99.6–99.9% PR; 0.6–0.7% UCR). Even the 7B-parameter Qwen2.5-VL backend reaches 86.9 construction Macro and 99.6% PR, showing that the structured workflow transfers to smaller VLMs. Editing completion varies more, indicating that backend capacity mainly affects difficult edit execution. See Appendix C.3.1 for full results.

Component ablation. History-only planning reduces construction and editing Macro by 5.3 and 7.4 points while largely preserving locality (PR 99.2%, UCR 1.1%), showing that chronological action/feedback history does not fully replace planner-side structured state. Disabling protection in the same setting further lowers editing Macro to 70.1 and sharply degrades locality (PR 90.1%, UCR 7.1%), consistent with the direct protection-set ablation. Single-pass execution causes the largest completion loss, while removing Skill Memory produces the largest Complex-tier drop among the memory ablations.

Table 3: **External benchmarks.** (a) E2A-Bench: 63 edits (room-macro); (b) adapted SceneEval: 94 single-room cases. See Appendix C.2.

(a) E2A-Bench editing

Method	IF↑	SC↑	PP↑
LayoutGPT-E	44.5	42.2	78.3
Edit-As-Act	69.1	86.6	91.6
MUSE	**70.4**	**87.3**	**94.0**

(b) Adapted SceneEval-94 construction

Method	CNT↑	SUP↑	COLsc↓	ACC↑
LayoutGPT	20.69	28.63	30.00	46.69
Holodeck	42.72	61.31	73.00	**90.04**
MUSE	**55.41**	**83.83**	**25.29**	79.71

Table 4: **Component ablation.** History-only uses the scene, requirements, and action/feedback history instead of structured state; its second variant disables protection sets.

	Construction		Editing		
Variant	Macro↑	Complex↑	Macro↑	PR↑	UCR↓
MUSE (full)	**93.5**	**95.9**	**81.2**	**99.9**	**0.6**
history-only planning	88.2	93.4	73.8	99.2	1.1
history-only w/o protection	86.3	90.8	70.1	90.1	7.1
w/o requirement tracking	90.9	93.7	76.4	97.5	2.5
w/o protection sets	91.0	94.4	75.3	90.2	9.7
w/o Scene Memory	89.4	92.1	73.2	95.3	4.2
w/o Skill Memory	88.7	89.6	71.4	96.5	3.4
single-pass	14.5	25.5	28.5	76.2	19.8

Navigation curriculum generation. We compare incremental MUSE with independently regenerating each stage over 100 three-stage curricula. Both produce navigable, collision-free final scenes, but MUSE better preserves cross-stage progression, improving difficulty monotonicity from 0.394 to 0.850 and free-space IoU from 0.649 to 0.978. This suggests that incremental authoring produces more spatially consistent curricula for navigation research. The metrics are computed from exported layouts using a grid-based proxy rather than a trained navigation policy.

Table 5: **Human evaluation.** Mean ratings (1–5) and 4-way overall preference (%).

Method	Satis.↑	Preserv.↑	Plaus.↑	Pref.↑
Edit-As-Act	2.5	4.1	4.3	16.2%
SceneReVis	3.2	3.5	3.8	11.0%
ReSpace-E	3.6	3.8	3.5	9.4%
MUSE	**4.5**	**4.8**	**4.6**	**63.4%**

Table 6: **Navigation curricula.** Results over 100 three-stage curricula; @3 denotes the final stage.

Method	Mono.↑	FS-IoU↑	Walk@3↑	Reach@3↑	Path@3↓	Col./OOB↓
Stage-wise	0.394	0.649	**0.878**	**0.999**	2.183	0.0/0.0
MUSE	**0.850**	**0.978**	0.868	0.992	**1.736**	0.0/0.0

6 Conclusion

In this paper, we formulated controllable 3D scene authoring as incremental requirement satisfaction and introduced MUSE, a memory-grounded multi-agent framework for requirement compilation, iterative execution, and stepwise verification. Together with AuthorBench, a requirement-level benchmark for construction and preservation-aware editing, our experiments show that MUSE outperforms one-shot baselines in dense construction and mitigates the editing completion-locality trade-off, achieving strong task completion with near-perfect structural stability (PR 99.9%, UCR 0.6%). Human evaluations and downstream navigation-proxy tests further support its alignment with user intent and spatial stability. Detailed limitations and future directions are discussed in Appendix E.

Ethics Statement

Our human evaluation involved voluntary, anonymous, non-invasive assessments of synthetic 3D scenes by 50 adult participants. Reading the study information and proceeding signaled consent. The

study involved no physical or psychological intervention, deception, or collection of sensitive personal information or participant identifiers, making it a low-risk visual-evaluation task. Details of the study procedures are provided in Appendix B.4, and broader impacts are discussed in Appendix E.1.

Reproducibility Statement

We document the benchmark splits, evaluation metrics, baseline adapters, model configurations, and prompting procedures in the appendix. Appendices B.1, B.2.1, B.3, A.4.1, and A.4.3 provide details of the experimental setup and evaluation. The submitted supplementary materials include our implementation code and a demonstration video.

Statement on AI Use

The authors used generative AI tools to assist with manuscript drafting, language polishing, grammar correction, and improvements to clarity and organization, as well as code development, data analysis, visualization, and interpretation of results. Within the research methodology, language and vision-language models support requirement compilation, scene-operation planning, and feedback-based verification in MUSE. Their roles, backend configurations, and use in the evaluated methods are described in the methodology and experimental settings. The authors take responsibility for the accuracy and integrity of the final manuscript and all reported results.

References

Bucher, M. J. J., & Armeni, I. (2025). ReSpace: Text-Driven Autoregressive 3D Indoor Scene Synthesis and Editing. *arXiv preprint arXiv:2506.02459*.

Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., & Salakhutdinov, R. (2020). Learning to explore using active neural SLAM. In *Proc. ICLR*.

Dhamo, H., Manhardt, F., Navab, N., & Tombari, F. (2021). Graph-to-3D: End-to-End Generation and Manipulation of 3D Scenes Using Scene Graphs. In *Proc. ICCV*.

Feng, W., Zhu, W., Fu, T.-J., Jampani, V., Akula, A., He, X., Basu, S., Wang, X. E., & Wang, W. Y. (2023). LayoutGPT: Compositional Visual Planning and Generation with Large Language Models. In *Proc. NeurIPS*.

Fu, H., Cai, B., Gao, L., Zhang, L.-X., Wang, J., Li, C., Zeng, Q., Sun, C., Jia, R., Zhao, B., & Zhang, H. (2021a). 3D-FRONT: 3D Furnished Rooms With layOuts and semaNTics. In *Proc. ICCV*.

Fu, H., Jia, R., Gao, L., Gong, M., Zhao, B., Maybank, S., & Tao, D. (2021b). 3D-FUTURE: 3D Furniture Shape with TextURE. *International Journal of Computer Vision*, 129, pp. 3313–3337.

Fu, R., Wen, Z., Liu, Z., & Sridhar, S. (2024). AnyHome: Open-vocabulary generation of structured and textured 3D homes. In *Proc. ECCV*.

Gu, Q., Kuwajerwala, A., Morin, S., Jatavallabhula, K. M., Sen, B., Agarwal, A., Rivera, C., Paul, W., Ellis, K., Chellappa, R., Gan, C., de Melo, C. M., Tenenbaum, J. B., Torralba, A., Shkurti, F., & Paull, L. (2024). ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning. In *Proc. ICRA*.

Gu, Z., Cui, Y., Li, Z., Wei, F., Ge, Y., Gu, J., Liu, M.-Y., Davis, A., & Ding, Y. (2025). ArtiScene: Language-driven artistic 3D scene generation through image intermediary. In *Proc. CVPR*.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Jackson, T., Brown, N., Luu, L., Levine, S., Hausman, K., & Ichter, B. (2023). Inner monologue: Embodied reasoning through planning with language models. In *Proc. CoRL 2022*, PMLR 205, pp. 1769–1782.

Lin, C., & Mu, Y. (2024). InstructScene: Instruction-driven 3D indoor scene synthesis with semantic graph prior. In *Proc. ICLR*.

Noh, S., Seo, S., Park, G.-M., & Kang, H. (2026). Edit-As-Act: Goal-Regressive Planning for Open-Vocabulary 3D Indoor Scene Editing. In *Proc. CVPR*.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proc. UIST*.

Paschalidou, D., Kar, A., Shugrina, M., Kreis, K., Geiger, A., & Fidler, S. (2021). ATISS: Autoregressive transformers for indoor scene synthesis. In *Proc. NeurIPS*.

Pfaff, N., Cohn, T., Zakharov, S., Cory, R., & Tedrake, R. (2026). SceneSmith: Agentic Generation of Simulation-Ready Indoor Scenes. *arXiv preprint arXiv:2602.09153*.

Pun, H. I. D., Tam, H. I. I., Wang, A. T., Huo, X., Chang, A. X., & Savva, M. (2026). HSM: Hierarchical Scene Motifs for Multi-Scale Indoor Scene Generation. In *Proc. 3DV*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. In *Proc. ICML*, PMLR 139, pp. 8748–8763.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proc. EMNLP-IJCNLP*, pp. 3982–3992.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Proc. NeurIPS*.

Sumers, T. R., Yao, S., Narasimhan, K., & Griffiths, T. L. (2024). Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Sun, F.-Y., Liu, W., Gu, S., Lim, D., Bhat, G., Tombari, F., Li, M., Haber, N., & Wu, J. (2025). LayoutVLM: Differentiable Optimization of 3D Layout via Vision-Language Models. In *Proc. CVPR*.

Tam, H. I. I., Pun, H. I. D., Wang, A. T., Chang, A. X., & Savva, M. (2026). SceneEval: Evaluating Semantic Coherence in Text-Conditioned 3D Indoor Scene Synthesis. In *Proc. WACV*.

Tang, J., Nie, Y., Markhasin, L., Dai, A., Thies, J., & Nießner, M. (2024). DiffuScene: Denoising diffusion models for generative indoor scene synthesis. In *Proc. CVPR*.

Wang, C., Zhong, H., Chai, M., He, M., Chen, D., & Liao, J. (2024a). Chat2Layout: Interactive 3D Furniture Layout with a Multimodal LLM. *arXiv preprint arXiv:2407.21333*.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2024b). Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Wang, X., Yeshwanth, C., & Nießner, M. (2021). SceneFormer: Indoor scene generation with transformers. In *Proc. 3DV*.

Xia, H., Li, X., Li, Z., Ma, Q., Xu, J., Liu, M.-Y., Cui, Y., Lin, T.-Y., Ma, W.-C., Wang, S., Song, S., & Wei, F. (2026). SAGE: Scalable Agentic 3D Scene Generation for Embodied AI. In *Proc. CVPR*.

Xiaomi MiMo. MiMo-V2.5. Official model release, 2026. `https://huggingface.co/collections/XiaomiMiMo/mimo-v25`.

Yang, Y., Jia, B., Zhi, P., & Huang, S. (2024a). PhyScene: Physically Interactable 3D Scene Synthesis for Embodied AI. In *Proc. CVPR*.

Yang, Y., Lu, J., Zhao, Z., Luo, Z., Yu, J. J. Q., Sanchez, V., & Zheng, F. (2024b). LLplace: The 3D Indoor Scene Layout Generation and Editing via Large Language Model. *arXiv preprint arXiv:2406.03866*.

Yang, Y., Sun, F.-Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., Callison-Burch, C., Yatskar, M., Kembhavi, A., & Clark, C. (2024c). Holodeck: Language guided generation of 3D embodied AI environments. In *Proc. CVPR*.

Yang, Y., Jia, B., Zhang, S., & Huang, S. (2025). SceneWeaver: All-in-One 3D Scene Synthesis with an Extensible and Self-Reflective Agent. In *Proc. NeurIPS*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *Proc. ICLR*.

Zhao, Y., Sun, S., Zhang, M., Shi, Y., Yang, X., & Bian, J. (2026). SceneReVis: A self-reflective vision-grounded framework for 3D indoor scene synthesis via multi-turn RL. *arXiv preprint arXiv:2602.09432*.

Zhen, H., Li, X., Zhao, Y., Zhang, H., Liu, S., Mo, K., Gan, C., & Radhakrishnan, S. (2026). 3D-Layout-R1: Structured Reasoning for Language-Instructed Spatial Editing. *arXiv preprint arXiv:2603.22279*.

Zhu, X., Huang, X., Xie, Q., Deng, Z., Yu, J., Guan, Y., Liu, Z., Zhu, L., Zhao, Q., Liu, L., & Zeng, L. (2025). Imaginarium: Vision-Guided High-Quality 3D Scene Layout Generation. *ACM Transactions on Graphics*, 44(6).

APPENDIX

Appendix A describes the method and implementation. Appendix B specifies the benchmark and evaluation protocol. Appendix C reports additional experiments. Appendix D presents qualitative examples and failure analysis. Appendix E discusses limitations, future work, and broader impacts.

A METHOD DETAILS

A.1 OVERALL ALGORITHM

Algorithm 1 specifies requirement execution, consistency repair, verification, and memory updates within one authoring turn.

Algorithm 1. MUSE: Incremental scene authoring with three-layer memory

Input: Scene $\mathcal{S}_t$, input I_t, scene memory $\mathcal{G}_t$, skill library $\mathcal{K}$, max steps $T_{\max}$
Output: Updated scene $\mathcal{S}_{t+1}$, updated memory $(\varnothing, \mathcal{G}_{t+1}, \mathcal{K})$
$\mathcal{B}_t \leftarrow \textsc{Architect}(I_t,\ \mathcal{G}_t,\ \mathcal{K})$
$\mathcal{S}_0 \leftarrow \mathcal{S}_t$
$\mathcal{R}_0 \leftarrow \textsc{GroundSceneState}(\mathcal{S}_t,\ \mathcal{G}_t,\ \mathcal{B}_t)$
$\mathcal{W}_0 \leftarrow \textsc{InitWorkingMemory}(\mathcal{B}_t,\ \mathcal{R}_0)$
$\mathcal{G}_0 \leftarrow \mathcal{G}_t$
for $\tau = 1$ **to** $T_{\max}$ **do**
  $\text{calls}_\tau \leftarrow \textsc{Sculptor.Plan}(\mathcal{B}_t,\ \mathcal{R}_{\tau-1},\ \mathcal{W}_{\tau-1},\ \mathcal{G}_{\tau-1})$
  $\mathcal{S}_\tau \leftarrow \textsc{Sculptor.Execute}(\mathcal{S}_{\tau-1},\ \text{calls}_\tau,\ \mathcal{W}_{\tau-1})$
  $\mathcal{R}_\tau \leftarrow \textsc{LocalConsistencyPass}(\mathcal{S}_\tau,\ \mathcal{R}_{\tau-1},\ \text{calls}_\tau)$
  $\mathcal{F}_\tau \leftarrow \textsc{Inspector.Verify}(\mathcal{R}_\tau,\ \mathcal{B}_t,\ \mathcal{W}_{\tau-1})$
  $\mathcal{W}_\tau \leftarrow \textsc{UpdateWorking}(\mathcal{W}_{\tau-1},\ \text{calls}_\tau,\ \mathcal{F}_\tau)$
  $\mathcal{G}_\tau \leftarrow \textsc{UpdateScene}(\mathcal{G}_{\tau-1},\ \mathcal{R}_\tau)$
  if $(\textsc{AllKeySatisfied}(\mathcal{F}_\tau) \wedge \textsc{NoBlockingIssues}(\mathcal{F}_\tau)) \vee \textsc{StallDetected}(\mathcal{W}_\tau)$ **then**
    break
  end if
end for
$(\mathcal{S}_{t+1}, \mathcal{R}_{\text{final}}) \leftarrow \textsc{GlobalFeasibilityPass}(\mathcal{R}_\tau)$
$\mathcal{G}_{t+1} \leftarrow \textsc{UpdateScene}(\mathcal{G}_\tau,\ \mathcal{R}_{\text{final}})$
return $\mathcal{S}_{t+1},\ (\varnothing, \mathcal{G}_{t+1}, \mathcal{K})$

A.2 REQUIREMENT COMPILATION AND MEMORY

A.2.1 REQUIREMENT REPRESENTATION

The requirement program is defined as:

$$\mathcal{B}_t = \big(I_t,\ \textit{room_type},\ \{r_1, \ldots, r_n\},\ \mathbf{g},\ \mathbf{s}\big),$$

where $\mathbf{g}$ denotes scene-aware constraints grounded in the current room context, and $\mathbf{s}$ a style profile. Each requirement r_i is a structured record:

$$r_i = \big(\textit{type}_i,\ \textit{text}_i,\ \textit{priority}_i,\ \phi_i\big),$$

with $\textit{type}_i \in \{\texttt{hard}, \texttt{edit}, \texttt{preserve}, \texttt{forbid}\}$. Style, material, color, mood, and density preferences are stored in the top-level guidance/profile rather than as requirement types. The optional checkable predicate ϕ_i has the form:

$$\phi_i = \big(\textit{predicate},\ \textit{subject},\ \textit{object},\ \textit{qualifiers}\big),$$

where *predicate* $\in$ {`count`, `exists`, `left_of`, `right_of`, `near`, `on_top_of`, `against_wall`, ...} defines a deterministically checkable spatial or counting assertion, and *subject*/*object* are typed references carrying `semantic_type`, `canonical_type`, and `role` fields for robust matching.

A.2.2 SKILL MEMORY RETRIEVAL

Motif card schema. Each motif card in Skill Memory $\mathcal{K}$ stores a `motif_type`, trigger phrases, expected anchors, decomposition guidance, role bindings, optional qualifiers, and negative hints. An optional `exclusive_with` group prevents retrieval of competing interpretations.

Table 7: **Representative motif cards.** Fixed decomposition priors supplied to the Architect.

Motif	Triggers	Decomposition prior	Negative hint
Bed setup	bedside, flanking the bed	Split left/right nightstands and lamps into separate relation/support checks	Do not use one object for both sides
Dining set	chairs around table, dining area	Bind table first, then place chairs around distinct sides or perimeter slots	Do not collapse all chairs into count only
TV wall	TV wall, media console	Ground wall direction, place console on floor, attach TV to same wall	Do not treat TV as freestanding if wall-mounted
Reading corner	reading nook, cozy corner	Create zone, chair, side table, lamp, and clear access relation	Do not scatter objects across unrelated zones

Retrieval mechanism. Card retrieval proceeds as follows:

1. For each card in $\mathcal{K}$, compute a relevance score based on trigger-phrase overlap (exact match, substring, and keyword overlap) with the input instruction I_t and anchor-pattern overlap with the current scene objects.
2. Rank cards by score and select the top-k (k=2 by default).
3. Apply mutual-exclusion filtering: if two selected cards are in the same exclusive group, retain only the higher-scoring one.
4. Inject the selected cards' `preferred_decomposition` and `negative_hints` into the Architect's prompt as structured context.

In all experiments reported in this paper, Skill Memory is frozen at test time; no motif cards are added or updated during evaluation.

A.2.3 Requirement enrichment

For each placement predicate, the Architect extracts the referenced object and checks for an existing `hard` or `edit` requirement with an `exists` or `count` predicate targeting the same `canonical_type`. If absent, it injects a higher-priority `hard`/`exists` requirement and links it through `depends_on_requirement_ids`. For example, placing a lamp on a nightstand requires the nightstand to exist before placement.

A.2.4 Working Memory and protection state

Working Memory $\mathcal{W}_\tau$ at inner turn τ maintains the full execution state:

$$\mathcal{W}_\tau = \big(\mathit{focus}_\tau,\ \mathit{done}_\tau,\ \mathit{pending}_\tau,\ \Pi_\tau,\ \mathit{bindings}_\tau,\ \mathit{actions}_\tau,\ \mathit{issues}_\tau\big),$$

Table 8 defines the fields, their update sources, and their roles in execution. Here $\Pi_\tau = (T_\tau, P_\tau, P_\tau^T)$ collects the target and protection sets.

A.2.5 Scene Memory structure

Scene Memory $\mathcal{G}_t$ retains object identities, geometry, and spatial relations across instructions (Table 9). The graph is updated after each execution and verification step, so later instructions can refer to the current scene, for example, "the desk against the east wall." Zones and object groups support local scheduling and grouped edits.

The executor derives support surfaces, available wall regions, and free-floor placement regions from the current scene. These geometric domains guide placement; active requirement status and protection bindings belong to Working Memory.

A.2.6 Cross-turn memory inheritance

Each new instruction recompiles the requirement program and initializes Working Memory using the current Scene Memory. Protection applies within that authoring turn. Earlier requirements remain

Table 8: **Working Memory schema.** Execution state and protection sets are reinitialized for each instruction.

Field	Updated by	Stores	Used for
Focus	Scheduler	Current requirement ID and scope	Select local action target
Done / pending	Inspector	Requirement status partition	Stop condition and retry planning
Target set T	Architect / Scheduler	Objects intended to change	Separate target edits from non-target content
Protected set P	Inspector	Anchors and satisfied bindings	Block deletion or replacement regressions
Transform-protected P^T	Inspector	Objects that also cannot move/resize/rotate	Preserve layout and relation stability
Bindings	Inspector	Role-to-object links	Ground references and expand protection
Actions	Sculptor	Tool calls, blocked actions, degradations	Diagnose execution failures
Issues	Inspector	Failure modes and repair hints	Guide the next focused action

Table 9: **Scene Memory structure.** Scene geometry and relations provide context for later instructions.

Component	Stored state	Role in later turns
Room shell	Boundary, floor frame, canonical axes	Keep placement and OOB checks consistent
Wall structure	Wall IDs, directions, geometry, links to objects	Resolve wall references and support wall-relative placement
Support structure	Support objects, geometry, support relations	Identify supports and attached objects
Zones and groups	Spatial bounds, member objects, anchors	Schedule local actions and grouped edits
Object graph	Stable IDs, categories, poses, sizes, support/relation edges	Ground references and detect changed non-target objects

explicitly enforced only if included in the new program, so later instructions can revise or remove prior content.

Figure 5 illustrates five successive interactions, separately from the controlled three-stage stress test in Appendix C.4.1.

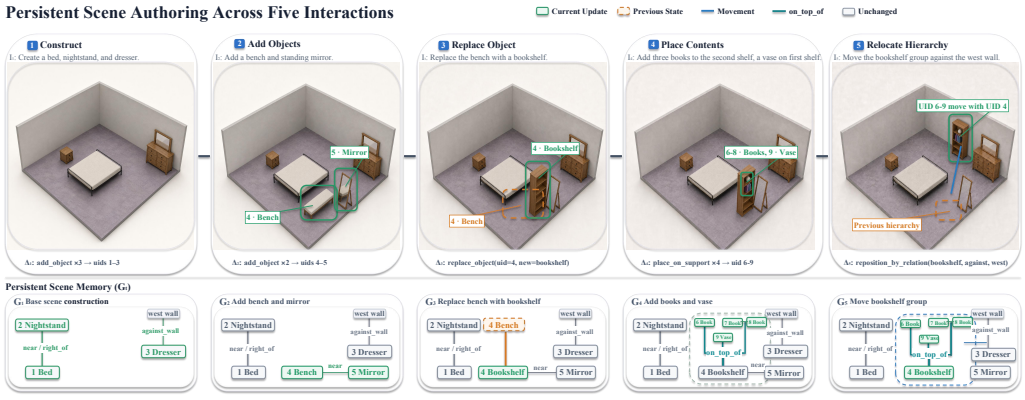


Figure 5: **Scene Memory across five interactions.** The top row shows successive edits; the bottom row shows inherited Scene Memory. Stable IDs track replacement at UID 4, while the bookshelf carries its books and vase when moved.

A.3 Execution, Verification, and Repair

A.3.1 Focus Scheduling and Stall Recovery

The focus scheduler selects the next requirement using dependency eligibility, spatial locality, and the scores below.

Scoring signals. For each candidate requirement $r_i \in \textit{pending}_\tau$, the scheduler combines the following main signals:

1. **Verifiability bonus** (+100): Requirements with rule-based verifier support (*verification_mode* = `rule_based`) receive a large bonus, as their satisfaction can be deterministically confirmed.
2. **Constraint kind priority**: Edit (+45) and entity (+35) requirements rank above placement (+10) and layout (-15) requirements, reflecting the dependency ordering principle: objects must exist before they can be positioned.
3. **Area overlap**: Shared area tokens with the previous focus add $30 + \min(15, 5n)$ points for n shared tokens.
4. **Graph locality**: Zone/object scopes receive local bonuses (+8/+4); transition bonuses favor shared zones, anchors, and zone types or patterns. Returning from zone to room scope incurs a -12 penalty.
5. **Priority weight**: Higher-priority requirements (lower *priority* integer) receive a modest bonus ($\max(0, 20 - \textit{priority} \times 5)$).
6. **Self-repetition penalty** (-50): If r_i was the previous focus and is still unsatisfied, it is penalized to encourage the scheduler to try alternative requirements.

The implementation also adds small candidate-order and turn-index offsets before ranking.

Selection with tie-breaking. Candidates are ranked by descending score, with requirement ID breaking score ties. A sole candidate or a leader ahead by at least 25 points is selected directly. Otherwise, the top three are sampled with weights $w_i = \max(\text{score}_i, 1)$, normalized by their sum. The deterministic seed hashes the turn index, previous focus ID, sorted shortlisted IDs, and blocked IDs.

Progress monitoring and escape. Local focus switching and global termination use separate criteria. A still-unsatisfied focus is temporarily blocked when both of its last two actions remain unsatisfied and use the same removal, movement, rotation, scaling, or replacement tool and either change overlapping object IDs or both make no object-level change. Blocked candidates are reconsidered if no unblocked candidate remains. Independently, the authoring loop stops after three consecutive rounds without requirement-level progress (Appendix A.4.1).

Dependency eligibility. Hard requirements are considered before edits. Entity requirements are eligible directly; placement requirements wait until their known dependency IDs leave the pending set; layout requirements wait for pending entity and placement requirements. Within the selected group, unblocked eligible candidates are preferred. If no eligible candidates remain, the scheduler falls back to pending candidates rather than deadlocking.

A.3.2 Tool hierarchy, lowering, and consistency passes

Declarative tools express spatial intent without requiring exact coordinates:

- `add_object_with_relation(ref_uid, relation, category, ...)`: Place a new object relative to an existing reference (e.g., `left_of`, `right_of`, `near`).
- `place_on_support(support_uid, category, ...)`: Place an object on a support surface (desk, nightstand, shelf).
- `place_under_support(support_uid, category, ...)`: Place an object beneath a support surface.
- `attach_to_wall(wall_direction, category, ...)`: Mount an object on a specified wall.
- `reposition_object_by_relation(uid, ref_uid, relation)`: Move an existing object to a new position defined by a spatial relation to a reference.

Imperative tools specify precise parameters:

- `add_object(category, pos, rot, size)`: Add an object at exact coordinates.
- `remove_object(uid)`: Remove an object by ID.
- `move_object(uid, new_pos)`: Move to exact position.

- `rotate_object(uid, new_rot)`: Set exact rotation.
- `scale_object(uid, new_size)`: Set exact dimensions.
- `replace_object(uid, new_category, ...)`: Swap category while preserving position.
- `terminate`: Signal loop termination.

Table 10: **Tool execution and fallbacks.** Geometric operations and checks associated with each action.

Action	Geometric operation	Fallback	Affected checks
Relation placement	Add/move object at relation-derived floor position	Nearby free-floor placement around the reference	Relation, collision, OOB
Support placement	Add object at support-surface height and free support footprint	Nearby floor placement with logged degradation	Surface/support, collision
Wall attachment	Add object flush to wall segment and align rotation	Nearest available wall segment	Wall, orientation, OOB
Object replacement	Remove target and add replacement at inherited pose/scale	Keep target and report blocked replacement if protected	Object, preserve, forbid
Local repair	Move only edited neighborhood by minimum feasible offset	Surface issue to Inspector if protected content is required	Physical validity, previously satisfied checks

Geometric grounding. Support placement uses the unoccupied footprint within the support's bounding box and sets y to the surface height plus half the object height. Wall attachment computes the wall-normal offset and aligns the object's rotation with the wall. Relation placement uses the reference object's bounding box, facing direction, and a configurable offset.

Tool degradation. If grounding fails, the executor uses the fallback in Table 10. A `ToolDegradation` record stores the original tool, fallback tool, and reason in Working Memory. The original requirement remains unsatisfied until verified.

Local consistency pass. After each step, the executor checks added or moved objects, their supports, and immediate collision partners. It corrects support heights, wall offsets, and small collisions using the minimum feasible adjustment. Objects in P_τ and P_τ^T are never moved or deleted by this pass; repairs that would affect them are reported to the Inspector.

Global feasibility pass. After loop termination, one full-scene pass checks residual collisions and out-of-boundary placements. It prioritizes preserving protected content, then target-related objects introduced by the current instruction, then other objects. Conflicts between unprotected objects favor the smaller displacement while preserving satisfied hard requirements where possible. Unresolved violations remain in the final OOB / Col. counts.

A.3.3 Inspector feedback and re-verification

Re-verification rules. After execution and local repair, the Inspector checks the focused requirement and requirements involving edited or repaired objects, including those that share objects with the edited neighborhood. Affected requirements, including previously satisfied ones, are checked before protection bindings are updated.

Feedback and state updates. The Inspector returns a structured feedback record

$$\mathcal{F}_\tau = \{f_i\}_{i=1}^{n}, \qquad f_i = \big(\textit{status}_i,\ \textit{source}_i,\ \textit{matched_ids}_i,\ \textit{failure_mode}_i,\ \textit{repair_hint}_i\big),$$

where $\textit{status}_i \in \{\texttt{satisfied}, \texttt{unsatisfied}, \texttt{unknown}\}$ and $\textit{source}_i$ records whether the decision came from a rule-based predicate, model-assisted judgment, or both. When a rule-based predicate is available, its result takes precedence for the corresponding hard spatial or counting

assertion; model-assisted feedback is used for residual style, perceptual, or open-ended semantic aspects. The reconciled record updates Working Memory by moving satisfied requirements to $done_\tau$, attaching matched object IDs to protection bindings, and recording failure modes or repair hints for the next planning turn.

Figure 6 shows this feedback loop in an execution trace.



Figure 6: **Inspector-guided repair.** The first action creates the rug but fails its under-support requirement. Feedback directs a local repair of the rug; re-verification confirms target satisfaction and preservation.

A.4 Implementation details

A.4.1 Runtime configuration

Unless a backend replacement is explicitly specified, MUSE runs use MiMo-V2.5 (Xiaomi MiMo, 2026) with temperature 0, turn budgets of 30/8 for construction/editing, and a 300-second request timeout. Architect and Sculptor planning requests are not retried; Inspector requests retry at most twice with a five-second backoff only for timeout, connection, 408/429, or 5xx failures. Core `max_completion_tokens` uses the provider default, while empty-scene initialization uses 2,000; an unparsable response receives at most one format repair. A run stops when all goals pass, after three stalled rounds without requirement-level progress, or at its turn cap. The nominal budget is one run per case; output-less runs follow the recovery rule in Appendix B.3. The exported scene is scored, and wall-clock time is measured end to end.

A.4.2 Asset library and retrieval details

The Architect and Sculptor provide object descriptions and size hints for asset retrieval. Spatial relations are excluded from retrieval text and enforced by placement tools and verification.

Unified asset registry. The registry combines 3D-FUTURE (Fu et al., 2021b) with optional long-tail or dynamic sources into 66,655 records. Records contain identity/source fields (`asset_key`, `asset_id`, `source`), render path, bounding-box size (`bbox_xyz`), caption, tags, optional category, and text/image embedding indices. Source labels allow restricted retrieval, such as 3D-FUTURE-only selection for baseline adapters.

Query formation and scoring. A query combines the planner's object description (e.g., "compact desk lamp") with an optional target size from the requirement or edit operator.

For a query q, candidate asset a, and target size $\mathbf{s}^*$, the retriever ranks assets using a weighted score

$$\text{score}(a \mid q, \mathbf{s}^*) = \lambda_{\text{sbert}} \cos(\mathbf{e}_q^{\text{text}}, \mathbf{e}_a^{\text{text}}) + \lambda_{\text{clip}} \max_v \cos(\mathbf{e}_q^{\text{clip}}, \mathbf{e}_{a,v}^{\text{clip}}) - \lambda_{\text{size}} \cdot \frac{1}{3} \sum_{d \in \{x,y,z\}} \frac{|s_{a,d} - s_d^*|}{\max(s_d^*, \epsilon)}.$$

The first term matches the query against normalized SBERT (Reimers and Gurevych, 2019) text embeddings, the second term adds multiview CLIP (Radford et al., 2021) similarity when image features are enabled, and the last term penalizes size mismatch with the requested bounding-box hint. Source filters are applied before top-k extraction, and the resulting candidate list is sorted by the final score.

Selection policy and traceability. MUSE retrieves the top 20 candidates and selects the highest-scoring asset in deterministic evaluation. Stochastic variants may sample using normalized final scores. Each scene object records the selected `asset_id`, `asset_source`, bounding box, render path, and a `selection_breakdown` with the SBERT score, CLIP score, size error, and combined score.

A.4.3 Agent prompts and runtime context

Full prompts, enums, and JSON schemas are included in the supplementary code. Table 11 summarizes runtime inputs and outputs.

Architect parsing rules. The Architect outputs one JSON object, preserves `original_request` exactly, and assigns sequential IDs to concise English requirements without inventing objects or constraints. Requirement types and style fields follow Appendix A.2.1. It normalizes faithful synonyms (e.g., "next to" to `near`) and support relations; clauses that cannot be checked retain their text with `verifier_constraint = null`. Retrieved cards guide decomposition and role binding without overriding explicit requests. An optional preference profile supplies only a soft bias for ambiguous requests.

Table 11: **Agent runtime messages.** Inputs supplement each agent's fixed system prompt.

Agent	Output	Runtime context
Architect	DesignBrief JSON	Request, scene context, few-shot examples, retrieved motif cards, optional preference profile
Sculptor	Tool-calls JSON block	Focus, pending same-scope requirements, protection sets, recent actions, issues, scene summaries
Inspector	Feedback JSON	Requirements, execution result, current scene, rendered view when available

B Benchmark and Evaluation

B.1 AuthorBench

Composition and sampling. The 145 construction cases cover bedroom, living room, dining room, study, and office (25 each), and kitchen (20). The fixed editing split samples 10 cases per (complexity tier, room type) cell deterministically from the 1584-case pool, yielding 240 cases. Table 12 reports tier, source, and target coverage. Construction contains 1043 goal checks; the editing pool contains 4140 goal and 4752 preserve checks, of which 648 and 720, respectively, are in the test split. Editing cases are additionally balanced by edit family; split-balancing metadata are withheld at test time.

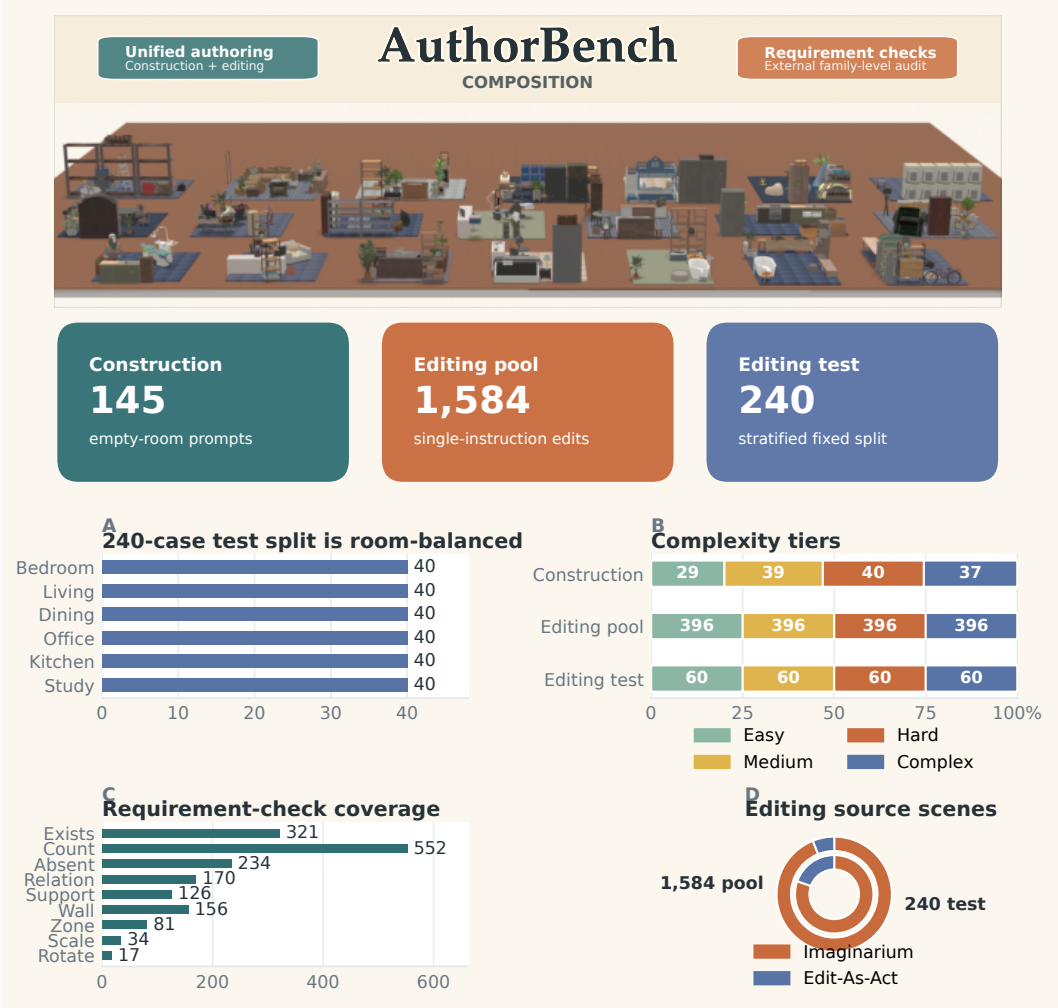


Figure 7: **AuthorBench composition.** Room, complexity, source-scene, and check-type distributions for construction and editing.

Table 12: **Editing pool and test-split coverage.** (a) Edit tiers and source families; (b) source-scene complexity and number of targets.

(a) Edit design		
Statistic	**Pool**	**Test**
E1 atomic	396	60
E2 grounded	396	60
E3 structured	396	60
E4 compound	396	60
Imaginarium	1488	193
Edit-As-Act	96	47

(b) Scene and target burden		
Statistic	**Pool**	**Test**
S1 simple	300	35
S2 medium	1008	135
S3 dense	192	41
S4 extreme	84	29
Q0 single target	1068	161
Q1 two–three targets	374	57
Q2 four-plus targets	142	22

Dataset creation. Each case follows three stages:

- *(i) Skeleton drafting.* Room-specific templates specify object inventory, allowed/forbidden categories, supports, wall references, and functional zones, with tier-balanced sampling within each room type.
- *(ii) Constraint compilation.* Each skeleton becomes a natural-language prompt and external checks typed as object, relation, surface, zone, preserve, or forbid, with machine-checkable predicates where feasible (Appendix B.2.2).
- *(iii) Human review.* Two annotators independently check consistency among the prompt, source scene (for editing), and predicates, as well as realizability. Disputed cases are revised or discarded.

Check provenance and freezing. AuthorBench and MUSE were co-developed. Construction checks derive from reviewed specifications, and editing checks from frozen source-scene metadata and target identities. Neither derives from MUSE outputs, memories, trajectories, or scores. AuthorBench was finalized before MUSE and Skill Memory were frozen.

Tier definitions. Construction tiers progress from inventory, count, and negative constraints (Easy), to relations and wall placement (Medium), support and functional zones (Hard), and interacting constraint families in compound prompts (Complex). Editing tiers E1 atomic, E2 grounded, E3 structured, and E4 compound correspond to Easy, Medium, Hard, and Complex, respectively.

Editing source scenes. Scenes from Edit-As-Act (Noh et al., 2026) and Imaginarium (Zhu et al., 2025) are normalized to stable object IDs, canonical categories, simple geometry, supports, wall anchors, and functional zones. These fields support references such as "the desk against the east wall" and preservation checks against source-object IDs. Every method receives the same normalized scene for each case; external checks are withheld at test time.

B.2 Metrics and external verification

B.2.1 Metric definitions

Goal fulfillment. For a case with annotated goal checks $\mathcal{G}$, GF is the fraction of satisfied checks:

$$\mathrm{GF} = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} \mathbf{1}[g \text{ satisfied}].$$

Both tasks annotate four families: object (existence, absence, count, scale, rotation), relation, surface (support and wall placement), and zone (functional zones and clear areas). Construction Macro averages all four family GFs; Macro (no Surf.) averages object/relation/zone, and Core averages object/relation. The reported editing Macro averages object/relation/surface, excluding Zone. All averages are unweighted. All-Goal is the fraction of cases satisfying every annotated goal check; All-Goal (no Surf.) recomputes this fraction after excluding surface checks.

Editing locality. For editing cases, preservation rate (PR) measures the fraction of protected checks that remain valid:

$$\mathrm{PR} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \mathbf{1}[p \text{ preserved}].$$

Unintended change rate (UCR) measures accidental modification of non-target initial objects:

$$\mathrm{UCR} = \frac{\#\{\text{changed non-target objects}\}}{\#\{\text{non-target objects}\}}.$$

Physical validity. OOB (out-of-boundary) and Col. (collisions) are the average number of out-of-boundary objects and colliding object pairs, computed from the final scene geometry. For construction, these measure the physical validity of the generated scene. For editing, these measure new physical issues introduced by the edit operation, assuming the source scene is physically valid.

Complexity retention. Construction and editing cases are organized into four tiers: Easy, Medium, Hard, and Complex. We report GF for each tier, then compute retention:

$$\mathrm{Ret.} = \frac{\mathrm{GF}_{Complex}}{\mathrm{GF}_{Easy}}.$$

Retention is a descriptive ratio across distinct difficulty subsets. Values above 100% mean higher GF on the evaluated Complex subset than on the Easy subset; they do not establish that increasing task complexity improves performance.

B.2.2 External verifier predicate catalog

Table 13 lists the external checks and their scene inputs. Preservation checks additionally reference the source scene.

Table 13: **AuthorBench predicate inventory.** External checks evaluated on exported scenes.

Family	Predicates / checks	Inputs	Typical failure modes
Object	Existence, count, negative existence, category match	Object IDs, canonical category, count target	Missing object, extra forbidden object, wrong category
Relation	Left/right, front/behind, near/far, parallel	Subject/object boxes, room frame, facing direction	Wrong side, too far/near, orientation mismatch
Surface	On-support, under-support, wall anchoring, against-wall	Support surface, wall domain, object footprint, height	Unsupported object, wall offset, wrong support
Zone	Inside-zone, zone containment, free-floor area, clear region	Zone polygon, object footprint, free-space grid	Object outside zone, cluttered zone, insufficient clear area
Preserve	Existence, pose, relation, support preservation	Source-scene IDs, original poses and relations	Deleted non-target, moved anchor, broken support/relation
Forbid	Category, region, relation forbiddance	Negative constraint, scene objects, zones/relations	Forbidden object, forbidden placement, forbidden relation
Physical	OOB, collision pairs, support-height consistency	Room boundary, bounding boxes, support heights	Out-of-room object, interpenetration, floating/sunken object

B.2.3 Verifier coverage and fallback judgment

AuthorBench applies the same external evaluator to every method's exported scene, independently of MUSE's internal decomposition, feedback, and stopping decisions.

Independent semantic judging. For residual semantic or perceptual constraints that cannot be checked geometrically, an independent judge receives the original requirement, exported scene representation, and rendered views under the same prompt and protocol for every method. Its judgments never override deterministic predicate outcomes.

B.3 BASELINE ADAPTERS AND FAIRNESS PROTOCOL

Fairness protocol. Main AuthorBench comparisons use MiMo-V2.5 for all LLM/VLM methods. Baselines retain their native prompts, stopping rules, and control mechanisms, without MUSE's verification, repair, or protection modules. Methods share benchmark inputs, room shells or source scenes, and the common asset pool where conversion is required. Native outputs are converted once to unified scene JSON for common rendering and scoring (Table 14). Failed or partial runs are scored from their last evaluable scene without manual repair. Only runs with neither an evaluable final scene nor an intermediate snapshot are rerun under the same configuration; exported scenes that fail requirements are scored as produced.

Table 14: **Baseline interfaces, adapters, and limitations.** All outputs use the common AuthorBench scene format and scoring protocol. Weak families are not explicitly represented or verified in the native interface.

Baseline	Native interface and adapter	Limitations / output handling
LayoutGPT / LayoutGPT-E	One-shot text-to-layout; prompt plus room/source-scene serialization; convert final layout	No native protection; weak zones; prompt-only support/wall constraints
LayoutVLM	Native optimization once per case; room shell plus candidate asset metadata; export layout	No editing or native preservation; weak support/zone constraints
Holodeck	Single-room pipeline without per-case retuning; room shell plus prompt; export scene	No editing or native preservation; weak dense support/wall constraints
ReSpace / ReSpace-E	Native SSR asset sampling and prompt handling; convert final SSR scene once	Weak support/wall and zone constraints; preservation-biased editing
Edit-As-Act	Goal-regressive action loop; source scene plus instruction through prepared action interface	No construction; weak surface/zone-heavy edits; score last emitted scene without case-specific retries
SceneReVis	Released batch pipeline and multi-turn revision; export scene snapshots	No explicit protection sets; weak support/wall verification; use final scene or last snapshot for partial runs

The *final-scene export rate* is the fraction of cases emitting an evaluable `final_scene.json`. It measures output availability, separately from All-Goal success.

Table 15: **Final-scene export availability.** Entries give evaluable final scenes / cases (rate). Export availability is distinct from task success; "—" denotes an unevaluated task.

Method	Construction	Editing
LayoutGPT / LayoutGPT-E	145/145 (100.0%)	240/240 (100.0%)
Holodeck	145/145 (100.0%)	—
LayoutVLM	141/145 (97.2%)	—
ReSpace / ReSpace-E	145/145 (100.0%)	240/240 (100.0%)
SceneReVis	99/145 (68.3%)	229/240 (95.4%)
Edit-As-Act	—	212/240 (88.3%)
MUSE	145/145 (100.0%)	240/240 (100.0%)

B.4 HUMAN EVALUATION PROTOCOL AND INTERFACE

Participants and study setting. We recruited 50 independent adults through an open campus-wide call: 16 participants without relevant 3D/CV/graphics training, 16 with architecture or design backgrounds, and 18 with prior 3D-vision or computer-graphics training. Participation was voluntary and uncompensated.

Evaluation procedure. Each participant assessed four anonymized editing results, one from each of MUSE, Edit-As-Act, ReSpace-E, and SceneReVis. For each method, a result was sampled uniformly from a pool of 10 editing cases. Participants viewed the source scene, instruction, and edited result, then rated instruction satisfaction, preservation, and physical plausibility on a 1–5 scale (1 = strongly

disagree, 2 = disagree, 3 = neutral, 4 = agree, 5 = strongly agree). The rating items were shuffled and method names were hidden, yielding 200 result-level evaluations (50 per method). Each participant additionally completed 10 anonymized four-way preferences in which all candidates corresponded to the same source scene and editing instruction, yielding 500 preference judgments.

Consent and ethics. Reading the study information and proceeding signaled consent. The study consisted of voluntary, anonymous visual judgments of synthetic 3D scenes; it involved no physical intervention, deception, sensitive content or personal data, or procedure expected to adversely affect participants' psychological well-being. These characteristics make the study a low-risk, non-invasive visual-evaluation task.

(a) Result-level rating form. **(b) Four-way forced-choice form.**

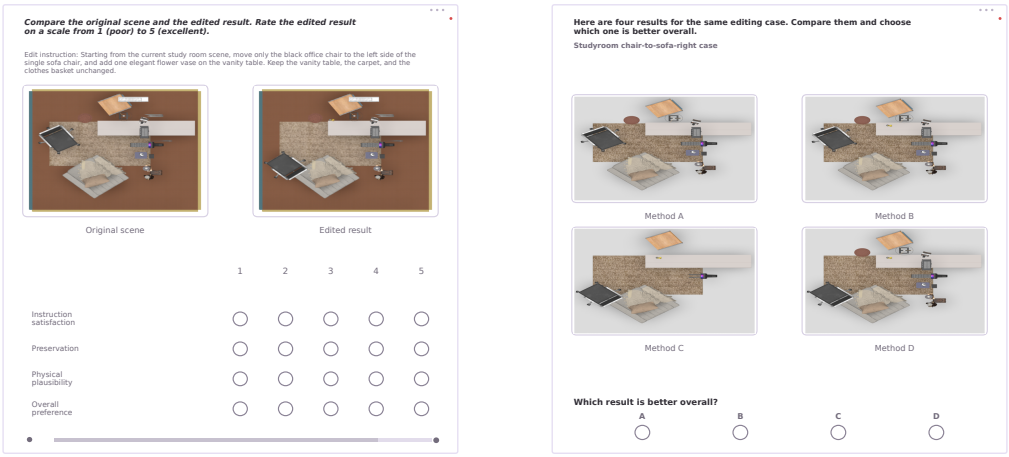


Figure 8: **Human evaluation interfaces.** (a) Ratings of instruction satisfaction, preservation, and physical plausibility; (b) overall preference among four outputs for the same case.

C Additional Experiments

C.1 Full results and breakdown

Tables 16 and 17 give the complete family and difficulty breakdowns, including editing completion and locality. Metric definitions are in Appendix B.2.1; room-wise results follow below.

Table 16: **Full construction metrics.** Family scores, diagnostic aggregates, and difficulty breakdowns; definitions are in Appendix B.2.1. Bold marks column-wise best values, including ties.

Method	Goal Fulfillment ↑							Strict Completion ↑		Complexity Robustness ↑					Physical Validity ↓	
	Obj.	Rel.	Surf.	Zone	Macro	Macro (no Surf.)	Core	All-Goal	All-Goal (no Surf.)	Easy	Med.	Hard	Complex	Ret.	OOB	Col.
LayoutGPT	98.4	83.8	35.0	79.0	74.1	87.1	91.1	37.9	80.0	99.3	86.0	83.4	82.2	82.7	1.4	2.3
Holodeck	83.3	57.6	21.5	30.9	48.3	57.3	70.5	14.5	24.8	84.7	73.7	67.9	54.1	63.9	**0.0**	**0.0**
LayoutVLM	98.0	69.7	22.1	59.3	62.3	75.7	83.9	26.9	62.8	**100.0**	77.4	72.7	81.7	81.7	**0.0**	**0.0**
ReSpace	71.9	36.4	8.6	32.1	37.3	46.8	54.2	11.0	17.2	81.7	59.6	50.8	47.0	57.5	0.4	0.5
SceneReVis	46.9	17.2	9.2	8.6	20.5	24.2	32.1	2.1	2.1	47.8	40.6	32.0	31.4	65.6	0.3	0.9
MUSE	**99.6**	**96.0**	**93.3**	**85.2**	**93.5**	**93.6**	**97.8**	**80.7**	**87.6**	99.3	**97.9**	**97.8**	**95.9**	**96.6**	**0.0**	0.1

C.1.1 Per-room-type performance breakdown

C.2 External benchmark validation

E2A-Bench protocol. We run MUSE once on each of the 63 tasks using the frozen AuthorBench editing configuration (Appendix A.4.1), adapting benchmark instructions to its interface. Scoring uses the native Instruction Fulfillment (IF), Scene Coherence (SC), and Physical Plausibility (PP)

Table 17: **Full editing metrics.** Macro averages object, relation, and surface fulfillment. Bold marks column-wise best values, including ties.

Method	Goal Fulfillment ↑				All-Goal↑	Complexity Robustness ↑					Edit Locality		Physical Validity ↓	
	Obj.	Rel.	Surf.	Macro		Easy	Med.	Hard	Complex	Ret.	PR↑	UCR↓	OOB	Col.
LayoutGPT-E	**86.0**	21.1	19.3	42.2	36.2	**86.7**	59.2	61.0	67.2	77.5	25.1	95.7	1.8	5.2
Edit-As-Act	30.6	14.1	5.0	16.6	5.4	21.7	23.3	27.2	22.2	102.3	84.9	17.8	0.2	0.5
SceneReVis	33.6	28.2	4.2	22.0	10.4	31.7	33.3	26.6	25.4	80.1	92.6	5.9	0.8	2.1
ReSpace-E	46.3	19.7	5.9	24.0	11.2	40.8	27.5	36.2	38.6	94.6	**100.0**	1.9	1.2	3.5
MUSE	79.0	**74.7**	**89.9**	**81.2**	**76.7**	80.0	**84.2**	**73.2**	**83.9**	**104.8**	99.9	**0.6**	**0.1**	**0.3**

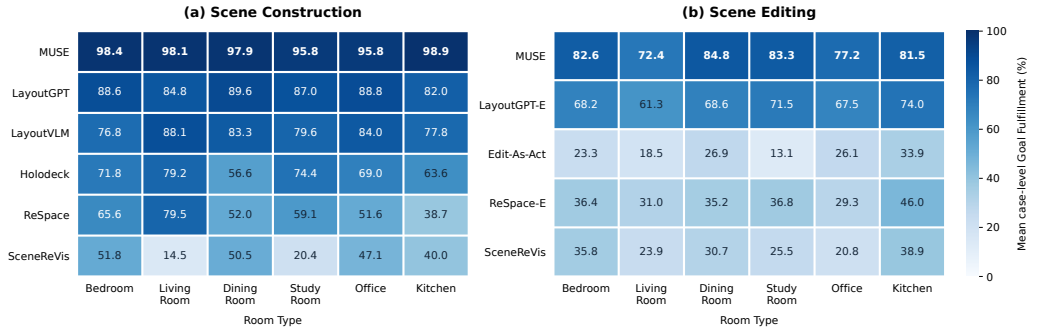


Figure 9: **Goal Fulfillment by room type.** Cells average case-level GF within each task and room type, distinct from family-level Macro. Construction uses 25 cases per room except kitchen (20); editing uses 40. Darker cells indicate higher scores. MUSE leads in all six room types for both tasks.

evaluator. LayoutGPT-E and Edit-As-Act results come from the E2A-Bench report. All methods use the same unweighted average over nine room categories (Table 18).

Table 18: **E2A-Bench room-wise results (%).** The final row averages the nine room categories equally.

Room	LayoutGPT-E			Edit-As-Act			MUSE		
	IF	SC	PP	IF	SC	PP	IF	SC	PP
Bathroom	50.1	62.5	78.3	58.7	88.9	89.1	65.5	85.2	92.4
Bedroom	35.7	30.2	53.8	45.7	73.1	91.9	54.3	81.4	92.6
Computer Room	53.8	32.1	84.7	73.6	88.0	94.1	72.8	90.2	95.7
Dining Room	50.5	31.4	66.2	89.7	95.3	92.7	90.2	92.5	94.8
Game Room	52.1	48.9	83.5	58.7	79.9	91.1	55.5	76.0	92.8
Kids Room	28.3	30.1	88.6	91.1	89.0	96.3	75.0	90.4	95.0
Kitchen	59.4	58.2	88.3	55.0	92.3	93.7	69.6	89.1	94.4
Living Room	38.6	40.7	83.9	72.9	90.1	93.6	73.7	90.0	94.2
Office	32.1	45.4	77.1	76.4	82.9	81.9	77.3	91.2	93.8
Room macro	44.5	42.2	78.3	69.1	86.6	91.6	**70.4**	**87.3**	**94.0**

C.2.1 Additional external protocols

LayoutGPT-native construction. Table 19 uses LayoutGPT's official 3D-FRONT split, retrieval setup, and native KL and OOB metrics. MUSE achieves lower KL and zero OOB errors in both evaluated room types.

Adapted SceneEval-94 construction evaluation. We use the 94 single-room cases of SceneEval-100 (Tam et al., 2026), excluding six multi-room cases. For every method, official Object Count (CNT), Object–Object Relationship (OOR), and Object–Architecture Relationship (OAR) annotations are translated into natural-language requirements. Table 20 also reports Support (SUP), scene-level

Table 19: **LayoutGPT-native construction evaluation.** KL is lower-is-better; OOB reports scene/object percentages under LayoutGPT's official protocol.

Method	Bed. KL↓	Bed. OOB↓	Liv. KL↓	Liv. OOB↓
LayoutGPT-GPT4	0.142	38.77 / 11.15	0.161	64.15 / 20.07
MUSE	**0.127**	**0 / 0**	**0.129**	**0 / 0**

Collision (COLsc), and Accessibility (ACC). Holodeck achieves higher accessibility, which MUSE's current requirement and verification loop does not explicitly optimize.

Table 20: **Adapted SceneEval-94 construction evaluation.** Higher is better except scene-level Collision (COLsc).

Method	CNT↑	OOR↑	OAR↑	SUP↑	COLsc↓	ACC↑
LayoutGPT	20.69	2.29	6.05	28.63	30.00	46.69
Holodeck	42.72	18.91	47.98	61.31	73.00	**90.04**
MUSE	**55.41**	**26.64**	**49.36**	**83.83**	**25.29**	79.71

C.3 Ablations and generalization

C.3.1 Cross-backend generalization

We replace MiMo-V2.5 with five VLM backends through the same OpenAI-compatible multimodal interface, keeping prompts, tools, memory, rendering, scoring, and runtime settings fixed (Appendix A.4.1). Table 21 uses the same 145 construction and 240 editing cases as the main results.

Table 21: **Cross-backend generalization on AuthorBench (%).** MiMo-V2.5 is the default. Bold marks the best displayed value per column, including ties.

Backend	Construction (145 cases)		Editing (240 cases)			
	Macro↑	All-Goal↑	Macro↑	All-Goal↑	PR↑	UCR↓
GPT-4o	92.2	80.7	69.0	49.6	**99.9**	**0.6**
MiMo-V2.5 (default)	**93.5**	80.7	**81.2**	**76.7**	**99.9**	**0.6**
Qwen3-VL-72B	92.2	78.6	69.5	50.8	**99.9**	0.7
Qwen3-VL-235B-FP8	92.7	**83.5**	78.1	54.6	99.7	0.7
InternVL3-78B	89.6	76.6	71.2	49.2	**99.9**	**0.6**
Qwen2.5-VL-7B	86.9	69.0	62.1	42.5	99.6	0.7

C.3.2 Memory and asset mechanism checks

History-only planning controls. The two controls retain the same Architect output, model, assets, action space, Inspector, geometric repair, evaluator, and turn budgets as the full system. The planner receives the current scene, full requirement list, and complete chronological action/feedback history, but not the structured focus, completed/pending ledger, role bindings, Scene Memory summary, or protection lists. Internal state remains available for verification, stopping, and, in the protected variant, execution-time guards. The no-protection variant additionally disables these guards.

Skill Memory retrieval. Table 22 separates cases by whether top-2 retrieval activates a motif card (Appendix A.2.2). Gains over the ablation are concentrated on triggered cases, consistent with the use of motif guidance.

Asset-library scaling. On the same 145 construction cases, the unified registry improves coverage and completion over 3D-FUTURE alone, at an additional 0.10 seconds of retrieval time per object (Table 23).

Table 22: **Skill Memory mechanism check.** Macro goal fulfillment (%) on retrieval-triggered and non-triggered subsets. Δ is full MUSE minus the ablation without Skill Memory, in percentage points.

Task	Subset	Cases	Full	w/o Skill	Δ
Construction Macro	Triggered	135	93.0	87.9	+5.1
Construction Macro	Non-triggered	10	100.0	99.3	+0.7
Editing Macro	Triggered	173	87.2	73.6	+13.6
Editing Macro	Non-triggered	67	65.8	65.7	+0.1

Table 23: **Asset-library scaling on the 145 construction cases.** Retrieval time is reported per object in seconds.

Library	Assets	Coverage↑	Unresolved↓	Macro-GF↑	All-Goal↑	Retrieval / object↓
3D-FUTURE only	16,563	94.4	5.6	89.4	73.8	0.07 s
Unified	66,655	**99.1**	**1.4**	**93.5**	**80.7**	0.17 s

C.4 Continuous editing and navigation curriculum

C.4.1 Continuous editing stress test

We evaluate 50 three-stage chains, each comprising initial construction followed by two edits to the preceding output. The resulting 100 editing stages test scene-state inheritance and preservation across instructions.

Table 24: **Continuous editing stress test.** Strict completion is derived from run status, distinct from AuthorBench's external All-Goal metric.

Edit family	**Tasks**	**Completed**	**Completion (%)**
Add	58	57	98.3
Delete	8	7	87.5
Replace	8	6	75.0
Scale	8	8	100.0
Move	9	7	77.8
Rotate	9	4	44.4
Overall	100	89	89.0

MUSE completes 139 of 150 stages (92.7%); both edit stages complete in 39 of 50 chains. Table 24 reports edit-level completion. Rotation is the weakest operation: five of nine cases retain facing-relation errors. Other failures involve stalled insertion or removal, incomplete replacement, or unresolved movement relations.

C.4.2 Downstream navigation curriculum details

Each case has three stages in the same room. The incremental variant edits each preceding stage with MUSE; regeneration creates each stage independently from its prompt. Both use the same scene format and grid-based navigation proxy. Table 25 defines the protocol, and Table 26 gives stage-wise results.

Table 25: **Navigation curriculum evaluation protocol.** Metrics are computed from exported layouts, not from a trained embodied agent policy.

Parameter / metric	Definition
Cases and stages	100 curriculum cases, each with 3 stages, for 300 scenes per method
Grid resolution	0.1 m occupancy grid over the room floor
Obstacle buffer	0.05 m expansion around object footprints
Walkable ratio	Fraction of floor cells not occupied by buffered obstacles
Reachable-object ratio	Fraction of target objects reachable from free space by grid search
Mean access path length	Mean shortest path length from accessible free cells to target access rings
Difficulty monotonicity	Case passes if walkable area does not increase and path length does not decrease beyond tolerance across stages
Free-space preservation IoU	Intersection-over-union of free-space masks between successive stages
Physical validity	Collision and out-of-boundary rates from exported scene geometry

Table 26: **Full stage-wise navigation curriculum metrics.** Both variants use the same grid-based proxy; path length is in meters.

Method	Stage	Walkable	Reachable	Path m	Col.	OOB
Incremental MUSE	1	0.906	1.000	1.235	0.000	0.000
Incremental MUSE	2	0.886	1.000	1.554	0.000	0.000
Incremental MUSE	3	0.868	0.992	1.736	0.000	0.000
Stage-wise regeneration	1	0.912	1.000	1.233	0.000	0.000
Stage-wise regeneration	2	0.894	1.000	1.794	0.000	0.000
Stage-wise regeneration	3	0.878	0.999	2.183	0.000	0.000

C.5 Efficiency

All results are inference-only, with no training or fine-tuning on AuthorBench. Table 27 accounts for the final construction and editing runs; development, curation, debugging, and pilot evaluations incurred additional compute.

Table 27: **Frozen-run efficiency audit.** Full-run accounting over all 145 construction cases and the fixed 240-case editing split. Text API costs are estimated from the listed token usage at MiMo-V2.5 OpenRouter rates of $0.14 per million uncached input tokens and $0.28 per million output tokens (September 22, 2026; `https://openrouter.ai/xiaomi/mimo-v2.5`), excluding image-token charges.

Audit metric	Construction (145)	Editing (240)
Total LLM requests	1,685 / 11.62 per case	1,555 / 6.48 per case
Sculptor planning requests	657 / 4.53 per case	725 / 3.02 per case
Proposed tool calls	805 / 5.55 per case	984 / 4.10 per case
Executed scene operations	801 / 5.52 per case	972 / 4.05 per case
Input / output tokens	19.38M / 0.67M	18.24M / 0.82M
Estimated text API cost	$2.90 / $0.020 per case	$2.78 / $0.012 per case
Mean / median closed-loop turns	4.97 / 4	2.65 / 3
Cases reaching the turn cap	4 (2.8%)	12 (5.0%)
Mean / median wall time	108.24 / 84.91 s	479.53 / 296.23 s

D Qualitative Results and Failure Analysis

D.1 Additional construction and editing examples

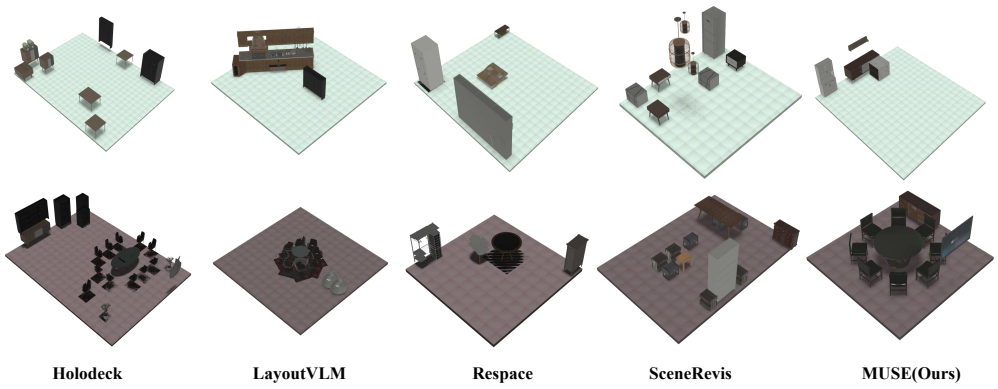


Figure 10: **Additional construction comparisons.** AuthorBench examples with multi-object spatial and support constraints.

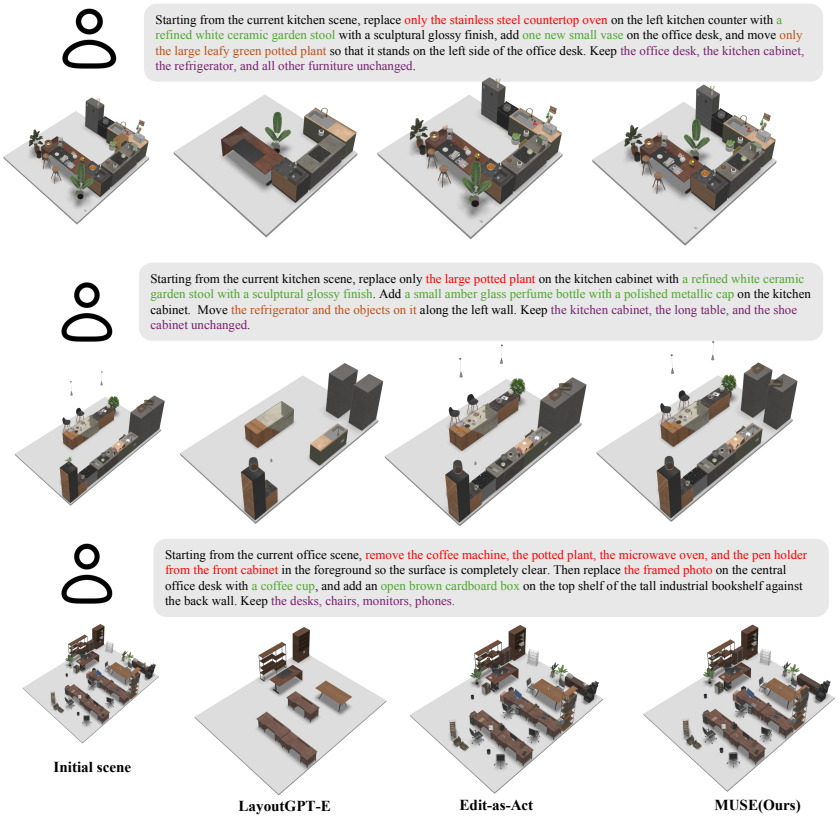


Figure 11: **Additional editing comparisons.** Kitchen and office cases combine replacement, insertion, movement, or removal with preservation of non-target content.

D.2 Failure mode analysis

Precise transforms can fail despite correct target grounding and non-target preservation. Table 28 reports GF by operation; scale and rotation have the lowest scores.

Table 28: **Fixed-split editing operation breakdown.** Operation memberships overlap for compound cases; GF is goal fulfillment on the fixed 240-case split.

Operation	Case memberships	GF↑
Add	200	85.6
Move	46	70.1
Remove	39	71.3
Replace	28	72.6
Rotate	17	63.2
Scale	34	48.8

On the stricter atomic audit of 29 move/replace/rotate cases, 16 failures fall into identifiable categories. Of four failed moves, one is geometric infeasibility, two leave the relation unresolved after verification/repair, and one grounds the wrong same-category target. Of five failed replacements, four miss or misbind the target and one has execution–evaluation synchronization failure after retrieval. Of seven failed rotations, all involve structured orientation parsing or orientation-verification/repair; the three `parallel_to_object` rotations succeed, while the `face_object` cases remain difficult. No audited failure is caused by asset retrieval.

Other failures involve interacting support, wall, and zone constraints, or residual physical cleanup in dense scenes. In compound edits, an insertion or replacement may succeed while a dependent spatial relation remains unmet.

E Limitations and Future Work

Evaluation scope and cross-turn constraints. Evaluation covers indoor scenes and short interaction chains, including 50 three-stage chains in the stress test (Appendix C.4.1); performance in open-ended sessions and other domains remains untested. Scene Memory persists, but Working Memory and protection sets reset with each instruction. Earlier requirements must enter the current program to remain enforced; the system does not automatically track their continued relevance or resolve conflicts. AuthorBench and MUSE were co-developed, although checks do not derive from method outputs (Appendix B.1).

Geometric reliability and representation. Local repair and the global feasibility pass do not guarantee physical feasibility. Precise transforms remain difficult, with 48.8% GF on scale-operation cases in the fixed editing split (Appendix D.2). The pipeline also depends on structured scene geometry, metadata, and asset coverage; missing or ambiguous references can prevent target grounding.

Runtime and next steps. Mean end-to-end editing time is 479.53 seconds per case (Appendix C.5). Future work should track active requirements across instructions, resolve conflicts, improve transform verification and repair, and reduce latency through selective verification and caching. Longer chains and less structured scenes would test broader applicability.

E.1 Broader impacts

Localized editing and explicit checks could aid interactive design and indoor simulation for embodied-AI research. Checks cover only specified properties: navigation results use a grid-based layout proxy, without a trained policy or physical deployment (Appendix C.4.2). Applications requiring accessibility, stability, or other unverified properties need additional validation and human review.