HDSL: A Hierarchical Domain-Specific Language for Structured 3D Indoor Scene Generation and Localized Editing with LLM Agents

Letian Li[1], Chao Shen[2], Shuzhao Xie[1], Chenghao Gu[1], ZhengXiao He[3], Yu Meng[1], Xin Yang[4], Wenyuan Jiang[5], Zhi Wang[1]*

[1]Shenzhen International Graduate School, Tsinghua University, China
[2]School of Computer Science, Shanghai Jiao Tong University, China
[3]Department of Electrical and Computer Engineering, University of Arizona, USA
[4]Zhejiang University, China, [5]D-INFK, ETH Zürich, Switzerland
lilt24@mails.tsinghua.edu.cn, wangzhi@sz.tsinghua.edu.cn

Abstract

Text-driven indoor scene generation and editing require an intermediate representation that language models can both produce and revise. Existing LLM-based systems often rely on scene graphs or global constraint lists, which are compact but underspecify local geometry and make instruction-based edits difficult to localize. We frame the task as structured program generation and local program repair, and propose the Hierarchical Descriptive Scene Language (HDSL), an XML/CSS-style domain-specific language in which rooms, regions, objects, and support surfaces form a tree with parent-relative local coordinates. Our pipeline generates HDSL subtrees recursively with bounded verification, grounds non-virtual nodes through multimodal asset retrieval, and applies force-directed layout optimization to repair boundary and collision errors. For editing, Hierarchical Retrieval-Augmented Generation retrieves the relevant subtree, rewrites only that local context, and merges the result back through a deterministic three-way merge. In our reproduced benchmark, HDSL improves average object coverage, text-scene alignment, and generation time over full text-to-scene baselines while remaining competitive with layout-only reproductions on geometry metrics. On thirty paired editing instructions, HRAG reduces token use by 67.6% and runtime by 68.1%, and cuts non-target removals by 87.5% over full-scene rewriting; attribute ablations confirm that local coordinates and virtual/leaf flags are necessary for valid, locality-preserving edits.

1 Introduction

Text-driven indoor scene generation and editing asks a system to convert a natural-language request into a renderable 3D indoor scene, and then revise that scene through follow-up instructions. This task supports VR (Höllein et al., 2023), AR (Suzuki et al., 2022), embodied AI (Duan et al., 2022), and rapid design prototyping, where users need both initial scene construction and iterative modification. Agentic LLM pipelines are especially promising here: instead of training a separate generator for every domain, an agent can decompose a prompt into planning steps, retrieve assets, call geometric checks, and revise intermediate outputs through tool feedback.

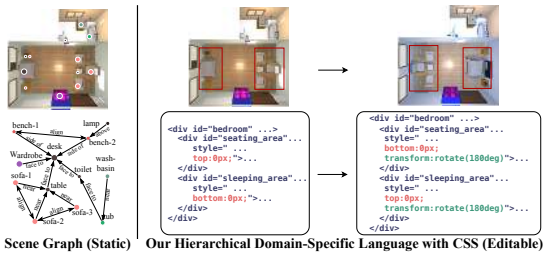


Figure 1: **HDSL as an editable scene interface.** Compared with flat scene graphs, an XML/CSS-style hierarchy stores local geometry, support relations, and object identity in addressable nodes, so an edit can target the relevant subtree instead of rewriting the whole scene.

The current agentic route, however, remains expensive and hard to control. Holodeck (Yang et al., 2024c) and I-Design (Çelen et al., 2025) ask LLMs to produce scene graphs or constraint sets before retrieving and arranging Objaverse-scale assets (Deitke et al., 2023); DirectLayout (Ran et al., 2025) and LayoutVLM (Sun et al., 2025) further explore language-driven layout construction. These systems demonstrate the flexibility of agents, but their scene interfaces often leave metric geometry and object identity distributed across solvers, validators, and post-processing state. As a result, a complex room may require long prompts, repeated retries, and global checks even when the user wants a small local change. Editing makes this cost more visible: follow-up instructions often require feeding back a large scene context or rerunning a broad generation process, which increases token use and raises the risk of moving, deleting,

* Corresponding author.

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 5519–5538
November 6-10, 2026 ©2026 Association for Computational Linguistics

or re-grounding unrelated objects (El Amine Boudjoghra et al., 2025).

These problems are difficult because indoor scenes are nested, support-heavy, and partly metric. A room contains functional zones, zones contain furniture groups, and many objects depend on tables, shelves, counters, or walls. A representation for an LLM therefore has to expose local coordinates, support relations, and persistent object identity while keeping each prompt small enough to reason over reliably. Prior work often treats this as a downstream repair problem, adding stronger validators, backtracking, render feedback, or ad hoc edit rules after the scene interface has already been chosen. These components are useful, but they do not make token consumption and localized editing the central design target.

We instead treat the scene representation as the control surface for the agent. The scene interface is what an LLM agent writes against and a geometry engine later executes, so its design deserves the same attention as the planner itself. Inspired by LLM-based web development (Xiao et al., 2025) and CSS layout generation (Feng et al., 2023), we introduce **Hierarchical Descriptive Scene Language (HDSL)**, an XML/CSS-style DSL for 3D indoor scenes. HDSL acts as a harness around the LLM: it constrains outputs to parseable nodes, stores local geometry in the parent frame, exposes deterministic checks, and gives each room, region, object, and support container an addressable identity, as shown in fig. 1. Its design commitments, tree containment, parent-relative local coordinates, virtual/leaf flags, and support nesting, make subtree addressing, local rewriting, re-anchoring, and merging atomic operations instead of conventions that every module must rebuild. A flat JSON list has no hierarchy and therefore no subtree to address; a scene graph records pairwise relations but no stable coordinate frame; a constraint list supports global reasoning but offers no local rewrite target. Our pipeline generates HDSL subtrees recursively, grounds concrete nodes through multimodal asset retrieval, repairs residual spatial errors with force-directed layout optimization, and edits scenes by retrieving and rewriting only the relevant subtree before a deterministic three-way merge. Our contributions are:

- We propose **HDSL**, a hierarchical XML/CSS-style scene DSL that turns 3D indoor generation into bounded, checkable, and addressable structured prediction for LLM agents.
- We design a **hierarchical generation pipeline** with recursive LLM agents, sibling-level coordination, generate–verify–revise checks, multimodal asset grounding, and force-directed layout optimization.
- We introduce **HRAG-HDSL** for localized instruction-based editing, backed by 30 paired single-step edits, 10 chained multi-step sequences, a retrieval diagnostic, and attribute ablations that isolate the contribution of the representation itself; HRAG reduces token use by 67.6% and runtime by 68.1% and cuts non-target removals by 87.5% over full-scene rewriting.

2 Related Work

Structured LLM Interfaces and DSLs. Many LLM systems ask the model to produce structured intermediate forms rather than free text. Declarative prompting can hand symbolic constraints to a solver (Ye et al., 2023), and program-generation approaches can move arithmetic or spatial reasoning into an external language or interpreter (Yang et al., 2024a; Gumin et al., 2025). These methods work because the output format constrains the model and exposes errors to deterministic checks. For indoor scene generation, however, the intermediate form must also carry support relations, local coordinates, and object identity so that later instructions can modify only the affected part of the scene.

Current LLM-based scene generators such as Holodeck (Yang et al., 2024c) and I-Design (Çelen et al., 2025) use scene graphs or constraint lists as the intermediate representation. These representations are compact for one-shot generation, but they do not expose a stable, addressable program surface for later edits. HDSL instead uses an XML/CSS-style representation: it keeps the syntax close to familiar web layout, but adds scene-specific attributes for 3D geometry, support relations, and retrieval metadata for downstream grounding.

Scene Representation. Indoor scene generation uses both 3D representations and text-facing representations. Point clouds and voxel grids are common for learned end-to-end models, while LLM-based systems often use scene graphs, constraint lists, or executable programs before asset retrieval and placement. Scene Language (Zhang et al., 2025), for example, uses a Lisp-like code form

and an interpreter to translate symbolic scene code into a final representation.

For generation with follow-up editing, the representation must do more than describe a scene: it must let the system locate the affected region, preserve unrelated content, and expose enough geometry for a local rewrite. Scene graphs are often too abstract for this purpose, while procedural programs can hide the final state behind control flow. HDSL uses a static declarative tree, so every generated object remains directly indexable and editable.

Learning-based Methods. Learned indoor-scene generators usually train directly on 3D scene data. DiffuScene (Tang et al., 2024) and PhyScene (Yang et al., 2024b) use layout diffusion for single-room generation. End-to-end diffusion methods, such as DiffInDScene (Ju et al., 2024) and LT3SD (Meng et al., 2025), synthesize finer-grained scenes from annotated 3D data.

However, learning-based scene generation models often need costly training and large datasets, making them difficult to apply to scene types with limited examples, such as industrial warehouses. Our approach does not train a new scene generator; it designs a recursive interface that lets a pretrained LLM plan unseen scene types through local, verifiable layout decisions.

LLM-based Methods. LLM-based scene generation methods can be broadly categorized by how they represent and resolve layouts. Constraint-based approaches such as Holodeck (Yang et al., 2024c) and I-Design (Çelen et al., 2025) have the LLM emit a set of spatial constraints, which are then solved by a search or backtracking algorithm. DirectLayout (Ran et al., 2025) and LayoutVLM (Sun et al., 2025) further study prompt-driven or vision-language layout optimization, but their public pipelines are most reproducible as layout-only systems with fixed or externally supplied object sets. HSM (Pun et al., 2026) introduces hierarchical scene motifs over HSSD-HAB assets, but its full pipeline depends on support-surface annotations and expensive VLM feedback. These systems improve different parts of the generation pipeline, yet they still provide limited support for partial scene modification without re-solving or regenerating a large context. Our method treats the scene representation itself as the model interface: each node is spatially self-contained within its parent, so relevant subtrees can be retrieved, edited, and merged back without changing the rest of the global scene state.

3 Methods

We view text-to-scene generation as structured prediction from a natural-language prompt P to a scene program $\Phi(\mathcal{S})$, and instruction-based editing as program repair from $(\Phi(\mathcal{S}), P_{\text{edit}})$ to an updated program $\Phi'(\mathcal{S})$. This formulation makes the interface between the LLM and the scene renderer explicit: the model writes and revises a parseable program, while deterministic modules validate geometry, retrieve assets, and merge local edits.

Our method is built around three requirements: a scene representation that exposes local geometry to the LLM, a generation procedure that plans complex rooms through bounded local decisions, and an editing procedure that changes only the relevant subtree, as shown in fig. 2.

3.1 Hierarchical Descriptive Scene Language

HDSL represents an indoor scene as an XML/CSS tree. Formally, a scene $\mathcal{S}$ is encoded as $\Phi(\mathcal{S}) = (\mathcal{N}, \mathcal{E})$, where each node $n \in \mathcal{N}$ denotes a room, region, object, or virtual support container, and each edge in $\mathcal{E}$ denotes a parent-child containment or support relation. Each node stores semantic, geometric, hierarchy-related, and retrieval-related attributes; the full schema and an example are provided in table 8 and fig. 3.

This representation is both a generation target and an editing index. CSS supplies local coordinates and rotation in the parent frame, while XML nesting keeps the context short and directly addressable. Unlike a flat JSON list or a symbolic scene graph, the parent-child syntax itself records containment, support, and local coordinate frames, so retrieval and editing can operate on subtrees rather than on a reconstructed global state. The tree also matches indoor structure: rooms decompose into functional zones, objects form local groups, and many small objects are placed on supporting surfaces. Table 1 summarizes the purpose and evidence behind each attribute. See Sec. A.2 for the full lifting rule.

3.2 Hierarchical Generation via LLM Agents

HDSL turns generation into local recursive planning rather than asking the LLM to instantiate the whole scene at once. For each expandable node n_i, the LLM receives its ancestor path $\mathcal{P}(n_i) = \{n_0 = r, \ldots, n_k\}$, parent geometry, inherited floorplan in-

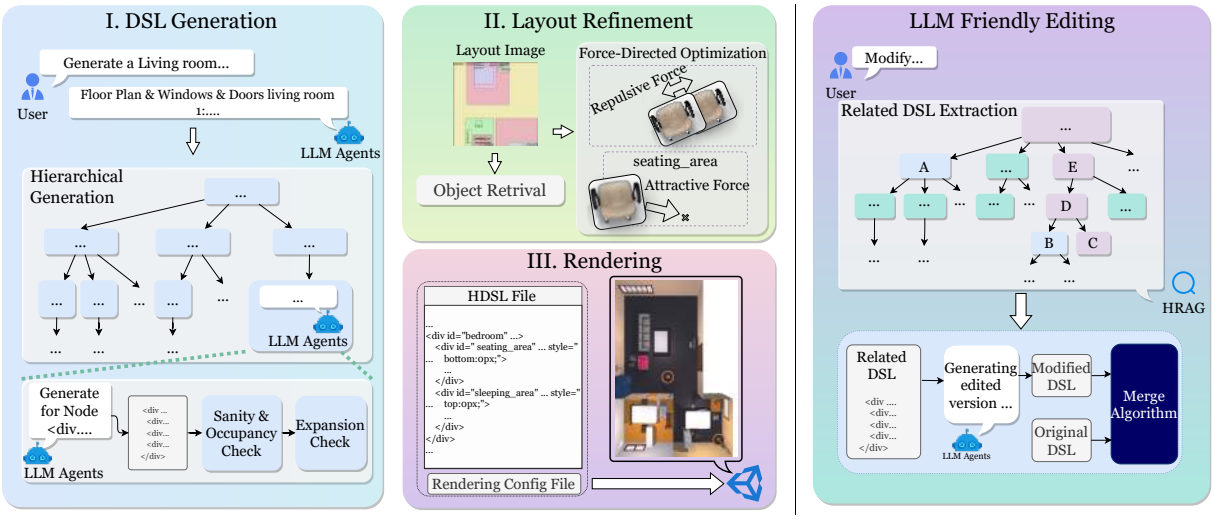


Figure 2: **HDSL framework.** We recursively generate an XML/CSS scene program with LLM agents, retrieve 3D assets, refine the layout with FDLO, render the scene, and support localized editing through HRAG and hierarchical merging for incremental updates.

Table 1: **HDSL design rationale and ablation evidence.** Each attribute serves a stated purpose; the evidence column reports the ablation or diagnostic from Section 4 that supports it.

Attribute	Purpose	Evidence
Local coordinates	Relative parent-child positioning; enables recursive generation and subtree re-anchoring	Global-coordinate variant: preservation 97.05% → 93.22%, collateral deletions 2.89 → 6.67
`virtual` / `is_leaf` flags	Mark recursive termination and bound expansion depth	Removing them: 0% parseable output
Support nesting	Attach objects to tables, shelves, counters, and floor	Qualitative validation; physical stacking and recursive placement
Node IDs + ancestor path	Address subtrees and retrieve edit context	Retrieval diagnostic: 94.0% context reduction, 100% target hit rate

```
<div id="living_room" description="A room with vertical 270 high wall cm"
    class="room" size="(800.0,600.0,0.0)" virtual="true" is_leaf="false"
    variance_type="N/A" style="left:0px;top:0px;width:800.0px;height:600.0px;">
    <div id="seating_group" description="a seating group with a 5-seat sofa"...>
        ...
    </div>
    <div id="tv_stand" description="a tv stand with a tv on top"...>
        ...
    </div>
    ...
</div>
```

Figure 3: **Example HDSL of a living room**. The snippet shows nested virtual zones, concrete objects, support relations, and local CSS-style spatial attributes. Due to space constraints, we show only the essential code needed to illustrate the hierarchy.

formation, global instructions, and few-shot examples (Dong et al., 2024). This makes each prompt a bounded structured-output task: the model only needs to emit the direct children of one container while preserving enough scene context for functional consistency.

At each step, the LLM decides the child set $\mathcal{C}(n_i)$ and emits, for each child c_j, its semantic description, class, 3D size, virtual flag, leaf flag, and CSS pose. The virtual flag distinguishes spatial regions or support containers from objects that require asset retrieval, while the leaf flag determines whether the child should be recursively expanded in later iterations. To avoid sparse or overcrowded local layouts, we score the occupancy of the generated children:

$$\rho(n_i) = \frac{\sum_{c_j \in \mathcal{C}(n_i)} \text{area}(c_j)}{\text{area}(n_i)}$$

together with validity and semantic coverage. We use a bounded multi-candidate generate–verify–revise loop: candidates are parsed as XML, checked for structural and spatial validity, repaired when deterministic fixes are safe, and otherwise rejected with feedback for the next attempt. Siblings are generated in one call so their relative positions are coordinated, while completed disjoint subtrees can be expanded in parallel. Additional prompt constraints, validators, and thresholds are provided in appendix A.4.

Object Retrieval. After HDSL generation, non-virtual nodes are grounded with Objaverse assets (Deitke et al., 2023) using the Holodeck retrieval pipeline (Yang et al., 2024c) and OpenCLIP

matching (Cherti et al., 2023). Retrieval details are provided in the appendix.

3.3 Force-Directed Layout Optimization

LLM-generated layouts may still contain boundary violations, sibling overlaps, overly tight virtual containers, or wall-facing objects. We therefore apply a recursive force-directed refinement within each parent node. For child i, FDLO combines a boundary attraction term and an overlap repulsion term:

$$\vec{F}_{\mathrm{a}} = \alpha \frac{\vec{D}_i}{\|\vec{D}_i\|}, \quad \vec{F}_{\mathrm{r}}^{ij} = \beta \, \mathrm{IoU}_{ij} \frac{\vec{\Delta}_{ij}}{\|\vec{\Delta}_{ij}\|}.$$

Here, $\vec{D}_i$ points from the child center toward the valid parent region, $\vec{\Delta}_{ij}$ separates overlapping siblings, and IoU_{ij} measures their 3D-box overlap. The child position is updated by:

$$p_i^{t+1} = p_i^t + \vec{F}_{\mathrm{a}} + \sum_{j \neq i} \vec{F}_{\mathrm{r}}^{ij}.$$

The implementation uses staged container expansion, boundary correction, overlap removal, clamping, and wall-facing orientation repair; details are provided in appendix A.6. During editing, the refinement is restricted to movable nodes whenever possible, preserving unrelated scene regions intact.

3.4 Incremental Editing via LLM Agents

The hierarchy gives editing an explicit retrieval unit: an instruction usually refers to an object, a support surface, or a local functional group rather than the whole scene. We therefore propose Hierarchical Retrieval-Augmented Generation (HRAG), inspired by RAG (Lewis et al., 2020), to retrieve the relevant HDSL context before calling the LLM. For each node, we concatenate its identifier, class name, description, and ancestor path as the retrieval feature; raw coordinates and sizes are not embedded, but are included in the retrieved DSL passed to the editor. A BGE-family encoder (Chen et al., 2024) embeds both node features and the edit instruction. HRAG combines embedding similarity, lexical matches, and intent-aware constraints: rearrangement edits keep a wider local context, while remove and replace edits use a narrower target set. For rearrangement, the movable-object phrase is parsed separately from the destination phrase, so an explicit destination container guides context without replacing the target object.

The retrieved partial DSL contains the target nodes, their ancestors, and sibling context, since the LLM must see nearby occupancy to edit without disrupting the parent container. The edit prompt requires one XML-parsable HDSL fragment with the same root identifier, preserves unrelated siblings and asset metadata, and forbids strategy text or markdown. The edited fragment is then merged back into the original scene.

We formalize the editing algorithm with three components: semantic retrieval, localized rewriting, and subtree merging. Given a scene $\Phi(\mathcal{S}) = (\mathcal{N}, \mathcal{E})$ and parent function $\pi(n) = p$ when $(p, n) \in \mathcal{E}$, HRAG computes the relevant node set $\mathcal{N}_{\mathrm{rel}}$ for edit instruction P_{edit}. Let $e_i = \mathrm{Embed}(f(n_i))$ and $e_P = \mathrm{Embed}(P_{\mathrm{edit}})$:

$$s_i = 100 \cdot \frac{\langle e_i, e_P \rangle}{\|e_i\| \|e_P\|},$$
$$s_{\max} = \max_{n_i \in \mathcal{N}} s_i,$$

where $f(n_i)$ denotes the path-aware textual feature of node n_i. Let δ_e be the intent-dependent similarity gap for P_{edit}, and let b_i be the keyword bonus from lexical overlap with the node identifier, class, and description. HRAG selects:

$$C_e = \{n_i \in \mathcal{N} \mid q_i\},$$
$$q_i \equiv s_{\max} - s_i \leq \delta_e \ \vee \ b_i > 0,$$
$$\mathcal{N}_{\mathrm{rel}} = \mathrm{TopK}_{K_{\max}}(C_e, s_i + b_i).$$

If fewer than $K_{\min}$ nodes are selected, we fall back to the top-$K_{\min}$ nodes by embedding similarity. We then construct the subtree $\Phi_{\mathrm{rag}}(\mathcal{S}) = (\mathcal{N}_{\mathrm{rag}}, \mathcal{E}_{\mathrm{rag}})$ that satisfies:

1. $\mathcal{N}_{\mathrm{rel}} \subseteq \mathcal{N}_{\mathrm{rag}}$,
2. $\mathcal{N}_{\mathrm{sib}} = \{m \in \mathcal{N} \mid \exists n \in \mathcal{N}_{\mathrm{rel}}, \pi(n) = \pi(m)\} \subseteq \mathcal{N}_{\mathrm{rag}}$,
3. $\forall n \in \mathcal{N}_{\mathrm{rag}}, \pi(n) \in \mathcal{N}_{\mathrm{rag}}$ if $\pi(n)$ exists.

Next, the LLM edits $\Phi_{\mathrm{rag}}(\mathcal{S})$ to produce $\Phi'_{\mathrm{rag}}(\mathcal{S})$. Finally, we merge the edited subtree and apply the refinement pipeline to generate the updated scene. The merge algorithm performs a three-way reconciliation over the original $\Phi(\mathcal{S})$, the unedited partial subtree $\Phi_{\mathrm{rag}}(\mathcal{S})$, and the edited subtree $\Phi'_{\mathrm{rag}}(\mathcal{S})$, as shown in fig. 4. It applies edited attributes only to selected target subtrees, preserves original nodes included only as spatial context, allows deletion only for remove/replace

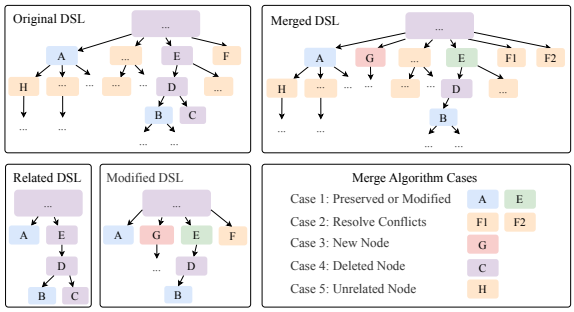


Figure 4: **Three-way merge.** The merge reconciles the original DSL, retrieved context, and edited subtree into a final HDSL while preserving unmodified nodes. The full algorithm is in the appendix.

intents, and inherits asset metadata for unchanged objects. The merge operation is:

$$\text{Merge}\left(\Phi, \Phi_{\text{rag}}, \Phi'_{\text{rag}}\right) \rightarrow \Phi_{\text{final}}.$$

Here, Φ_{final} denotes the merged output, which is further refined in subsequent steps. The synthesized DSL, along with the scene configuration file, is then rendered using the simulation framework introduced by AI2-THOR (Kolve et al., 2017).

4 Experiments

We evaluate **HDSL** through four questions: whether the representation improves text-prompted scene generation, whether hierarchical planning and FDLO account for the gains, whether HRAG makes instruction-based edits cheaper and more local than rewriting the whole scene, and whether the representation attributes themselves drive the editing behavior. Unless otherwise specified, HDSL uses Qwen2.5-72B-Instruct (Qwen et al., 2025). The editing experiments test a model-interface property: whether an instruction can be grounded to the relevant structured context without forcing the LLM to rewrite unrelated scene content.

4.1 Text-Prompted Generation

Baselines and scope. We separate full text-to-scene generation from layout-only diagnostics. HDSL, Holodeck (Yang et al., 2024c), and I-Design (Çelen et al., 2025) are complete systems with asset retrieval and rendering. DirectLayout (Ran et al., 2025) and LayoutVLM (Sun et al., 2025) are reported only as geometry diagnostics after conversion to our scene JSON format, so their NObj counts are layout slots rather than rendered objects. We select six MIT Scene Dataset categories (Quattoni and Torralba, 2009); HDSL, Holodeck, I-Design, and DirectLayout use five runs per category, while LayoutVLM contributes one adapted layout per category. HSM (Pun et al., 2026) is excluded from the six-category main table because the reproducible public setup depends on HSSD-HAB support annotations and paired prompts; its three-prompt check is reported in the appendix. Table 2 situates the underlying representations along the axes that matter for generation and editing.

Metrics. Following (Çelen et al., 2025; Lin and Mu, 2024; Tang et al., 2024), we evaluate generation with NObj, OOB (Out-of-Bounds), PIoU, CLIP, and generation time; formal definitions and timing scope are provided in the appendix.

Results. table 3 shows the main trade-off. Among full text-to-scene systems, HDSL has the highest average object count (58.19 versus 43.15 for Holodeck and 13.20 for I-Design), the best average CLIP score (20.99), and the lowest average generation time (383.31s). The gain is largest in offices, gyms, classrooms, and warehouses, where flat global planning often under-fills the room or places objects near walls. HDSL does not obtain the lowest PIoU overall because I-Design produces much sparser scenes, but it keeps PIoU low while adding far more objects. DirectLayout and LayoutVLM are not used to support end-to-end claims; they show that HDSL remains competitive on converted layout geometry while solving the harder asset-grounded scene-generation task. We further report HSM (Pun et al., 2026) in the appendix as a paired reproducibility check, including the partial public-pipeline case.

4.2 Instruction-Based Editing

HDSL exposes an editable node for each object, support surface, and functional region, so natural-language instructions can be converted into local subtree rewrites rather than full-scene regeneration. fig. 6 shows object movement, local swaps, scene enrichment, and reasoning-based edits.

We quantify locality on thirty paired single-step instructions, using the same base scenes and instructions for HRAG-HDSL and for a full-HDSL rewrite baseline that gives the LLM the whole scene and asks for a complete edited scene. Geometric validity alone does not characterize editing quality: as the eight-case stress check in the appendix shows, a rewritten scene can pass OOB

Table 2: **Design space of LLM-facing scene representations.** Holodeck uses a scene graph, I-Design a constraint list, and Scene Language and HSM hierarchical scene graphs. Hierarchy: nesting into regions and groups; Coords: parent-relative geometry; Addr.: a stable path identifies a local group; Rewrite: in-place group edits are native; Gen.: LLM scene generation; Edit.: instruction-based editing; Reprod.: pipeline runs from public assets and code. ✓/○/×: yes/partial/no.

System	Hierarchy	Coords	Addr.	Rewrite	Gen.	Edit.	Reprod.
HDSL (ours)	✓	✓	✓	✓	✓	✓	✓
Holodeck	×	×	×	×	✓	×	✓
I-Design	×	×	×	×	✓	×	○
Scene Language	○	○	×	×	✓	×	○
HSM	✓	×	○	×	✓	×	○

Table 3: **Quantitative results on generation** across distinct scene categories. The full-system comparison is between HDSL, Holodeck, and I-Design. Layout-only rows are marked with † and are geometry diagnostics, not entries in the full text-to-scene leaderboard. **Bold** and *italics* mark best and second-best results among full scene-generation systems only.

Method	Metric	Apartment	Bedroom	Classroom	Gym	Office	Warehouse	**Avg.**
Holodeck (Yang et al., 2024c)		**41.40**	**26.40**	*36.71*	*30.20*	*77.60*	*46.60*	*43.15*
I-Design (Çelen et al., 2025)	NObj↑	9.00	*11.00*	18.00	10.00	17.00	14.20	13.20
DirectLayout† (Ran et al., 2025)		5.60	5.00	27.60	8.40	9.40	6.20	10.37
LayoutVLM† (Sun et al., 2025)		13.00	8.00	28.00	14.00	25.00	22.00	18.33
HDSL (ours)		*25.67*	7.43	**54.80**	**52.75**	**135.08**	**73.40**	**58.19**
Holodeck (Yang et al., 2024c)		*0.13*	**0.05**	**0.00**	*0.02*	*0.15*	**0.00**	*0.06*
I-Design (Çelen et al., 2025)	OOB↓	0.67	0.58	0.39	0.62	0.46	0.54	0.54
DirectLayout† (Ran et al., 2025)		0.00	0.04	0.00	0.12	0.00	0.00	0.03
LayoutVLM† (Sun et al., 2025)		0.00	0.12	0.00	0.21	0.12	0.05	0.08
HDSL (ours)		**0.03**	*0.15*	*0.01*	**0.01**	**0.04**	*0.07*	**0.05**
Holodeck (Yang et al., 2024c)		2.18	6.96	1.31	0.74	1.02	2.48	2.45
I-Design (Çelen et al., 2025)	PIoU↓	**0.02**	**0.01**	**0.01**	**0.01**	**0.00**	**0.00**	**0.01**
DirectLayout† (Ran et al., 2025)		0.75	0.04	0.43	0.37	1.68	0.28	0.59
LayoutVLM† (Sun et al., 2025)		0.61	0.09	0.03	0.98	0.03	1.40	0.52
HDSL (ours)		**0.02**	*0.03*	**0.01**	*0.03*	*0.01*	*0.35*	*0.07*
Holodeck (Yang et al., 2024c)		**23.53**	**26.05**	17.26	*19.42*	9.23	**23.34**	*19.81*
I-Design (Çelen et al., 2025)	CLIP↑	14.68	*25.47*	**26.03**	13.34	**15.93**	22.03	19.58
HDSL (ours)		*22.65*	23.78	*24.55*	**22.12**	*10.31*	*22.51*	**20.99**
Holodeck (Yang et al., 2024c)		637.90	389.40	463.44	416.80	839.43	567.10	552.34
I-Design (Çelen et al., 2025)	Time↓	842.90	687.10	871.80	925.20	813.60	952.70	848.90
HDSL (ours)		**303.67**	**278.20**	**434.33**	**264.75**	**597.00**	**421.89**	**383.31**

† DirectLayout and LayoutVLM are layout-only reproductions: NObj counts converted layout slots, and CLIP/time are omitted because these public pipelines do not include the same asset retrieval, rendering, or refinement stack.

and PIoU checks while deleting objects or moving unrelated furniture. On the thirty-instruction benchmark, both methods produce valid DSL on every case, but HRAG uses 67.6% fewer tokens and 68.1% less runtime, and it reduces non-target removals from 1.33 to 0.17 per edit (−87.5%), as shown in table 4.

Table 4: **Instruction-based editing on thirty paired instructions.** HRAG-HDSL retrieves and rewrites the relevant subtree; the full-HDSL rewrite baseline regenerates the complete scene with the same model and refinement stack.

Metric	HRAG-HDSL	Full HDSL rewrite
Valid DSL ↑	100.0%	100.0%
Fallback ↓	0.0%	0.0%
Tokens ↓	**5.4k**	16.6k
Runtime ↓	**50.5s**	158.4s
Removed non-target ↓	**0.17**	1.33

4.3 Ablation Study

We perform ablation studies to test whether the gains come from the proposed representation and localization mechanisms. The experiments focus on hierarchical generation with refinement and partial editing through HRAG.

Ablation on Generation Components. We evaluate three configurations on the same six scene types: the full model, direct HDSL generation without hierarchical generation (HG), and hierarchical gen-



Figure 5: **Qualitative results on generation** of residential homes, commercial spaces, industrial settings, and public areas. The examples illustrate denser object coverage and more even use of large open areas.

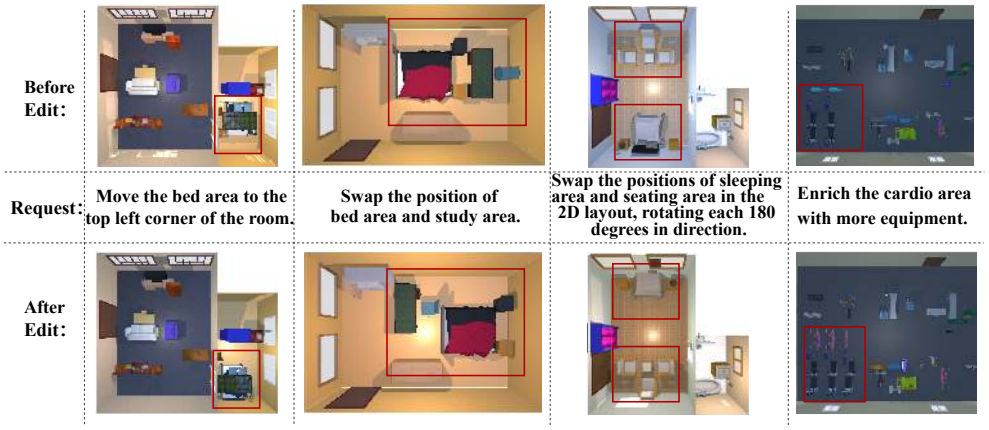


Figure 6: **Editing examples.** HDSL supports object movement, local swaps, scene enrichment, and reasoning-based edits while preserving nearby spatial relationships.

eration without force-directed layout optimization (FDLO). Since LLM generation is stochastic, we compare relative trends within this controlled run. table 5 shows that HG increases object coverage and CLIP alignment, while FDLO mainly reduces boundary and overlap errors. Their combination gives the best overall trade-off.

Ablation on Representation Design. The attribute ablation in table 6 isolates the representation itself. We fix the pipeline and prompts and vary only the HDSL encoding on nine editing cases: the full encoding, a variant that replaces parent-relative local coordinates with global coordinates, and a variant that removes the `virtual` and `is_leaf` flags. The global-coordinate variant still completes every target edit, but collateral damage rises: average preservation drops from 97.05% to 93.22%, and av-

Table 5: **Generation ablation** on pipeline components. HG increases object coverage and CLIP alignment, while FDLO reduces spatial violations.

HG	×	√	×	√
FDLO	×	×	√	√
NObj ↑	8.46	*55.82*	14.10	**65.23**
OOB ↓	0.40	0.36	**0.03**	*0.10*
PIoU ↓	**0.03**	**0.03**	0.09	*0.05*
CLIP ↑	17.62	*20.71*	16.96	**20.76**

erage non-target removals more than double, from 2.89 to 6.67. Removing the flags is more decisive, since the output no longer parses under the HDSL schema and validity falls to 0%. Local coordinates and the termination flags are load-bearing rather than cosmetic; Table 2 contrasts these commitments with the representations used by prior systems.

Table 6: **HDSL attribute ablation** on nine editing cases. The pipeline and prompts are fixed; only the encoding varies. Both parseable variants complete every target edit.

Encoding	Valid ↑	Preserv. ↑	Removed ↓
HDSL-full	100.0%	**97.05%**	**2.89**
HDSL-global	100.0%	93.22%	6.67
HDSL-no-virtual	0.0%	–	–

Retrieval Diagnostic. HRAG only helps if it selects the right context before the LLM call. We annotate gold target nodes with a scene-vocabulary procedure (appendix) on nine cases, seven with spatial references and two without, and measure what the retriever selects. Base HRAG reduces the editing context to 6% of the full scene while hitting at least one target object in every case, at 84.1% precision, 25.0% target recall, and 14.2% oracle recall; the moderate recall is expected, since bulk instructions name a functional group and retrieving the whole group works against the small-context design goal. Appendix A.10 reports a geometry-aware pre-filter variant that resolves spatial phrases such as "near the window" before embedding retrieval.

Chained Multi-Step Editing. Single-step edits do not test whether a representation survives repeated modification, so we chain 10 sequences of two to three edits in which later instructions refer to objects changed earlier, and compare HRAG-HDSL against a flat-JSON editing interface backed by the same LLM. End-to-end success is tied at 30% for both, but region-coherence drift separates the two interfaces (table 7): HDSL keeps the referenced functional group nearly intact at every step, while flat JSON scatters the group once later instructions depend on it. Drift is the average change in pairwise relative offsets among the leaf objects of the group named in the instruction, where zero means the internal arrangement is unchanged; tree addressing keeps member arrangements stable when an instruction refers to a group such as the "bed area", while a flat object list has no group to hold onto.

Table 7: **Chained multi-step editing** over 10 sequences. Region-coherence drift measures the change in internal pairwise offsets of the referenced functional group; lower is better.

Drift ↓	Step 1	Step 2	Step 3
HDSL-HRAG	**60.6**	**34.5**	**0.2**
Flat JSON	104.3	566.7	7.7

4.4 User Study

We recruited 20 participants for a perceptual study. Participants were general users rather than method authors, did not see system names, and were not required to have specialized 3D modeling background. For generation, each participant evaluated six anonymized triplets from HDSL, Holodeck (Yang et al., 2024c), and I-Design (Çelen et al., 2025), giving 120 votes that we convert into Bradley–Terry scores (Bradley and Terry, 1952). For editing, participants rated nine rendered edits on a three-point compliance scale, giving 180 votes; 82.2% judged the edit as exactly or mostly matching the instruction, which complements the automatic displacement and preservation metrics.

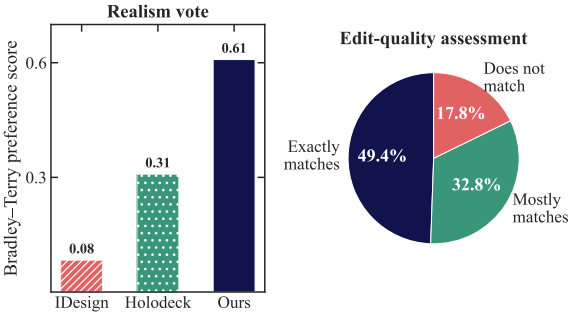


Figure 7: **User study results.** Participants prefer HDSL generations over Holodeck and I-Design, and 82.2% of editing votes rate the result as exactly or mostly matching the editing instruction overall.

5 Conclusion

We presented HDSL, a hierarchical DSL that turns 3D indoor scene generation and editing into bounded structured prediction for LLM agents. The same representation supports recursive generation, deterministic checking, asset grounding, force-directed refinement, and HRAG-based local editing. Across generation, ablation, and editing experiments, HDSL improves scene coverage and edit efficiency while preserving a readable node-level interface for inspection and repair.

Limitations

The generation benchmark covers six scene categories with five runs per category, so the reported effects are averages over a limited sample rather than inferential tests. We report robustness checks with alternative LLMs and the expanded thirty-case editing benchmark in Section 4 and the appendix, and we restrict strong claims to effects that are consistent across these checks. This work focuses on language-mediated scene structure: how an LLM can write, check, retrieve, and locally revise a 3D indoor scene. HDSL assumes a 3D asset database with reasonable category coverage, so rare-object visual fidelity depends on the retrieval corpus even when geometry is valid. FDLO targets static indoor layouts by repairing bounding-box collisions and boundary violations; dynamics, articulation, and task-level affordances can be layered through simulators or specialized planners. HRAG is most direct for edits with identifiable spatial targets, while broad style changes may retrieve larger contexts or use full-scene rewriting. Recent baselines are reported under public-release boundaries, and editing success is measured with strict displacement diagnostics alongside human compliance judgments for edits.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 92467204 and 62472249) and the Shenzhen Science and Technology Program (Grant No. KJZD20240903102300001).

References

Ralph Allan Bradley and Milton E. Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. 2025. I-design: Personalized llm interior designer. In *Computer Vision – ECCV 2024 Workshops*, volume 15624 of *Lecture Notes in Computer Science*, pages 217–234. Springer.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. 2023. Reproducible scaling laws for contrastive language-image learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2829.

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

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. 2022. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244.

Mohamed El Amine Boudjoghra, Ivan Laptev, and Angela Dai. 2025. Scanedit: Hierarchically-guided functional 3d scan editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 27105–27115.

Weixi Feng, Wanrong Zhu, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Xuehai He, S Basu, Xin Eric Wang, and William Yang Wang. 2023. Layoutgpt: Compositional visual planning and generation with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 18225–18250. Curran Associates, Inc.

Maxim Gumin, Do Heon Han, Seung Jean Yoo, Aditya Ganeshan, R. Kenny Jones, Kailiang Fu, Rio Aguina-Kang, Stewart Morris, and Daniel Ritchie. 2025. Procedural scene programs for open-universe scene generation: Llm-free error correction via program search. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, SA Conference Papers '25, New York, NY, USA. Association for Computing Machinery.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. 2021. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7514–7528, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. 2023. Text2room: Extracting textured 3d meshes from 2d text-to-image models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7909–7920.

Xiaoliang Ju, Zhaoyang Huang, Yijin Li, Guofeng Zhang, Yu Qiao, and Hongsheng Li. 2024. Diffindscene: Diffusion-based high-quality 3d indoor scene generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4526–4535.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv preprint arXiv:1712.05474*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Chenguo Lin and Yadong Mu. 2024. Instructscene: Instruction-driven 3d indoor scene synthesis with semantic graph prior. In *International Conference on Learning Representations (ICLR)*.

Quan Meng, Lei Li, Matthias Nießner, and Angela Dai. 2025. Lt3sd: Latent trees for 3d scene diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 650–660.

Hou In Derek Pun, Hou In Ivan Tam, Austin T. Wang, Xiaoliang Huo, Angel X. Chang, and Manolis Savva. 2026. HSM: Hierarchical scene motifs for multi-scale indoor scene generation. In *2026 International Conference on 3D Vision (3DV)*, pages 1356–1367. IEEE.

Ariadna Quattoni and Antonio Torralba. 2009. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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

Xingjian Ran, Yixuan Li, Linning Xu, Mulin Yu, and Bo Dai. 2025. Direct numerical layout generation for 3d indoor scene synthesis via spatial reasoning. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. 2025. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29469–29478.

Ryo Suzuki, Adnan Karim, Tian Xia, Hooman Hedayati, and Nicolai Marquardt. 2022. Augmented reality and robotics: A survey and taxonomy for ar-enhanced human-robot interaction and robotic interfaces. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA. Association for Computing Machinery.

Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. 2024. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R. Lyu. 2025. Designbench: A comprehensive benchmark for mllm-based front-end code generation. *Preprint*, arXiv:2506.06251.

Xiaocheng Yang, Bingsen Chen, and Yik-Cheung Tam. 2024a. Arithmetic reasoning with LLM: Prolog generation & permutation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 699–710, Mexico City, Mexico. Association for Computational Linguistics.

Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. 2024b. Physcene: Physically interactable 3d scene synthesis for embodied ai. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. 2024c. Holodeck: Language guided

generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16227–16237.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: Satisfiability-aided language models using declarative prompting. In *Proceedings of NeurIPS*.

Yunzhi Zhang, Zizhang Li, Matt Zhou, Shangzhe Wu, and Jiajun Wu. 2025. The scene language: Representing scenes with programs, words, and embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24625–24634.

Appendix

A Additional Implementation Details

A.1 Floorplan-to-DSL Conditioning

Throughout the generation process, we first follow the Holodeck methodology to generate floorplans based on user-provided prompts, including the positions of rooms, windows, and doors. This information is subsequently converted into the description field within the DSL attributes for later hierarchical scene generation and editing.

A.2 From 2D CSS Layout to 3D Poses

Each node uses a 2D CSS footprint in the parent plane and is lifted to 3D by fitting the retrieved asset's 3D bounding box to the corresponding floor region with rotation. The CSS coordinates define the (x, z) ground-plane projection, while the vertical coordinate y is determined through **place-on semantics**: each child lies on the supporting surface of its parent, with $y = y_{\text{parent}} + h_{\text{surface}}$. For objects with multiple supporting surfaces, such as bookshelves, the parent is expanded into child virtual containers whose geometry provides the corresponding vertical offset. The CSS `transform: rotate` property maps to rotation around the vertical axis, and the retrieved asset is scaled to the node's `size` attribute.

A.3 HDSL Node Schema and Example

Table 8: **Specification** of HDSL node attributes.

Attr.	Description	Attr.	Description
`id`	A unique identifier.	`virtual`	Whether it binds to a retrieved asset.
`style`	CSS properties specifying the layout.	`is_leaf`	Whether it is a leaf.
`class`	Specifies the asset class name.	`variance_type`	Variance type of sibling assets.
`size`	The 3D bounding-box size.	`description`	A textual description of the node.

HDSL follows an HTML/CSS-like structure: a scene is composed of nested div-style elements, and each element represents a room, region, object, or virtual support container. This choice uses LLMs' prior exposure to web code while keeping coordinates local to each parent. It also matches indoor-scene structure, where rooms decompose into zones, zones decompose into object groups, and gravity creates many place-on relations.

A.4 LLM Setup and Sanity Checks

We employ Qwen2.5-72B-Instruct (Qwen et al., 2025) as the base model for the reported LLM-agent scene generation and editing runs. The executable generation prompt requires the model to return exactly one XML-parsable HDSL fragment rooted at the requested node, with no natural-language strategy, markdown fence, heading, or explanatory text. We adopt a bounded multi-candidate generate–verify–revise loop for quality control. At each node, the implementation samples multiple candidates, parses them into HDSL, applies deterministic repairs when safe, and then scores candidate quality. Regeneration is triggered by malformed XML, duplicate identifiers, root mismatch, children outside parent bounds, invalid support nesting, emitted clearance/aisle nodes, excessive overlap, door/window blockage, implausible orientation, unrealistic asset-size assumptions, missing functional coverage, or extreme occupancy. Each node has a maximum attempt budget; if the budget is exceeded, the best valid candidate is used when available, otherwise the current branch is safely terminated without further expansion.

A.5 Parallel Expansion Policy

Parallelization is limited to recursive expansion of disjoint subtrees, not sibling generation. All siblings are generated jointly in a single LLM call, enabling explicit spatial coordination and avoiding intra-container conflicts. Subsequent parallel expansion is safe since each branch remains within its parent's footprint. A final force-directed layout refinement resolves residual overlaps and boundary violations during recursive refinement.

A.6 Force-Directed Layout Optimization Details

FDLO runs independently inside each parent node and is then applied recursively down the hierarchy. The attraction term uses $\vec{D}_i$ to move an out-of-bound child toward the valid interior of its parent. The repulsion term uses $\vec{\Delta}_{ij}$ to separate overlapping siblings, scaled by their 3D bounding-box IoU_{ij} so larger collisions receive stronger corrections. In practice, we first expand virtual containers when their children exceed the current footprint, then run boundary-focused refinement, balanced boundary-overlap refinement, and a final fine pass. The optimizer uses adaptive force scaling, boundary clamping, overlap post-processing, and automatic orientation correction for wall-adjacent objects. During editing, the same refinement pipeline is constrained to the movable subset when possible, which keeps unrelated scene regions stable. The main ablation quantifies the effect: adding FDLO to direct HDSL generation reduces OOB from 0.40

to 0.03, while adding FDLO to hierarchical generation reduces OOB from 0.36 to 0.10 and keeps PIoU low (0.03 to 0.05) despite denser scenes.

A.7 Implementation hyperparameters

The hyperparameters are as follows:

```
generation_max_attempts = 5
generation_num_candidates = 2
generation_accept_score = 0.58
occupancy_ratio_virtual_min = 0.12
occupancy_ratio_preferred_max = 0.8
occupancy_ratio_hard_max = 0.9
min_expand_area = 900
max_expand_depth = 5
retrieve_top_k = 10
related_editing_threshold = 5
edit_min_selected_nodes = 2
edit_max_selected_nodes = 8
edit_max_attempts = 6
FDLO_iterations = 500
FDLO_base_repulsion_force = 0.02
FDLO_boundary_force_multiplier = 10.0
FDLO_overlap_iou_eps = 0.02
FDLO_overlap_post_process_passes = 8
FDLO_virtual_container_fit_margin = 0.20
```

Candidate Quality and Occupancy. For each generated node, the implementation computes child-area occupancy and uses it as one component of a broader quality score. For virtual containers, occupancy below 0.12 is rejected as under-specified; occupancy up to 0.8 is considered preferred, and occupancy above 0.9 is rejected as overcrowded. For concrete non-leaf objects, the occupancy check is relaxed because small supported items may occupy only a small portion of the support surface. The final acceptance decision also considers bounds validity, sibling overlap, distribution, support semantics, door/window clearance, wall-facing orientation, functional coverage, and asset-size realism.
Expansion Check Threshold. Nodes with an area smaller than min_expand_area are treated as leaves. Furthermore, if the current node's depth exceeds max_expand_depth, expansion is halted to maintain computational efficiency and prevent over-segmentation of the scene hierarchy. Structural storage nodes such as shelves and bookcases may also be kept as leaves to avoid generating unsupported internal artifacts during expansion.
Object Retrieval Threshold. We use Objaverse (Deitke et al., 2023) assets and follow the Holodeck retrieval pipeline (Yang et al., 2024c). The retriever combines OpenCLIP (Cherti et al., 2023) visual-text similarity, Sentence-BERT (Reimers and Gurevych, 2019) textual similarity, category constraints, physical-size checks, and support-surface compatibility. Candidate retrieval is thresholded by semantic similarity and falls back to catalog-level category matches when strict retrieval is empty. The candidate set is capped by $retrieve_top_k = 10$ before asset assignment.
Related Editing Node Threshold. The released implementation uses a BGE-small-en-v1.5 encoder by default for HRAG node retrieval. For each node, the retrieval text concatenates its identifier, class, description, and ancestor context; coordinates and sizes are not part of the embedding text, but remain available in the retrieved HDSL fragment. Similarities are scaled by 100, and all nodes whose score is within $related_editing_threshold = 5$ of the maximum score are considered candidates. The gap is intent-aware: rearrangement uses a slightly wider context, while remove/replace uses a narrower one. For rearrangement, relation markers such as "closer to" and "to the center" are used to extract the movable-object phrase before container filtering, so a destination region does not override the object being moved. We match explicit node identifiers exactly when they appear in the movable phrase, add a lexical keyword bonus for direct matches with object ids/classes/descriptions, prune to direct target objects, and cap the final set to $edit_max_selected_nodes = 8$ with a fallback minimum of $edit_min_selected_nodes = 2$.
FDLO Defaults. FDLO is implemented as staged recursive refinement. Each parent node is optimized for up to 500 iterations with base repulsion 0.02, boundary force multiplier 10.0, overlap IoU epsilon 0.02, and 8 overlap post-processing passes. Before iterative refinement, virtual containers can be expanded to fit their children with a 0.20 margin and 1.25 area slack. After refinement, the implementation applies boundary clamping and automatic orientation correction for wall-adjacent objects after layout optimization.

A.8 Evaluation Metrics

Following prior text-to-scene evaluation protocols, we report five metrics. NObj counts placed object instances for full text-to-scene systems and converted layout slots for layout-only diagnostics. OOB (Out-of-Bounds) is the fraction of generated objects intersecting room boundaries. PIoU measures inter-object collision as the average pairwise 3D bounding-box intersection-over-union. CLIP uses CLIP similarity (Radford et al., 2021; Hessel et al., 2021) between rendered images and text

prompts to estimate text-scene alignment. Time is wall-clock time from the start of scene generation to a saved renderable scene, including LLM calls, parsing, asset retrieval, layout refinement, and scene serialization; it excludes one-time downloads and offline metric computation.

For recent baselines whose public code does not produce the same AI2-THOR scene configuration, we convert saved layouts into a common scene JSON and run the same NObj, OOB, and PIoU metrics. We do not report CLIP or end-to-end runtime for layout-only reproductions unless the baseline includes comparable asset retrieval, rendering, and refinement stages. All full-system timing rows were produced in the same local experiment environment, while layout-only rows are used only for geometry diagnostics in the table.

A.9 Editing Ablation Protocol

The editing ablation compares HRAG-HDSL with a full-HDSL rewrite baseline under the same base scenes and edit instructions. HRAG-HDSL retrieves the relevant subtree, asks the LLM to rewrite only that local context, and merges the result back. The full-HDSL rewrite baseline gives the LLM the complete HDSL scene and asks for a full edited scene, using the same model, asset reuse setting, coordinate recalculation, and refinement pipeline.

We evaluate three groups of signals. Target changed measures whether the intended object moves by at least 0.15m. This threshold is a strict furniture-scale diagnostic for the eight edit cases; it is not a universal semantic-success threshold and would be too coarse for small tabletop objects. We therefore also report continuous target displacement in table 9. Locality measures the mean displacement and moved rate of non-target objects; stable preservation excludes the target object and objects that were explicitly moved by post-processing repair. Metadata preservation checks whether unchanged objects keep their asset identifiers and retrieved asset metadata. This protocol is deliberately stricter than geometric validity alone: a method can have low OOB and PIoU while still failing the editing task by deleting objects or moving unrelated furniture items.

Expanded editing stress check. We evaluate eight paired edit instructions across apartments, classrooms, restaurant kitchens, and studio scenes. This run tests whether local editing remains parseable and content-preserving under the same layouts and instructions used for the full-HDSL rewrite baseline. For HRAG-HDSL, all cases produce valid DSL without fallback, preserve asset metadata, and remove no objects. Under the strict 0.15m target-change threshold used in Table 9, five of eight cases pass; the remaining cases are small-displacement edits where the target remains preserved but moves less than the threshold. As shown in Table 9, local editing uses far fewer LLM tokens and much less time while reducing non-target displacement and object removal.

Method	Cases	Target	Valid/fallback	Stable Δ	Moved	Removed	Cost
HRAG-HDSL	8	62.5%	100.0% / 0.0%	0.018m	4.3%	0.00	4.1k / 72.1s
Full HDSL rewrite	8	50.0%	50.0% / 50.0%	0.200m	23.8%	1.50	21.4k / 445.8s

Table 9: Expanded editing stress check on eight paired instructions. Target reports the fraction of cases whose intended object moves by at least 0.15m; valid/fallback reports DSL parse success and fallback rate, and stable Δ and moved rate are computed on stable non-target objects after editing.

Stress-check pattern. The non-exact HRAG cases in this stress check are threshold misses rather than parser or object-deletion errors: the intended object remains present and metadata-preserved, but its displacement is smaller than 0.15m after refinement. The three cases are the apartment coffee-table edit (0.144m), the classroom teacher-desk edit (0.054m), and the restaurant sink edit (0.039m). Full-scene rewriting shows a different profile. Four of eight full rewrites fall back to the original scene after the attempt budget, and two valid classroom rewrites remove six non-target objects each. Even when the output is geometrically valid, unrelated objects are more likely to move or disappear. This difference is why we report target change, parser reliability, and preservation metrics instead of relying only on OOB and PIoU.

A.10 Expanded Editing Benchmark and Diagnostics Protocol

Model. The thirty-instruction editing benchmark, the attribute ablation, the retrieval diagnostic, and the chained multi-step study are run with Qwen3-Max. All variants share the same pipeline, prompts, and refinement stack, and we report them as relative comparisons under matched conditions.

Thirty-instruction benchmark. The benchmark extends the original eight paired edits with twenty-two new instructions across apartments, bedrooms, classrooms, gyms, offices, and studios. Both methods see the same base scene and instruction. Preser-

vation is computed from node-id sets before and after the edit, and a removal is labeled non-target when the removed node does not match the instruction target keywords.

Attribute ablation. The three encoding variants keep the tree structure and prompts fixed. HDSL-global rewrites every node's position in room-level global coordinates, so the LLM must regress absolute positions during edits. HDSL-no-virtual drops the `virtual` and `is_leaf` attributes; the released parser requires both, which is why validity drops to zero instead of degrading gracefully.

Retrieval gold annotation. For each instruction we build a scene vocabulary from node ids, classes, and descriptions, and keep only query tokens that appear in this vocabulary as target or anchor nodes. The least common ancestor of these nodes forms the oracle context. Spatial phrases such as *near*, *corner*, *left*, and *window* add their referenced objects as anchors. Precision, recall, and hit rate are computed on target objects; oracle recall measures coverage of the oracle context. Seven of the nine cases contain spatial references.

Geometry-aware retrieval variant. We also evaluate a geometry-aware pre-filter that resolves spatial phrases such as *near the window* to candidate objects before embedding retrieval. As shown in table 10, it raises target recall from 25.0% to 49.3% and oracle recall from 14.2% to 29.6%, at the cost of precision ($84.1\% \rightarrow 69.4\%$) and a smaller context reduction ($94.0\% \rightarrow 79.8\%$).

Table 10: Geometry-aware retrieval variant on the same nine cases. The pre-filter resolves spatial phrases before embedding retrieval, trading precision and context size for higher recall. Both variants hit at least one target object in every case.

Variant	Context red. ↑	Prec. ↑	Recall ↑	Oracle ↑
Base HRAG	**94.0%**	**84.1%**	25.0%	14.2%
+ Geo-aware	79.8%	69.4%	**49.3%**	**29.6%**

Chained multi-step benchmark. We write 10 sequences of two to three edits per scene, where later steps refer to objects or groups modified earlier; for example, a sequence can move a bed area and then rearrange the dresser relative to it. Region-coherence drift is the average change in pairwise relative offsets among the leaf objects of the region named by the instruction, so a value of zero means the group's internal arrangement is preserved. The flat-JSON interface receives the same instructions with the scene serialized as a flat object list, and edits are applied by id matching.

A.11 Evaluation Boundaries

CLIP is a proxy for text-scene alignment, so we pair it with human preference judgments and geometry metrics rather than treating it as a standalone score. The 20-participant user study is used as a rendered-scene consistency check for the automatic metrics; broader population-level preference studies are a natural next step once larger repeated generations are available. Baseline reproducibility is handled through explicit reporting boundaries: DirectLayout and LayoutVLM are evaluated on the layout portion supported by their public releases, while HSM is reported on paired prompts because its full pipeline depends on HSSD-HAB support annotations and VLM feedback. We therefore reserve the strongest claims for comparisons with full text-to-scene systems and use recent layout-only baselines as geometry stress tests. For editing, the 0.15m target-change threshold is a strict diagnostic, so we report continuous target displacement, valid-DSL rate, fallback rate, non-target displacement, human compliance, and object removal as separate signals in the analysis.

A.12 FDLO Refinement Stability

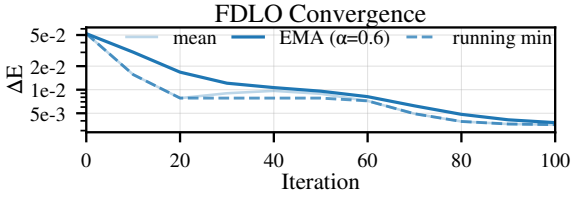


Figure 8: **FDLO convergence.** On a multi-room complex scene, the update magnitude decreases smoothly and plateaus without sustained oscillations.

The plotted value measures the magnitude of each refinement update during iterative optimization. The monotonic decay and later plateau show stable convergence behavior rather than oscillation in challenging, large-scale layouts. The component ablation gives the corresponding final metrics: with direct generation, FDLO reduces OOB from 0.40 to 0.03; with hierarchical generation, it reduces OOB from 0.36 to 0.10 while preserving low PIoU (0.05 after refinement).

A.13 Additional Model Comparison and Ablation

We additionally evaluate recent LLMs under the same generation setting and report average OOB/-PIoU and NObj/CLIP in Table 11.

Metric	Q3	G3	GPT	DS
OOB / PIoU ↓	3.29 / 0.94	3.47 / 0.86	**1.28 / 0.78**	2.47 / 1.04
NObj / CLIP ↑	84.50 / **23.54**	67.50 / 22.81	65.33 / 20.97	**102.17** / 22.99

Table 11: Additional comparison with recent LLMs: Q3 denotes Qwen3-max, G3 denotes Gemini3-Pro, GPT denotes GPT-5.2, and DS denotes DeepSeek-v3.2.

Recent hierarchical baseline. We further compare HDSL with HSM (Pun et al., 2026), a recent hierarchical scene-motif generator. We reproduce two HSM settings on three paired indoor prompts. The first uses the large-object branch, which is the most stable public setting. The second uses the full released pipeline after adding the official HSSD-HAB support-surface package, enabling large, wall, ceiling, and small objects. The full setting completes two prompts and produces a partial state for the home-office prompt after repeated invalid VLM responses during floor support placement; we include this partial state in the geometry evaluation and mark the row accordingly. This comparison is meant as a reproducibility check under matched prompts rather than a full benchmark. As shown in Table 12, HDSL matches the full HSM run in object count while producing fewer boundary violations and collisions with lower runtime.

Method	NObj ↑	OOB ↓	PIoU ↓	Time ↓
HSM-large (Pun et al., 2026)	6.00	0.1698	0.5223	309.0s
HSM-full (Pun et al., 2026)‡	11.67	0.3653	0.8708	654.5s
HDSL (ours)	**11.67**	**0.0000**	**0.0000**	**215.7s**

Table 12: Paired comparison with reproducible HSM settings on three indoor prompts. ‡ The full HSM row includes two completed cases and one partial home-office scene; runtime is averaged over the two completed cases in total.

B Three-Way Merge Details

Naive subtree replacement can fail because retrieved DSLs for editing are partial skeletons: to reduce context length, some contextual nodes may be included as leaves with their descendants pruned. Direct replacement would overwrite full nodes with these pruned versions, causing information loss. Our three-way merge shown in algorithm 1 preserves components present in the Original but missing in the Retrieved/Edited, while applying target edits and resolving identifier conflicts. Attribute conflicts are resolved by the selected target ids: edited attributes are accepted only for selected target subtrees, context-only nodes keep the original attributes, new edited children are accepted only for add/replace intents, and deletion is allowed only when a removed subtree contains a selected target under remove/replace intent. Unchanged objects keep their original retrieved asset metadata intact across all merge steps.

Algorithm 1 Guarded Three-Way Merge

Require: Original tree $\Phi(\mathcal{S})$, unedited tree $\Phi_{\text{rag}}(\mathcal{S})$, edited tree $\Phi'_{\text{rag}}(\mathcal{S})$, selected ids $\mathcal{T}$, edit intent I

Ensure: Merged tree $\Phi_{\text{merged}}(\mathcal{S})$

if $\mathcal{T} \neq \emptyset$ and $\Phi_{\text{rag}}(\mathcal{S})$ contains no id in $\mathcal{T}$ **then**

  return $\Phi(\mathcal{S})$

end if

Initialize Φ_{merged} with edited root attributes if $\mathcal{T} = \emptyset$ or the root id is in $\mathcal{T}$; otherwise use original root attributes.

for each child id u in the union of Original, Retrieved, and Edited children **do**

  if u exists in all three versions **then**

    Add $\text{Merge}(\Phi_u, \Phi_{\text{rag},u}, \Phi'_{\text{rag},u}, \mathcal{T}, I)$.

  else if u exists in Edited and Original but not Retrieved **then**

    Add edited child after resolving identifier conflicts.

  else if u exists only in Edited **then**

    if $\mathcal{T} = \emptyset$ or $I \in \{\text{add}, \text{replace}\}$ **then**

      Add the edited child.

    end if

  else if u exists in Retrieved but not Edited **then**

    if $I \in \{\text{remove}, \text{replace}\}$ and the retrieved subtree of u contains an id in $\mathcal{T}$ **then**

      Skip u as an intended deletion.

    else

      Preserve the Original child if available; otherwise preserve the Retrieved child.

    end if

  else if u exists only in Original **then**

    Preserve the Original child.

  end if

end for

return $\Phi_{\text{merged}}(\mathcal{S})$

Example. When editing a target object (e.g., a bed), a nearby contextual object (e.g., a desk) can appear in the retrieved partial DSL as an empty leaf. If we directly replace the subtree, this pruned contextual node would overwrite the fully-expanded version in the original scene, erasing its children (e.g., a computer). The three-way merge keeps the original desk subtree because the desk is context rather than a selected target, while still applying the bed's edited position and preserving its asset metadata across merge steps.

B.1 Representation Scope

Pure spatial hierarchies are less concise than procedural programs for repetitive patterns. We intentionally adopt a static declarative tree because it is retrieval-friendly: nodes can be indexed and edited directly without executing loops or resolving program state, which is important for localized editing. Control-flow abstractions can be layered on top by external template compilers that emit HDSL, preserving compatibility while keeping the core representation simple and reusable.

C Additional Qualitative Comparisons

For large-room prompts, we raised Holodeck's maximum room-size limit to match our experimental setting. In our I-Design runs, complex layouts sometimes enter repeated layout-modification loops or recursive deadlocks, so we added forced termination and an upper bound on recursion depth. For consistent visual comparison, we also integrated a rendering module into I-Design and selected representative camera viewpoints with matched lighting. Additional HDSL generation examples are shown in fig. 5.

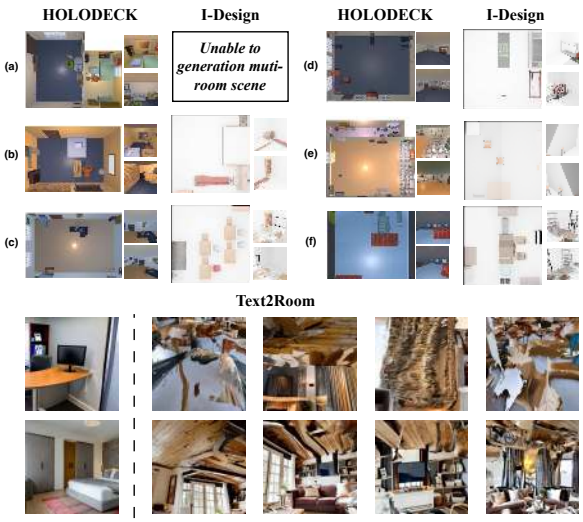


Figure 9: Additional baseline examples in sparse or open layouts. Holodeck tends to place objects near walls, I-Design relies on predefined room geometry and struggles with open-plan or multi-room settings, and Text2Room (Höllein et al., 2023) degrades outside familiar image-generation distributions during rendering.

Figure 10 contrasts the two editing interfaces on the ablation cases: full-scene rewriting can miss a

local edit, work in simple scenes, or become unstable in complex scenes with long context, while HRAG edits the retrieved local subtree.

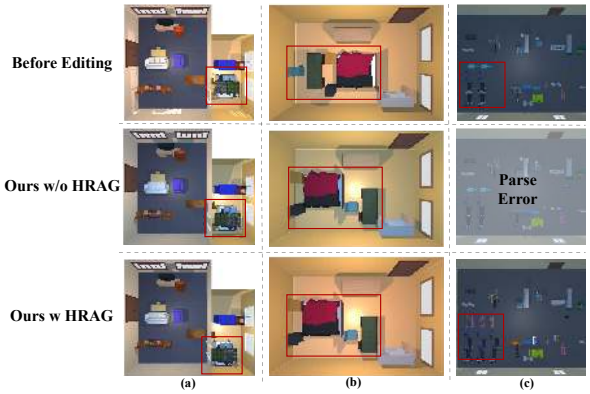


Figure 10: **Editing ablation examples.** Full-scene rewriting can miss a local edit, work in simple scenes, or become unstable in complex scenes with long context, while HRAG edits the retrieved local subtree.

Finer-grained instruction following. We provide an additional case showing that the LLM agent can handle finer-grained constraints in the prompt, including directions, relative ordering, and explicit object-count constraints inside each target scene configuration for testing.

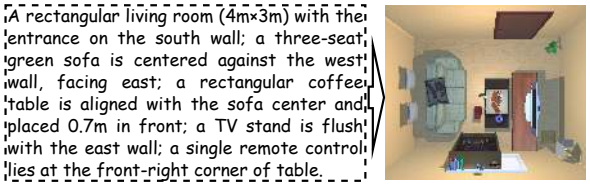


Figure 11: An example of finer-grained instruction following. The generation prompt is shown on the left and the resulting layout is shown on the right.

D User Study Questionnaire

The user study has two parts. For generation realism, 20 participants view three anonymized generated scenes, each shown from four camera views, and choose the most realistic scene. For editing, the same participants compare an original scene with the result of an instruction-based edit and rate compliance on a three-point scale: “exactly matches," “mostly matches," or “does not match." Participants were general users, not members of the method team, and the study did not require professional 3D modeling experience from participants.

For the generation study, we use 5 representative scenes per category to capture structural diversity

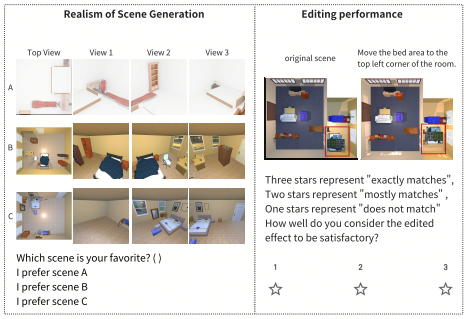


Figure 12: Sample questionnaire, consisting of two parts: generation realism (6 questions) and editing performance ratings (9 questions).

across varying room shapes and sizes while keeping human evaluation costs manageable. The display order of methods is randomized to mitigate bias, and participants do not see method names. We collected 120 votes for the generation evaluation and converted votes into Bradley–Terry preference scores for quantitative comparison. For editing, we collected 180 votes on the three-point compliance scale. The questionnaire does not ask participants to label error type, but the automatic edit logs show that non-exact HRAG cases are mostly small target-position misses after refinement rather than parser failures, object deletions, or metadata loss.

E Prompt Templates

The released implementation uses the templates in `hierarchical_prompts.py`. For readability, we report the executable constraints and template variables here while omitting long few-shot examples.

E.1 Generation Prompt

Input. INPUT_CHAIN, INPUT_LEAF, and the user requirement. The model designs only the direct children of INPUT_LEAF; valid descendants may be nested when they express physical support.
Output. Return exactly one XML-parsable HTML-like DSL fragment. The output must start with <div and end with ; no strategy text, markdown fence, heading, or explanation is allowed.
Required attributes. Every node includes id, description, class, size, variance_type, virtual, is_leaf, and inline style. The top-level node preserves the original INPUT_LEAF attributes.
Core rules. Use virtual=true for zones and support containers, and virtual=false for objects grounded by retrieval. Parent-child nesting denotes support, e.g., desk-to-laptop and counter-to-sink are valid, while table-to-chair and room-to-sink are invalid. Coordinates use the parent frame with 1px = 1cm, and transform: rotate(...deg) controls

top-view orientation.
Spatial constraints. Door spans are hard keep-clear regions and window spans are soft keep-clear regions. Do not emit clearance zones, aisles, corridors, walkable paths, empty areas, or keep-clear regions as DSL nodes. Avoid sibling overlap, keep wall-adjacent furniture facing the open room, and reduce optional large objects in compact bedrooms or kitchens before producing crowded layouts.
Class constraints. Do not create aggregate concrete classes such as small_items, small_cooking_items, accessories, decor, or miscellaneous_items; use concrete classes such as pot, book, lamp, toaster, or coffee_maker.

E.2 Reflection Prompt

Given REJECT_REASON and the former DSL fragment, return a corrected XML-parsable fragment only. The prompt repeats the no-markdown and no-clearance-node rules. If the rejection is caused by overlap, crowding, or too many physical sibling pairs, the model is instructed to remove optional large objects rather than shifting the same crowded set. Compact-bedroom and compact-kitchen cases receive explicit pruning guidance.

E.3 Editing Prompt

Input. The edit instruction, the retrieved HDSL context, and the root identifier of the provided scene.
Output. Return the full edited DSL fragment with the same root id as the provided scene; do not return only the edited child node. The output must be XML-parsable and contain no strategy text, markdown fence, heading, note, or explanation.
Preservation rules. Preserve existing node ids and unrelated sibling subtrees. For existing nodes that are not being replaced, keep class, retrieved_asset_id, retrieved_size, retrieved_description, and other asset metadata unchanged when present; update only the style, coordinates, rotation, or child set needed by the instruction.
Layout rules. Use only tags with inline CSS. Each element includes id, description, class, size, variance_type, virtual, and is_leaf. The same support, generic-class, door/window, overlap, and clearance-node constraints used for generation also apply during editing.

At runtime, the editor appends three short guards to this template: context siblings are preserved unless explicitly targeted, the returned fragment must keep the same root id, and an intent-specific suffix constrains add, remove, replace, rearrange, or modify instructions.