INTERIORAGENT: LLM Agent for Interior Design-Aware 3D Layout Generation

Kunal Gupta[1] Ishit Mehta[1] Kun Wang[1] Nicholas Chua[1] Abhimanyu Krishna[2] Yan Deng[2]
Ravi Ramamoorthi[1] Manmohan Chandraker[1]

[1]University of California, San Diego [2]Qualcomm

{k5gupta, mkchandraker}@ucsd.edu

Figure 1. **(Left)** shows 3D scenes generated by INTERIORAGENT with prompts "a dining room" (top) and "a beauty parlour" (bottom). **(Middle)** shows scenes where INTERIORAGENT deploys new tools, namely, an ASCII generator with the prompt "forest made to look like INTERIORAGENT 3DV 2026 VANCOUVER" (top) and human model placement with the prompt "a living room scene where friends have gathered to watch TV" (bottom). **(Right)** shows an editing application with the prompt "add a larger window and additional decor items".

Abstract

Creating interior layout designs has numerous applications, including virtual reality, architectural visualization and real estate planning. Generating realistic and functional indoor scenes requires a nuanced understanding of spatial configurations and human-centered design principles. We propose INTERIORAGENT *, an LLM-agent-driven framework for text-to-3D indoor scene generation that produces scenes with visual quality and functional utility that significantly surpass prior works. We achieve this through several key advantages of* INTERIORAGENT *: (1) encoding of interior design principles with a novel scene description language, (2) aesthetics and functionality through synthesis tools that satisfy design principles, (3) realism and prompt adherence with optimization tools that ensure ergonomics and iterative constraint satisfaction, (4) extensibility with a framework that allows incorporating even mature, complex tools like diffusion models, LLMs and 3D generation repositories. We evaluate* INTERIORAGENT *through a user study, where participants strongly favor its generated scenes over prior state-of-the-art methods. Additionally, we demonstrate novel applications uniquely enabled by* INTERIORAGENT *, including language-based scene editing and seamless tool integration for new tasks. Code and data will be publicly released.*

1. Introduction

Creating realistic and visually appealing 3D indoor environments has long been a goal in computer vision. Today,

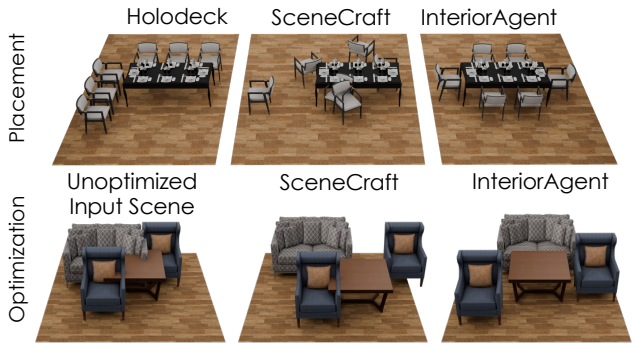


Figure 2. **Toy example illustrating design-aware placement and optimization : (Top)** Comparison of asset placement across methods for a dining setup. Holodeck [41] and SceneCraft [12] rely on coarse proximity constraints and struggle to arrange chairs consistently around the table, whereas INTERIORAGENT leverages interior design–aware synthesis tools to produce a coherent and ergonomic layout. **(Bottom)** Given an initial scene with overlapping furniture, SceneCraft's VLM-based optimization fails to fully resolve collisions. In contrast, INTERIORAGENT employs precise, design-informed gradient-based optimization to produce a collision-free and functionally valid arrangement.

design tools such as Planner5D, TargetHomePlanner, and RoomSketcher [24, 27, 35] are widely used, including by retailers like IKEA [10]. While prior approaches have leveraged LLMs to integrate natural language inputs into these tools for 3D scene generation, they still fall short in terms of visual quality and functional utility. We argue that an effective system is one that is systematically grounded in interior design principles, ensuring superior scene quality in both aesthetics and functionality. Such a system not only enhances realism but also makes interior design tasks more accessible, even for users without specialized expertise. Inspired by established work in interior design [21, 42], we present INTERIORAGENT , a framework for generating structured, visually compelling 3D indoor scenes from natural language prompts. INTERIORAGENT comprises three key components: (1) a program synthesizer that translates prompts into Python-like *scene* programs while integrating specialized tools; (2) an Interior Design–aware Scene Description Language (IDSDL) for authoring these programs; and (3) a suite of design-informed tools that drive both synthesis and optimization, improving ergonomics, realism, and aesthetics.

Scenes generated by INTERIORAGENT are both aesthetically pleasing and functionally sound (Figure 1). Prior LLM-based methods, including Holodeck [41], I-Design, SceneCraft [12], and LayoutVLM [32], struggle to synthesize and optimize interior design layouts. Consider the toy example in Figure 2 of placing six chairs around a rectangular table (top): Holodeck and SceneCraft rely on simple proximity rules, limiting their ability to capture interior design–oriented inter-object relations.

We empirically evaluate the scene generation quality of INTERIORAGENT through a perceptual study, demonstrating its superiority over existing text-to-3D baselines, Holodeck [41], I-Design [2], LayoutVLM [32], and qualitatively against SceneCraft [12] across a range of user prompts. Our analysis shows that INTERIORAGENT produces scenes that adhere more closely to physical realism compared to a state-of-the-art data-driven method, DiffuScene [34]. Further, we highlight INTERIORAGENT 's versatility by integrating tools like Stable Diffusion and full 3D generation repositories like SceneMotifCoder [33] to enhance visual appeal and capability for interactive dialogue-driven iterative scene refinement.

2. Related Works

Classical Methods Early systems like WALKEDIT [1] utilize constraints based on object associations to ascertain layouts. Several methods have leaned on principles of interior design to optimize scene layouts [3, 21, 42]. The INTERIORAGENT SDL incorporates such design principles, but allows intuitive language-based design, while avoiding the hand-crafted constraints and stochastic optimization pitfalls such as local minima faced by classical works [5].

Learning-Based Methods Recent works have used Gaussian mixtures [7, 19], autoregressive transformers [13, 17, 22], recursive autoencoder [14], and graph convolution networks [36, 43] to learn indoor scene layouts, as well as diffusion models [34, 38, 40] for indoor scene synthesis. But data-driven methods face hurdles due to the scarcity of high-quality 3D scene datasets with diverse layouts and annotations. Importantly, once trained, they are not extensible to new assets or design features without expensive retraining.

LLMs for indoor scene synthesis LayoutGPT [6] generates layouts as CSS-style descriptions but struggles with overlaps and poor arrangements due to direct pose regression. Holodeck [41] and I-Design [2] improve on this by prompting GPT-4 to define spatial constraints, resolved with an external optimizer. However, their JSON-like representations limit relations to simple proximity. FlairGPT [16] further incorporates interior design heuristics by prompting an LLM to reason about functional zones, anchor objects, and stylistic intent, followed by translation into executable programs. While effective at capturing high-level design structure, FlairGPT adopts a language-centric planning and translation pipeline, in contrast to approaches that synthesize executable scene programs directly and provide more explicit control over geometry, ergonomics, and extensibility. SceneCraft [12] and LayoutVLM [32] instead rely directly on Vision-Language Models (VLMs), either for iterative optimization or to ascribe proximity-based constraints. Both

approaches lack the depth of interior design reasoning and suffer from imprecise VLM-based 3D understanding. In contrast, INTERIORAGENT introduces a dedicated scene description language with design-informed synthesis and optimization tools, achieving structured, ergonomic, and high-quality layouts. While SceneCraft emphasizes tool learning, INTERIORAGENT focuses on effective tool usage, making their contributions complementary.

Program Synthesis with LLMs INTERIORAGENT leverages tools usage in a program synthesis framework for 3D scene generation. Several LLM-based code generators such as Codex [28], GPT-4 [25], and AlphaCode [15] are used as coding assistants in software development. Tool usage to deploy models and APIs such as HuggingGPT [30] and plug-and-play frameworks such as Chameleon [18] have enhanced LLM adaptability to visual reasoning. Works like Toolformer [29] and Gorilla [23] have explored fine-tuning to improve tool usage for LLMs. VisProg [9] leverages a vision-based API to prompt LLMs to tackle compositional visual tasks, which has been adapted to other domains, such as autonomous driving [39] and robotics [31].

Self-Correcting LLMs Several approaches improve LLM code accuracy: Self-Refine [20] iteratively refines outputs, [11] debugs with print statements, and "rubber duck debugging" [4] detects errors from execution results. Inspired by these, INTERIORAGENT employs a self-correcting mechanism that prompts an LLM to fix syntax errors, API misuse, and scene violations using feedback from execution tracebacks and scene constraints.

3. Interior Design aware Scene Generation

INTERIORAGENT generates 3D scenes from natural language by synthesizing Python programs in a custom Interior Design–aware Scene Description Language (IDSDL). IDSDL embeds design principles and integrates with AI tools and expert modules via a structured I/O framework. Unlike typical LLM-based 3D generation that relies on function calls or tool use, INTERIORAGENT emphasizes design fidelity and extensibility through a robust tools and API stack. As shown in Figure 3, given a prompt, PROGRAMSYNTHESIZER generates a scene program, evaluates embedded VLM-based constraints, and iteratively refines the program until all constraints are satisfied.

3.1. Scene Description Language

A procedural language for indoor layout generation must support core functions such as registering objects, positioning them within scene, and optimizing constraints to enhance functionality and realism. IDSDL excels in all three areas, achieving higher scene fidelity than state-of-the-art methods by strongly integrating interior design principles with a structured framework that accommodates advanced tools, including LLMs, diffusion models, and 3D generation repositories. We now discuss each of these functions in detail.

Object Registration. IDSDL offers a unified API for object retrieval from natural language descriptions, returning appropriately scaled instances suited to indoor settings. Unlike prior work, it supports diverse sourcing methods, including retrieval from multiple 3D asset datasets (e.g., 3DFront, HSSD, Objaverse), generation via text-to-3D models (e.g., Hunyuan3D), and domain-specific tools such as Stable Diffusion (for paintings) and SceneMotifCoder (for stacked objects). Figure 3 lists all retrievers currently implemented in IDSDL with more details in supplementary.

Each retrieval tool integrates cleanly by implementing a Python class (Figure 15), which must: (1) define a name, usage description (e.g., 3DFront provides high-quality furniture, Objaverse offers greater diversity), and examples; and (2) implement `__call__()` to return an object path and estimated width from a prompt. Beyond retrieving object primitives, IDSDL provides rich functions for registered objects such as local coordinate frame, ray intersections, etc (Figure 4), supporting layout generation and optimization.

Object Placement. Objects are typically placed by first assigning approximate positions via a planner (e.g., a VLM) to form an initial layout, then refining through constraint optimization to satisfy spatial and functional requirements. While the toy example in Figure 2 already shows improved placement and optimization over prior work, a more powerful capability stems from *functional groups*, borrowed from interior design, which cluster related objects to support specific activities (e.g., seating areas or workspaces).

For instance, the prompt *"Place two tables, each with two chairs, next to each other"* requires both chair–table and table–table constraints (Figure 5). LayoutVLM [32], which enforces all constraints simultaneously (left), introduces trade-offs between relations. For illustration, we adapt LayoutVLM into a hierarchical variant that first arranges chairs around each table, then places the two table–chair groups side by side (middle), yielding a more coherent layout and evidencing the value of group-level optimization. IDSDL is designed around this principle: by registering, placing, and optimizing within groups, it enables hierarchical reasoning that produces layouts faithful to functional intent (right) while avoiding trade-offs inherent in global placement.

While prior methods have acknowledged functional groups or "zones," they have largely treated them as chain-of-thought prompts to guide layouts. They neither capture the diversity of group types nor optimize them in isolation,

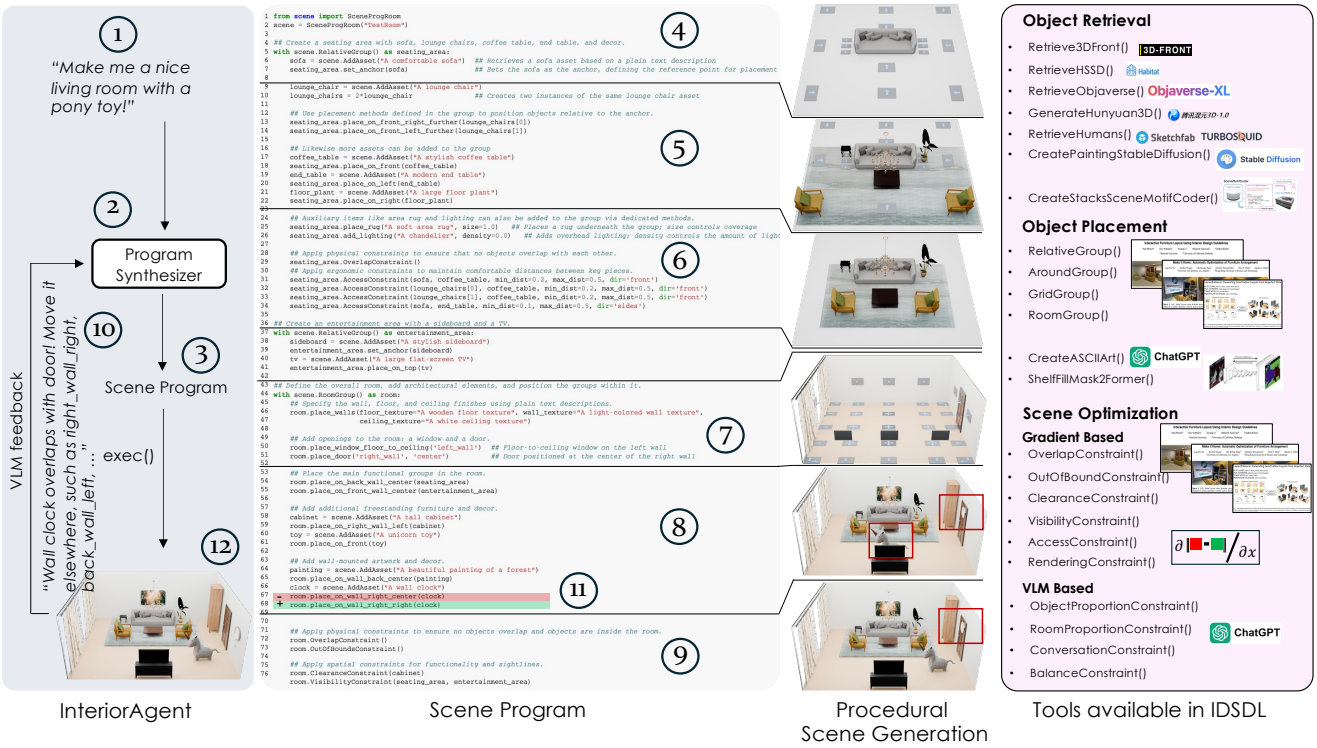


Figure 3. **InteriorAgent pipeline.** From a (1) natural language prompt, (2) ProgramSynthesizer generates an executable scene program in IDSDL (3). The scene is constructed through functional groups: (4) a `RelativeGroup` defines a seating area anchored on the sofa, enabling (5) placement of lounge chairs, a coffee table, and related assets. Objects are optimized at the group level (6), yielding layouts that are physically plausible and functionally coherent. (7) A `RoomGroup` manages walls, windows, and other wall assets, while also positioning functional groups at candidate locations (8). Room-level constraints (9) enforce non-overlap, adherence to room dimensions, and functional requirements such as maintaining TV visibility from the seating area—for example, the unicorn is shifted to preserve sightlines (red box). Execution of the program produces both the scene and (10) VLM feedback, which is passed back to ProgramSynthesizer for refinement. Program edits (11), such as relocating the wall clock to resolve its overlap with the door (red box), yield the final optimized scene (12). The InteriorAgent framework is enriched by a broad set of IDSDL tools, including asset retrievers, interior design–inspired placement and optimization routines, as well as AI-enabled modules such as LLMs, diffusion models, and 3D generation repositories.

Figure 4. Functionalities available to both IDSDL objects and groups for versatile scene generation.

Figure 5. **Toy Example (Functional Groups):** Illustration of hierarchical object placement via functional groups. **(Left)** Directly placing four chairs around two tables is challenging for VLM-based methods such as LayoutVLM [32], which struggle to satisfy all table–chair relations simultaneously. **(Middle)** Decomposing the task hierarchically—first arranging chairs relative to each table, then placing the table–chair groups—improves results, highlighting the value of group-level reasoning. **(Right)** InteriorAgent natively supports functional groups through its scene description language, enabling hierarchical placement that yields coherent and functionally consistent layouts.

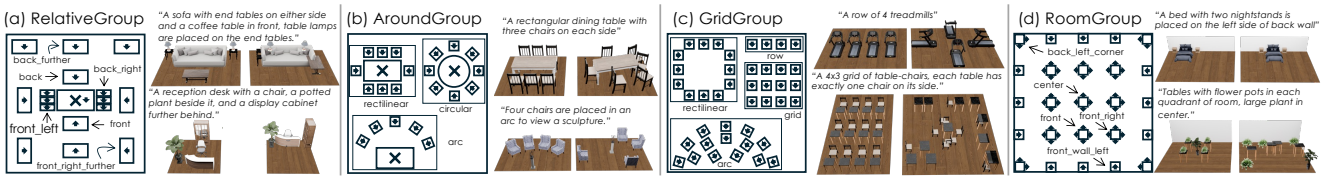


Figure 6. **Object placement.** Groups provide an intuitive mechanism for arranging assets based on interior design patterns. Boxes indicate candidate positions within a group, while arrows mark object orientations. Distances and orientations are fixed to support ergonomic factors such as accessibility, visibility, conversation, and circulation. For each group, example scenes generated with IDSDL (left) better align with the prompt (see supplementary for corresponding scene programs), whereas LayoutVLM (right) struggles due to limited placement capacity. This underscores the value of interior design–aware placement in improving both visual quality and functional utility.

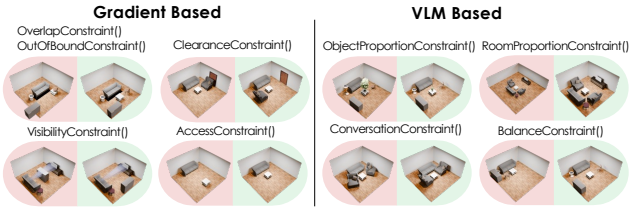


Figure 7. Visualization of constraints available in IDSDL.

thereby missing the advantages of hierarchical reasoning. In contrast, IDSDL introduces a rich set of group APIs inspired by interior design patterns (Figure 6). `RelativeGroup()` arranges objects relative to an *anchor object*—a focal element that sets the tone for the group—across 17 predefined positions based on anchor's local coordinate frame: eight adjacent (three per side, plus front and back), eight at greater distances, and one on top. Orientations are fixed to promote ergonomic layouts, such as conversational seating, while object-object distances, especially for adjacent placements, are chosen to ensure functional access. `AroundGroup()` places objects around an anchor in rectilinear, circular, or arc configurations, while `GridGroup()` supports repetitive patterns in rows, grids, or arcs. At the highest level, `RoomGroup()` organizes all groups and manages global elements such as windows, doors, and wall-mounted objects.

Each group is implemented as a Python class (Figure 16) that specifies: (1) a group name; (2) a placement routine, decorated with `@placemethod`, which includes a description, example usage, and pose-computation logic; and (3) an optional `compile()` method for fine-grained control over placement and optimization order. Groups can then be invoked directly by INTERIORAGENT using their descriptions and examples.

Scene Optimization. Our framework supports joint use of gradient-based optimization—effective for continuous constraints such as overlap minimization—and VLM-based optimization—suited to qualitative constraints like aesthetics—yielding superior visual appeal and functionality. See Figure 7 for visualization of various constraints. Gradient-based constraints `Overlap()` and `OutOfBound()` prevent object–object intersections and placement outside the room bounds. `Clearance()` reserves necessary space for accessibility (e.g., beside beds, in front of cabinets), while `Access()` ensures key items such as coffee tables remain within reach from seating. `Visible()` maintains clear sightlines, for example between a TV and seating.

Each constraint provides a *pseudo-gradient* for its objects, guiding updates that minimize violations. IDSDL accumulates these gradients and optimizes at the end of placement. To balance constraint satisfaction with minimal displacement, updates are applied selectively, sampled from a distribution that considers object size and surrounding free space, thus preserving group structure.

Consider a group $\mathcal{G}$, with ∂o denoting the total gradient from all constraints on each object $o \in \mathcal{G}$. Let FreeSpace : $\mathcal{G} \times \mathcal{D} \to \mathbb{R}$ measure the available space around o along directions $\mathcal{D} = [\hat{x}+, \hat{x}-, \hat{z}+, \hat{z}-]$, defined as the distance to the nearest object or wall. At optimization step s, the unnormalized action density $\Gamma^s_{|\mathcal{G}|\times 4} = [\tau^s_i]$ is

$$\tau^s_{i,j} = \max\left(\frac{j \cdot \partial o^s_i}{\Omega(o^s_i)} \cdot \text{FreeSpace}(o^s_i, j), 0\right) \tag{1}$$

favoring objects with smaller footprint $\Omega(o^s_i)$ and directions j aligned with ∂o^s_i that have greater free space, thereby reducing collision risk. A small subset of updates $U^s = \{(i,j) \mid (i,j) \sim \text{SoftMax}(\Gamma^s)\}$ with $|U^s| \ll |\mathcal{G}|$ is then sampled, and each selected object o_i moves along j:

$$o^{s+1}_{i,j} \leftarrow o^s_{i,j} + \eta \cdot \partial o^s_{i,j}, \tag{2}$$

where η is the learning rate. The process runs for S steps per group, yielding one optimization pass per execution.

VLM-based constraints use group renderings and numerical context to suggest edits to the scene program. `ObjectProportion()` enforces realistic relative object sizes within a group; `RoomProportion()` ensures room dimensions accommodate placed furniture; `Conversation()` orients seating to support interaction; and `Balance()` promotes even spatial distribution of objects.

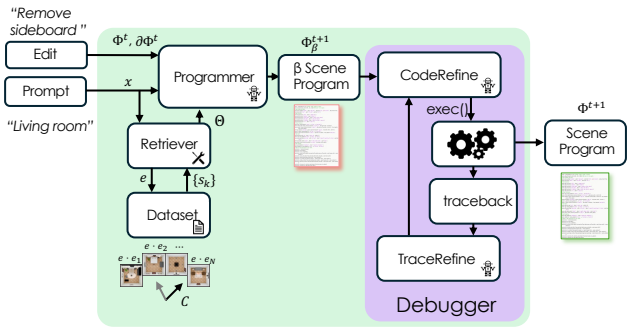


Figure 8. PROGRAMSYNTHESIZER takes an input prompt (or an optional edit prompt) and generates a valid scene program that, when executed, produces a 3D scene. This is done by prompting `Programmer` with the input descriptions and relevant in-context examples provided by the `Retriever` for reference. The `Programmer` outputs an initial β scene program, which may contain syntax errors; these are resolved by `Debugger` , resulting in an executable scene program. `Debugger` performs this refinement through an iterative loop between `CodeRefine` and `TraceRefine` , optimizing based on the scene program and any traceback information from its execution.

When executed, IDSDL consolidates *VLM feedback* and passes it to PROGRAMSYNTHESIZER to refine the scene program. At iteration t, the program Φ^t and feedback $\partial\Phi^t$ yield an updated program:

$$\Phi^{t+1} \leftarrow \text{PROGRAMSYNTHESIZER}\left(x|\Phi^t, \partial\Phi^t\right) \quad (3)$$

where x is the input prompt. Constraints are implemented as Python classes (Figure 17) specifying metadata (name, usage, examples, type, and optional VLM prompt) and a `compute_gradients()` method for pseudo-gradients computation and generating response from VLM.

3.2. Program Synthesizer

Given an input prompt and optional edit instructions, `Programmer` produces an initial β scene program, which may contain syntax errors. To improve accuracy, the `Retriever` provides relevant in-context examples that serve as references for `Programmer` . Any remaining errors in β are resolved by the `Debugger` , yielding the refined scene program (Figure 8). All LLM-based modules (`Programmer` , `Retriever`, `CodeRefine` , `TraceRefine`) use the same backbone VLM (GPT-4o), differing only in system prompts.

Programming. The `Programmer` is an LLM prompted to write scene programs. It is provided with IDSDL documentation along with up to K in-context examples Θ related to the input query. Additionally, `Programmer` can be used in "edit" mode by providing a previous scene program Φ^t and $\partial\Phi^t$, containing edit instructions, as an additional prompt. Given a prompt composed from these inputs, the `Programmer` VLM outputs a scene program Φ_β^{t+1}

$$\Phi_\beta^{t+1} = \texttt{Programmer}\left(x, \Theta|\Phi^t, \partial\Phi^t\right) \quad (4)$$

We refer to this scene program as a β scene program due to its tendency to have syntax errors. This program is subsequently debugged by the `Debugger` . A program based scene representation provides fine-grained control over object placement and relationships, enables iterative scene refinement through program modifications, and extends functionality via modern programming constructs and external libraries. In contrast, prior methods [2, 6, 8, 41] rely on less expressive scene representations like JSON and lists. Moreover, our ability to apply language-domain edits to a scene program enables language-based scene editing capabilities as shown in Section 4.

In-Context Examples. We curate a dataset of N instructive scene programs, each demonstrating how to use the IDSDL for a variety of scene layouts (a few examples are shown in supplementary). Each example comprises a brief layout description in plain English, λ_l, $0 \leq l \leq N$, and the corresponding scene program Φ_l. For each example, a 1D vector encoding $e_l = \mathcal{E}(\lambda_l)$ is generated using a text embedding model $\mathcal{E}$. Similar embedding is computed for a user's input $e = \mathcal{E}(x)$. We then calculate the cosine similarities between e and each embedding e_l in the database, yielding the similarity set $C(\lambda) = \{e \cdot e_1, e \cdot e_2, \cdots, e \cdot e_N\}$. The set of relevant in-context examples $\Theta = \{s_k\}$, $k \in \arg\text{top}_K(C)$ is thus passed to the `Programmer` which significantly enhanced the correctness of generated scene programs. In-context examples enhance VLM-driven scene generation by incorporating expert knowledge through Retrieval-Augmented Generation (RAG), unlike previous VLM-based 3D scene generation methods that rely solely on the VLM, often producing weaker results.

Debugging. The initial scene program, Φ_β^{t+1}, is reviewed by the `CodeRefine` tool, a VLM prompted to catch syntax errors and basic issues, producing an error-free version. Any remaining errors are identified by executing the program and capturing tracebacks, which are then refined by the `TraceRefine` tool, another VLM prompted to addresses errors indicated in the traceback. Through a few iterations, this process yields a functional scene program, Φ^{t+1}:

$$\Phi^{t+1} = \texttt{Debugger}(\Phi_\beta^{t+1}) \quad (5)$$

Using `CodeRefine` upfront for syntax fixes speeds up debugging, reducing execution cycles. We note `Debugger` is crucial for INTERIORAGENT to write accurate code given its complex tool-based IDSDL. Notably, prior program-synthesis-based 3D generation methods, such as [12], lack this functionality.

	Q1	Q2	Q3	Q4	Q5	Q6
Nearest E.g.	41%	42%	43%	65%	77%	76%
I-Design	65%	70%	79%	72%	71%	70%
Holodeck	71%	69%	70%	70%	67%	67%
LayoutVLM	70%	75%	73%	69%	63%	63%

Table 1. User preference for INTERIORAGENT -generated scenes over Nearest Example, I-Design, Holodeck, and LayoutVLM across Q1–Q6 (prompt adherence, ↑).

Nearest E.g.	I-Design	Holodeck	LayoutVLM	InteriorAgent
1.7	2.2	2.4	2.5	**3.4**

Table 2. Adherence of generated scenes to input prompts on a 5-point scale for different methods (↑).

Categories	OB	OO	Vis.	Cler.	Acc.	Conv.
w/o opt.	9%	22%	10%	27%	9%	7%
w/ opt.	**1%**	**2%**	**2%**	**8%**	**0%**	**0%**

Table 3. Impact of optimization on constraint satisfaction.

4. Experiments

We evaluate INTERIORAGENT by generating scenes from various inputs and comparing them to four state-of-the-art methods: Holodeck [41], I-Design [2], SceneCraft [12], and LayoutVLM [32]. For Holodeck, I-Design, and LayoutVLM, we use publicly available code and conduct a perceptual study. Since SceneCraft's code was unavailable at submission, we re-implemented it based on the original paper and report only qualitative results, incorporating its spatial skill library without retraining. Ablations are evaluated both qualitatively and quantitatively. Implementation details and additional experiments are provided in the supplementary.

Evaluation Dataset. We construct the **MIT Scenes** dataset to evaluate different models, derived from [26]. It covers 53 diverse indoor categories, each with three detailed prompts, for a total of 159 prompts.

4.1. Perceptual Study

We conduct a perceptual study comparing INTERIORAGENT with Holodeck, I-Design, and LayoutVLM, involving 50 undergraduate and graduate participants. Each completed a two-alternative forced-choice test across 50 top-down images and 360° videos of generated scenes. Participants answered: **Q1:** "In which scene are the room and objects sized more appropriately?", **Q2:** "In which scene are the objects arranged more naturally?", **Q3:** "Overall, which scene is more aesthetically pleasing?", **Q4:** "Overall, which scene is better?", **Q5:** "In which scene are the objects arranged more in accordance with the caption/prompt?", and **Q6:** "In which scene are the object–object relations more in accordance with the caption?". Participants also provided absolute ratings (Table 2). Quantitative results are summarized in Table 1, and qualitative comparisons across scene categories are shown in Figure 9. Users strongly preferred INTERIORAGENT , consistently rating its scenes as more aesthetically pleasing and overall superior to those from Holodeck, I-Design, and LayoutVLM. For Q1–Q3, nearest-neighbor scenes were often favored due to being human-curated; however, INTERIORAGENT still outperformed all other baselines. For Q5–Q6 and absolute ratings (Table 2), INTERIORAGENT was judged to adhere more closely to input prompts and, on Q4, to be the best overall.

4.2. Ablations

Figure 10 shows that all components of INTERIORAGENT are critical to achieving superior results. We quantitatively evaluate the impact of gradient- and VLM-based optimization by generating MIT Scenes prompts with and without optimization, measuring constraint violations for out-of-bounds (OB), object overlaps (OO), visibility (Vis.), clearance (Cler.), accessibility (Acc.), and conversation (Conv.). Gradient-based optimization primarily addresses OB, OO, Vis., Cler., and Acc., while VLM-based optimization enforces Conv. Table 3 reports a substantial reduction in violations with feedback enabled, underscoring its effectiveness in resolving these issues.

4.3. Applications

Scene editing. INTERIORAGENT 's chat-like interface allows users to generate and iteratively edit scenes, accommodating complex requirements beyond a single prompt. Figure, in supplementary, shows a dining room scene refined through prompts for object placement and aesthetic adjustments. Unlike Holodeck, LayoutVLM and I-Design, which generate scenes in a single feedforward pass, INTERIORAGENT allows user-specified customization with physical and aesthetic coherence.

Tool Integration INTERIORAGENT leverages template-specified context to integrate external tools. We demonstrate this with (i) human model placement in standing/sleeping poses, (ii) Stable Diffusion for paintings matching scene style, and (iii) `ASCIIGenerator()` for object arrangements forming words (Figure 1, top middle). Such extensibility is uniquely enabled by INTERIORAGENT 's program synthesis, beyond prior methods. For figures and additional tool integration, see supplementary.

5. Conclusion and Future Work

The LLM-based program synthesis approach of INTERIORAGENT enables features such as tool usage, self-correction, refinement and an expressive scene language, leading to realistic, ergonomic, editable and extensible 3D indoor scene generation. We envision broad use of INTERIORAGENT 's

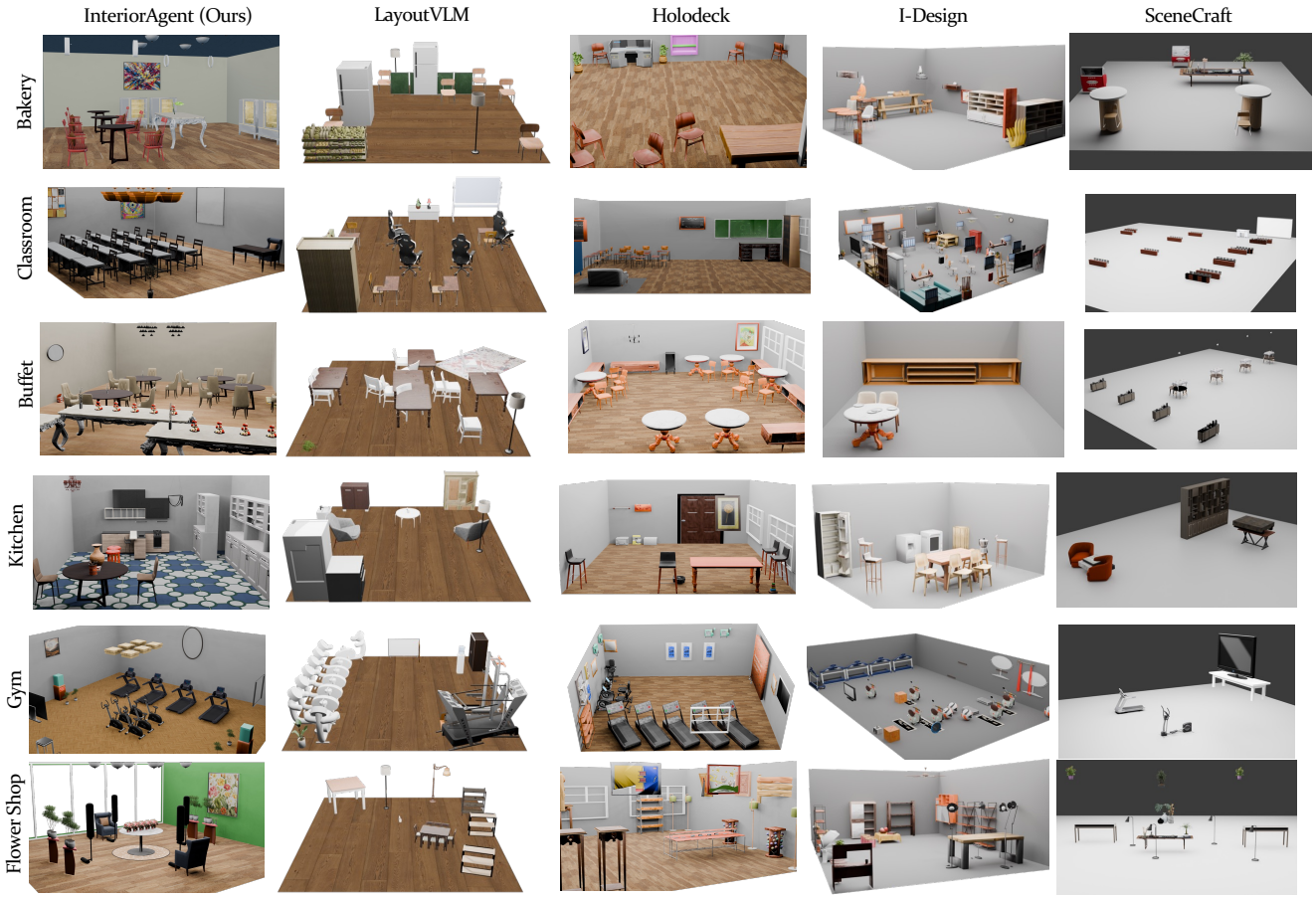


Figure 9. **MIT Scenes:** INTERIORAGENT generates scenes that closely follow input prompts. Kitchen: all baselines omit the island. Classroom: SceneCraft places desks and chairs too sparsely, while LayoutVLM, Holodeck, and I-Design yield chaotic arrangements. Gym: stationary bikes are not arranged in a grid. Bakery, Buffet, and Flower Shop: INTERIORAGENT produces more natural and coherent layouts.

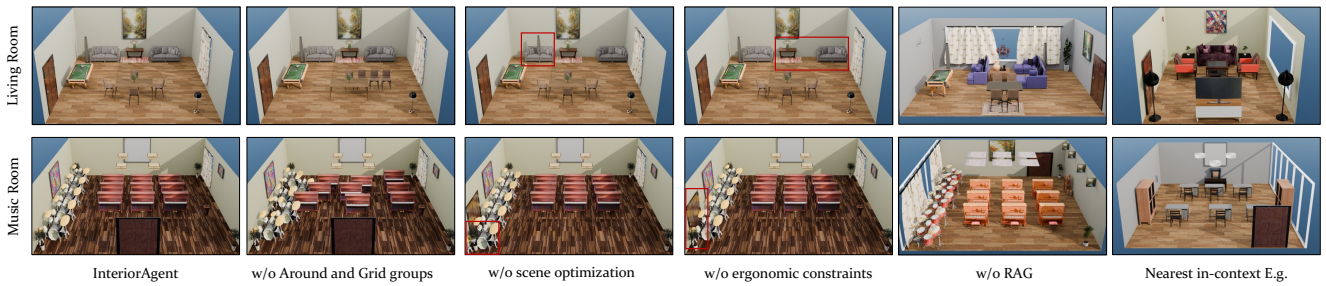


Figure 10. **Ablations :** Without the placement mechanisms of `Around` (top) and `Grid` (bottom) groups, VLMs misplace assets due to limited 3D spatial understanding. Disabling scene optimization (all gradient and VLM constraints off) reduces realism, producing overlaps (red highlights). Removing ergonomic constraints (access, clearance, visibility, conversation) compromises functionality, leading to accessibility issues. Providing all in-context examples to the LLM does not improve scene quality but increases token usage by $8\times$ compared to RAG with 5 examples. Despite differing from nearest neighbors, INTERIORAGENT generates diverse, high-quality 3D scenes with superior prompt adherence.

capabilities through its intuitive editing and flexible tool integration for new tasks. Promising directions for future work include extension to multi-room scenes and optimization of material and lighting, in addition to 3D layouts and objects.

References

[1] Richard W Bukowski and Carlo H Séquin. Object associations: a simple and practical approach to virtual 3d manipulation. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 131–ff, 1995. 2

[2] Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. I-design: Personalized llm interior designer. *arXiv preprint arXiv:2404.02838*, 2024. 2, 6, 7, 1

[3] Angel Chang, Manolis Savva, and Christopher D Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 2028–2038, 2014. 2

[4] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023. 3

[5] Bob Coyne and Richard Sproat. Wordseye: An automatic text-to-scene conversion system. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 487–496, 2001. 2

[6] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *arXiv preprint arXiv:2305.15393*, 2023. 2, 6

[7] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG)*, 31(6):1–11, 2012. 2

[8] Rao Fu, Zehao Wen, Zichen Liu, and Srinath Sridhar. Anyhome: Open-vocabulary generation of structured and textured 3d homes. In *European Conference on Computer Vision*, pages 52–70. Springer, 2025. 6

[9] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. 3

[10] Jordan Hobbs. Why IKEA Uses 3D Renders vs. Photography for Their Furniture Catalog. https://www.cadcrowd.com/blog/why-ikea-uses-3d-renders-vs-photography-for-their-furniture-catalog/, 2024. Accessed: 2024-01-19. 2

[11] Xueyu Hu, Kun Kuang, Jiankai Sun, Hongxia Yang, and Fei Wu. Leveraging print debugging to improve code generation in large language models. *arXiv preprint arXiv:2401.05319*, 2024. 3

[12] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scene as blender code. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. 2, 6, 7

[13] Kurt Leimer, Paul Guerrero, Tomer Weiss, and Przemyslaw Musialski. Layoutenhancer: Generating good indoor layouts from imperfect data. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–8, 2022. 2

[14] Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics (TOG)*, 38(2):1–16, 2019. 2

[15] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. 3

[16] Gabrielle Littlefair, Niladri Shekhar Dutt, and Niloy J. Mitra. Flairgpt: Repurposing llms for interior designs. *Computer Graphics Forum*, 44(2):e70036, 2025. 2, 1, 4, 5

[17] Jingyu Liu, Wenhan Xiong, Ian Jones, Yixin Nie, Anchit Gupta, and Barlas Oğuz. Clip-layout: Style-consistent indoor scene synthesis with semantic furniture embedding. *arXiv preprint arXiv:2303.03565*, 2023. 2

[18] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[19] Rui Ma, Akshay Gadi Patil, Matthew Fisher, Manyi Li, Sören Pirk, Binh-Son Hua, Sai-Kit Yeung, Xin Tong, Leonidas Guibas, and Hao Zhang. Language-driven synthesis of 3d scenes from scene databases. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018. 2

[20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[21] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM transactions on graphics (TOG)*, 30 (4):1–10, 2011. 2

[22] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in Neural Information Processing Systems*, 34:12013–12026, 2021. 2

[23] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. 3

[24] Planner5d. Planner5d: House Design Software. https://planner5d.com, 2024. Accessed: 2024-01-19. 2

[25] Russell A Poldrack, Thomas Lu, and Gašper Beguš. Ai-assisted coding: Experiments with gpt-4. *arXiv preprint arXiv:2304.13187*, 2023. 3

[26] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE conference on computer vision and pattern recognition*, pages 413–420. IEEE, 2009. 7

[27] RoomSketcher. Create Floor Plans and Home Designs Online. http://www.roomsketcher.com, 2024. Accessed: 2024-01-19. 2

[28] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code

explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43, 2022. 3

[29] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[30] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[31] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: program generation for situated robot task planning using large language models. *Autonomous Robots*, 47(8):999–1012, 2023. 3

[32] Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29469–29478, 2025. 2, 3, 4, 7, 1, 5

[33] Hou In Ivan Tam, Hou In Derek Pun, Austin T. Wang, Angel X. Chang, and Manolis Savva. SceneMotifCoder: Example-driven Visual Program Learning for Generating 3D Object Arrangements. 2024. 2, 5, 8

[34] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Scene graph denoising diffusion probabilistic model for generative indoor scene synthesis. *arXiv preprint arXiv:2303.14207*, 2023. 2

[35] Target. Home planner. https://www.target.com/room-planner/home, 2024. Accessed: 2024-01-19. 2

[36] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019. 2

[37] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*, 2024. 1

[38] Qiuhong Anna Wei, Sijie Ding, Jeong Joon Park, Rahul Sajnani, Adrien Poulenard, Srinath Sridhar, and Leonidas Guibas. Lego-net: Learning regular rearrangements of objects in rooms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19037–19047, 2023. 2

[39] Dongming Wu, Wencheng Han, Tiancai Wang, Yingfei Liu, Xiangyu Zhang, and Jianbing Shen. Language prompt for autonomous driving. *arXiv preprint arXiv:2309.04379*, 2023. 3

[40] Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. Physcene: Physically interactable 3d scene synthesis for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16262–16272, 2024. 2

[41] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, pages 20–25. IEEE/CVF, 2024. 2, 6, 7, 1

[42] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30,(4), July 2011, article no. 86*, 30(4), 2011. 2

[43] Guangyao Zhai, Evin Pinar Örnek, Shun-Cheng Wu, Yan Di, Federico Tombari, Nassir Navab, and Benjamin Busam. Commonscenes: Generating commonsense 3d indoor scenes with scene graphs. *arXiv preprint arXiv:2305.16283*, 2023. 2

INTERIORAGENT: LLM Agent for Interior Design-Aware 3D Layout Generation

Supplementary Material

6. Acknowledgments

This work was supported in part by NSF grant IIS 2402583, a Qualcomm Innovation Fellowship, ONR grant N00014-23-1-2526, gifts from Adobe, Rembrand, and the Ronald L. Graham Chair.

7. Reproducibility

All code, data, prompts and experimental details will be publicly released.

Please see the attached `constraints.py` file in the folder for reference.

8. Full prompts for Visualizations

MIT Scenes

Bakery := *"The bakery's layout includes refrigerated display cases along the left and back wall. In the center, a long wooden table displays freshly baked loaves and pastries. A few tables and chairs are placed in the front for customers."*

Kitchen := *"The kitchen features a large island in the center of the back wall with a marble countertop. Along one wall, there is a row of cabinets and there is a coffee table with two armchairs in the corner."*

Classroom := *"The classroom features rows of individual desks with attached chairs arranged in neat columns. A large whiteboard spans the front wall, with a teacher's desk placed in one corner."*

Gym := *"The center of the gym has several stationary bikes arranged in a grid pattern with a TV mounted on their front wall. Behind them, a row of treadmills are placed along the wall."*

Buffet := *"A buffet scene with a row of tables placed along the left, each having a variety of food items placed on top. In the center, four round tables are placed for seating"*

Flower Shop := *"The flower shop features a central display table with flower bouquets on top, and floor lamps illuminating the arrangements. Along the walls, there are additional tables with potted plants"*

9. MIT Scenes Dataset

Comparison to LayoutVLM[32] and Holodeck[41] Please see Figs. 11, 12 for more illustrations of scenes generated via INTERIORAGENT , LayoutVLM[32] and Holodeck[41]. We note the ability of INTERIORAGENT to more accurately follow descriptive prompts while maintaining better aesthetic quality. We observe that INTERIORAGENT is able to utilize tools in its toolkit to generate a diverse set of realistic scenes that are not only more visually pleasing, but also respect human-centric factors for better interior design. Further, we note that INTERIORAGENT is able to effectively use both 3D-FRONT as well as other datasets such as Objaverse to meet its asset requirements. This is a key advantage over previous methods which do not have a *tool* based approach and therefore, are not easily extensible beyond a fixed dataset.

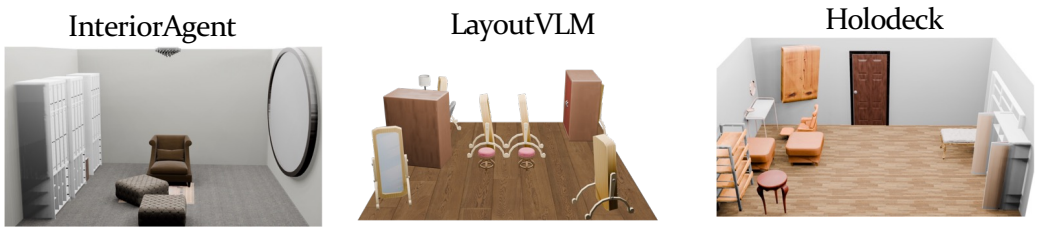
Comparison to FlairGPT[16] We compare INTERIORAGENT with FlairGPT [16], another state-of-the-art method for indoor scene generation also inspired by interior design principles. FlairGPT consists of three stages: a *Language Phase*, where an LLM is prompted via chain-of-thought to generate an initial layout by identifying zones, adding anchor and secondary assets, and specifying constraints; a *Translation Phase*, which converts these natural-language layouts and constraints into programs; and an *Optimization Phase*, which executes the program to produce the final scene.

Despite these apparent similarities, INTERIORAGENT differs in several key ways. First, it adopts a direct program synthesis approach to specify layouts and constraints, avoiding the limitations of natural language representations used in FlairGPT, Holodeck [41], I-Design [2], and LayoutGPT [32]. Prior work shows program synthesis provides stronger planning capabilities through programming structures and external tools [37]. This is reflected in reliability: FlairGPT fails to translate to an executable program roughly 25% of the time (tested on 20 MIT Scenes prompts, issue also noted in official code release), whereas INTERIORAGENT , supported by code debuggers, achieves a near 99% execution rate. Runtime also differs substantially: FlairGPT requires 20 minutes per scene versus 3 minutes for INTERIORAGENT .

Second, while both methods reference functional zones, INTERIORAGENT formalizes them via dedicated group *templates* that support diverse zone types with isolated object registration, placement, and optimization. This hierarchical design more faithfully captures functional groupings than FlairGPT's chain-of-thought prompting.

Third, IDSDL equips INTERIORAGENT with a richer suite of placement and optimization tools, including constraints unavailable in FlairGPT such as Clearance, Visibility, and VLM-based constraints (ObjectProportions, RoomProportion, Conversation, Balance). These contribute to more

Figure 11. Example renders of scenes generated with prompts from six MIT Scenes categories.



"The walk-in closet has an entire wall covered in mirror panels. The opposite wall features wardrobes. In the center, there are two ottomans and an armchair. It also has overhead lighting to illuminate the space."

"The computer room features a row of computer desks with desktop monitors along one wall. A large whiteboard is mounted on the opposite wall, and a large table is placed in the center with stools around it."

"The corridor features a door each along the left and right walls. A runner rug lines the center of the floor, several paintings are placed the front which are visible to couches placed in the back."

"The deli features several refrigerated units along the back wall, with a row of long glass display counter filled with fresh sandwiches and salads, positioned at some distance in front of them. A row of bar stools lines a counter along the wall with a huge window for casual seating."

"The dental office features a reception desk, a waiting area with a few chairs in the middle. Along the back wall, there are three dental chairs with a tray of instruments placed next to each. There are plants, floor lamps placed near the reception desk."

"The museum hall features a central exhibit showcasing a large dragon on a raised platform. Spotlights from the ceiling illuminate the artifact, and benches are positioned along the walls for visitors."

Figure 12. Example renders of scenes generated with prompts from six MIT Scenes categories.

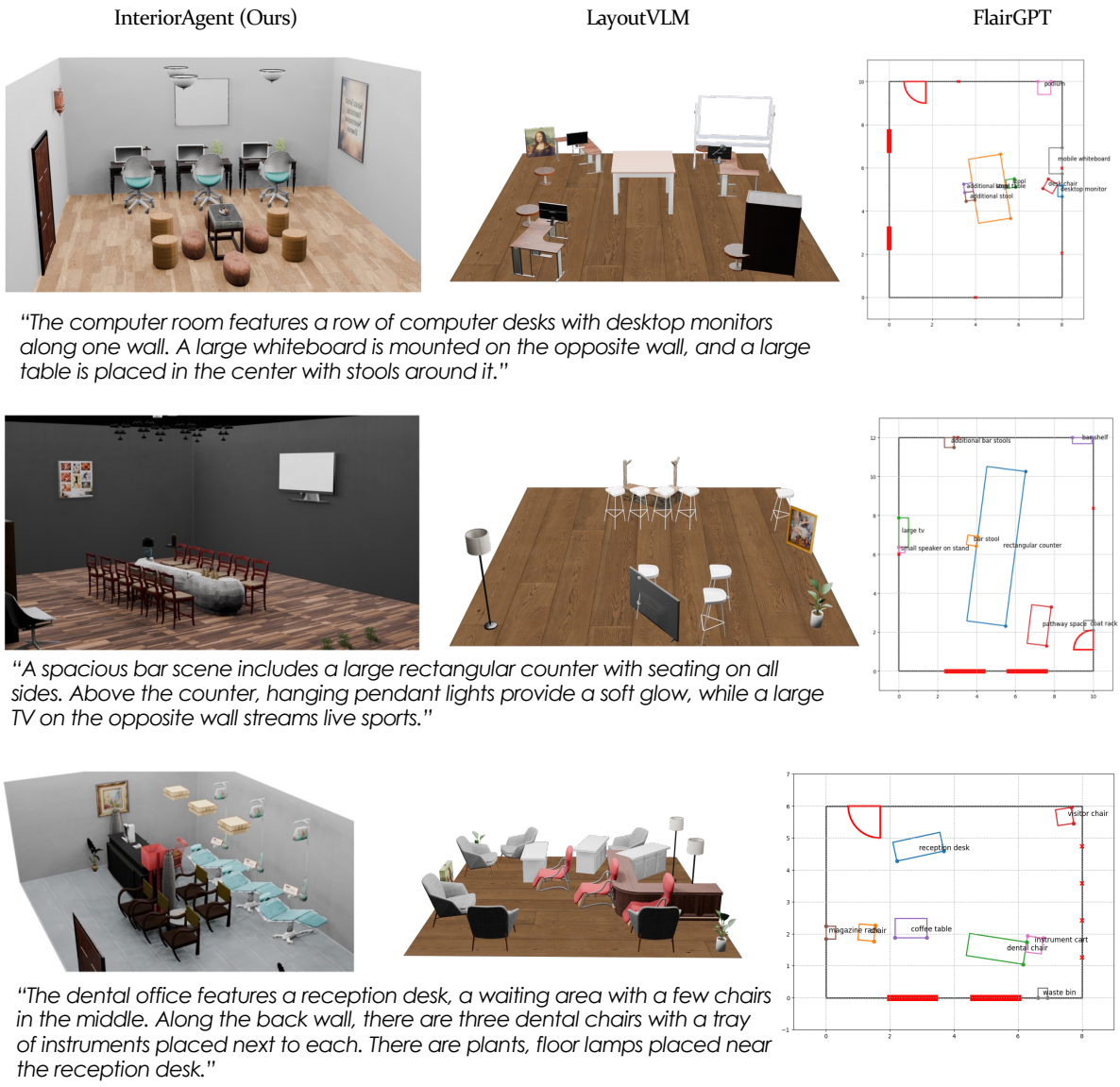


Figure 13. **Comparison to FlairGPT.** MIT Scenes renders for INTERIORAGENT , LayoutVLM [32], and 2D Layouts from FlairGPT [16]. *Computer room:* FlairGPT fails to place desks in a row with monitors; LayoutVLM produces multiple desks but not aligned. *Bar:* FlairGPT inserts a central table but only a single stool, missing seating on all sides. *Dental office:* FlairGPT adds just one dental chair instead of three, and omits plants and lamps near the reception desk. In contrast, INTERIORAGENT generates scenes that are prompt-faithful, aesthetically pleasing, and functionally coherent.

coherent and functional layouts (Figures 13, 14).

Finally, FlairGPT's placement and optimization capabilities are fixed, while INTERIORAGENT offers an extensible framework, allowing designers to easily add new templates for asset retrieval, placement, and optimization.

For comparison, we use the officially released FlairGPT code. However, it only produces 2D layouts without object orientations, preventing generation of 3D scenes and direct rendering comparisons. We therefore conduct a qualitative comparison on six MIT Scenes prompts (Figures 13, 14). Across all cases, INTERIORAGENT consistently outperforms FlairGPT [16] in object retrieval (FlairGPT often omits key objects specified in the prompt), placement (it frequently fails to capture explicit inter-object relations), and optimization (its scenes are less functional).

10. Placement Groups

See Figure 18

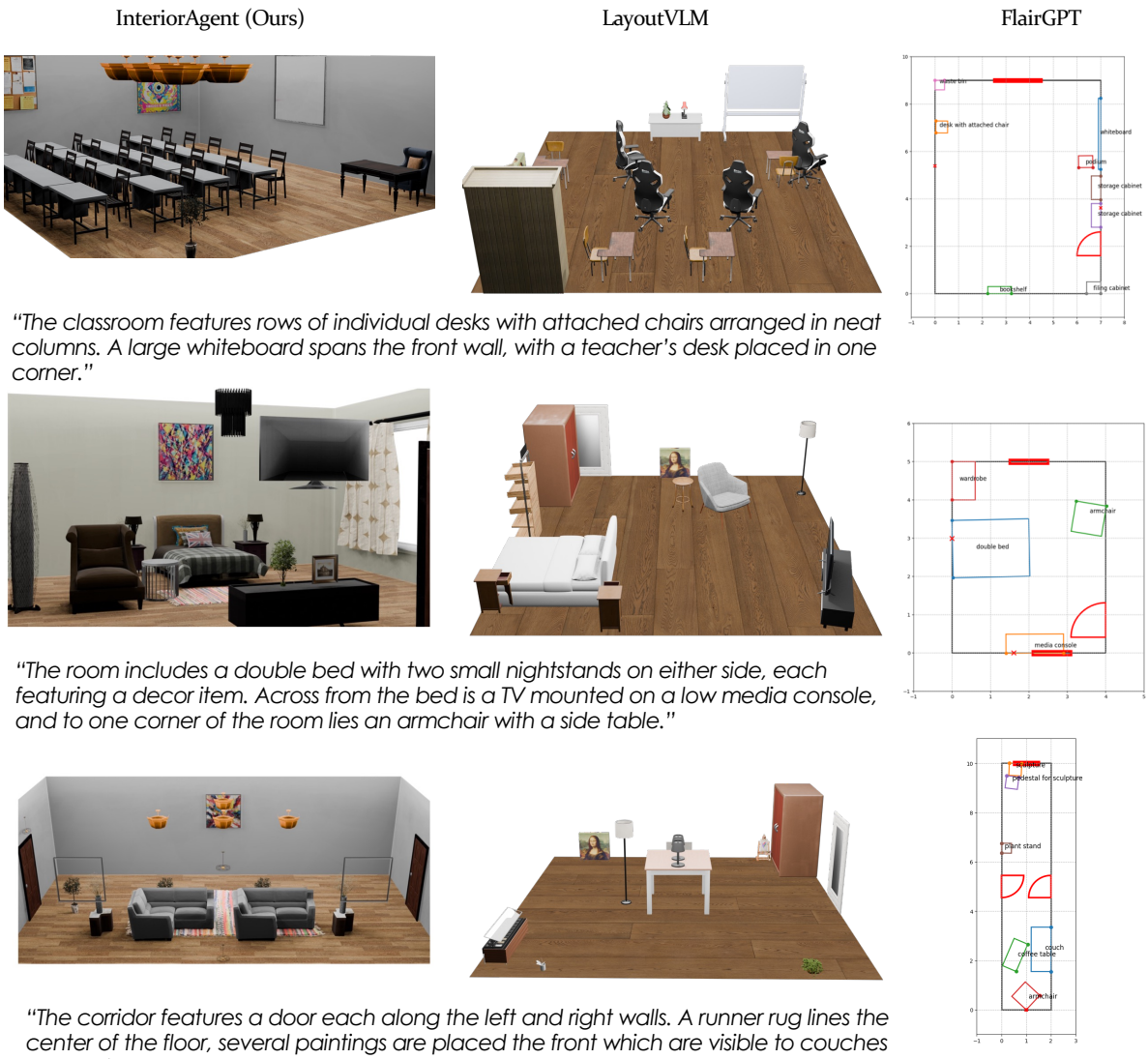


Figure 14. **Comparison to FlairGPT.** MIT Scenes renders for INTERIORAGENT , LayoutVLM [32], and 2D layouts from FlairGPT [16]. *Classroom:* FlairGPT places only a single desk–chair despite the prompt asking for a row of desks with attached chairs, and omits the teacher's desk–chair. *Bedroom:* Nightstands and a side table for the armchair are missing. *Corridor:* Entry–exit doors are misplaced, breaking the layout's intent of having seating in between. In contrast, INTERIORAGENT produces scenes that are prompt-faithful, aesthetically coherent, and functionally sound.

11. Tool Usage

INTERIORAGENT is highly extensible and as such its capabilities can be easily enhanced by providing additional tools. Here we show one example each for object retrieval, placement and optimization.

Object retrieval SceneMotifCoder [33] generates 3D object arrangements through visual program learning. A key feature of SceneMotifCoder is its ability to create *stacks* or *grid*-like arrangements of objects. Since objects like *"a stack of 7 books"* aren't readily available in popular datasets, we enable this functionality in INTERIORAGENT by adding SceneMotifCoder as one of our retrieval tools. To achieve this we write a template as shown in Figure 19, that gives relevant information to INTERIORAGENT about using this tool. Additionally, we follow the official documentation to correctly use SceneMotifCoder and use that to write the `__call__()` function that implements the functionality. Figure 20 shows that with such a minimal input, INTERIORAGENT is able to easily leverage the SceneMotifCoder tool to generate *stacked* objects for its larger scene synthesis

```python
class InteriorAgentAssetRetriever:
    def __init__(self, scene):
        self.name = "Some name like Front3DAssetRetriever"
        self.description = """A brief description of its use case,
                              like 3DFront is great for indoor furniture!"""
        self.example = """A few examples of this retriever in use
                          1. A wooden cabinet 2. A modern sofa"""
    def build(self): ## (Optional) some retrievers may require caching
                     ## or pre-computation such as loading models to GPU
        ## Example for building embeddings for 3DFront/HSSD retriever.
        ## Other datasets may use different embedding methods.
        all_descriptions = [self.VLM(f"""Give a brief description of the asset,
including its type, material, and any notable features. Can use this metadata: {metadata}""",
images=renders) for metadata, renders in self.all_models]
        scales = [self.VLM(f"""What is the width of this asset? {desc}""", images=renders)
                           for desc, renders in zip(all_descriptions, self.all_models.renders())]
        self.embd = [OpenAIEmbeddings().embed_query(desc), scale
                           for desc, scale in zip(all_descriptions, scales)]

    def __call__(self, description: str) -> tuple[str, float]:
        ## Logic to retrieve an asset based on the description
        embd = OpenAIEmbeddings().embed_query(description)
        similarity = self.embd.similarity_search(embd, k=1)
        path, scale = similarity[0].path, similarity[0].scale
        ## Can also use text-to-3D models to generate 3D assets. Result section show more
        ## sophesiticated retrievers including StableDiffusion for generating paintings and
        ## 3D generation repository SceneMotifCoder for stacked assets.
        return path, scale # asset path and width dimensions
```

Figure 15. Template for defining an asset retriever in IDSDL, with an example 3DFront retriever implementation.

```python
class InteriorAgentGroup:
    def __init__(self, scene, *args, **kwargs):
        self.name = "Some name like RelativeGroup"
        ## Some group specific parameters
        self.SIDE_GAP = 0.1  # Gap between objects placed on the sides

    @placemethod ## Decorator to define a placement method
    def place_on_left(self, obj, *args, **kwargs):
        # A brief description of its use case and parameters
        self.description = f"""
Place an object on the left side of the anchor object
Inputs: - obj: The object to be placed on the left side of the anchor object.
Outputs: - None, the object is placed in the scene.
"""
        # A few examples of this placement methods in use
        self.example = f"""
with scene.RelativeGroup() as group:
    group.set_anchor(sofa)  ## Set sofa (initialized earlier) as the anchor object
    end_table = scene.AddAsset("A modern end table")
    group.place_on_left(end_table) ## places the end table on the left side of the sofa
"""
        ## Logic to compute the object pose: x,y,z, theta
        # For relative placement, compute cardinal directions, center point and dimensions of anchor object
        dirs, center, dims = self.get_anchor_data()
        width, height, depth = dims
        front_dir, back_dir, left_dir, right_dir = dirs

        left = center + left_dir * (width / 2 + obj.get_width() / 2 + self.SIDE_GAP)
        x, y, z, theta = left[0], obj.get_height()/2, left[2], 0  # Place on floor
        obj.set_location(x,y,z)
        obj.set_rotation(theta)
        self.add_child(obj)  ## Adds the object to the group

    ## Optionally, control the placement and optimization flow of the group.
    def compile(self):
        for op in self.operations: ## All placements are stored in self.operations
            if self.operations[op] is not None:
                self.operations[op].execute()
        self.grad_optimize() ## Optimize gradient-based constraints
        self.vlm_optimize() ## Compute feedback for VLM-based constraints
```

Figure 16. Template for defining groups in IDSDL, with an example `place_on_left()` method in `RelativeGroup()`.

objective. See Figure 15 for object registration template.

Object placement The teaser figure shows a large scale scene consisting of a cherry blossom tree forest. Usually, such scenes are extremely challenging for an LLM to create given the limited 3D understanding capacity of LLMs. However, when the task is subdivided at a word and later at alphabet level, we note that LLMs can be reliably prompted to achieve desired results. Therefore, we use a prompt engineered LLM (GPT-4o) to serve as a tool for creating their own ascii art. Figure 21 shows the entire code used for this implementation. See Figure 16 for object placement template.

Object arrangement Another interesting use case can be enabled by allowing rendering based losses to work in tandem with gradient and VLM based losses which is made possible because IDSDL seamlessly unifies all the various constraint and types. We show an application where wall

```python
class InteriorAgentConstraint:
    def __init__(self, scene, group, *args, **kwargs):
        self.name = "Some name like OverlapConstraint"
        self.description = f""" ## A brief description of its use case and parameters
Ensures that no two objects overlap with each other
No inputs or outputs, this is a constraint that is applied to the group.
"""
        self.example = f""" ## A few examples of this constraint in use
with scene.<>Group() as group:
    ...
    group.OverlapConstraint()  ## Applies the overlap constraint to the group
"""
        self.type = 'GRADIENT'
    def compute_gradients(self):
        ## Compute pseudo gradients
        objects = self.group.children
        for i in range(len(objects)):
            for j in range(i + 1, len(objects)):
                obj1, obj2 = objects[i], objects[j]
                status, degree = obj1.is_overlap(obj2)
                if status:
                    v1, v2 = obj1.get_location(), obj2.get_location()
                    grad1, grad2 = (v1 - v2) * degree, (v2 - v1) * degree
                    obj1.grad += grad1 ## Add gradient to the total gradient of each object
                    obj2.grad += grad2
```

```python
class InteriorAgentVLMConstraint:
    def __init__(self, scene, group, *args, **kwargs):
        self.name = "ObjectProportionsConstraint"
        self.description, self.example = ... ## Set description, example
        self.type = 'VLM'  # VLM-based constraints
        self.system_desc = f"""
You are given an image showing front, right, back, and left views of a scene with several objects.
Check whether the objects' proportions make sense; if any object is too big or too small,
respond only with a rescale instruction in the form rescale [object] by [factor],
where the factor is a float between 0.1 and 0.9 (e.g., rescale coffee table by 0.5).
If everything looks correct, respond with no rescale."""

    def compute_gradients(self):
        ## Compute VLM-based feedback
        render = self.group.render()
        descriptions = self.group.get_descriptions()
        prompt = f"""
The scene has the following objects: {",".join(descriptions)}
Now reason about the relative proportions of the objects in the scene and to ensure that
they make sense in the context of the scene.
"""
        feedback = self.VLM(prompt, image_paths=[render])
        self.scene.vlm_feedback += feedback
```

Figure 17. Template for defining constraints in IDSDL, with an example gradient based `OverlapConstraint()` (top) and VLM based `ObjectProportionsConstraint()` (bottom)

art needs to be arrangement by comparing them to a target wall at arrangement specified by the user. Figure 22 shows the constraint template being filled with relevant context for LLM to use as well as a logic for computing pseudo gradients to allow movement of the assets such that they match the target as best as possible. Figure 23 shows the scene program generated and the obtained results. See Figure 17 for object arrangement optimization template.

12. Scene Editing Application

Figure 24 shows the visualization of scene editing application discussed in the main paper. INTERIORAGENT 's chat-like interface allows users to generate and iteratively edit scenes, accommodating complex requirements beyond a single prompt.

13. Failure Cases

While INTERIORAGENT significantly outperforms prior work with its program synthesis approach, challenges remain to be addressed in future works. Most importantly, we note that INTERIORAGENT may suffer when the underlying tool malfunctions or doesn't show expected behavior. Figure 25 shows a few such examples. In the leftmost example, the retrieval tool incorrectly retrieves a bundle of t-shirts when asked to retrieve a 'packing station' to be placed in the warehouse. In the middle two examples, while ergonomics constraints account for visibility and clearance, it is possible that the LLM simply misses out on applying relevant

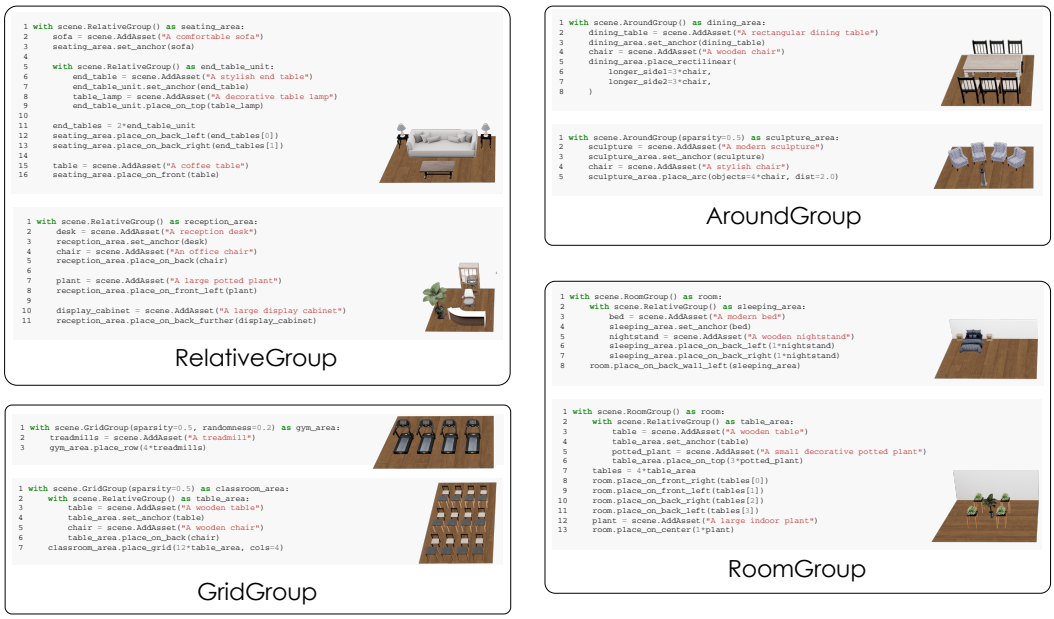


Figure 18. Placement programs involving various group types demoed in method section. Note as few as 10 lines of code are sufficient to represent a wide variety of scenes including living room, reception area (RelativeGroup), dining area, a art museum viewing area (AroundGroup), gym, classroom (GridGroup), bedroom and floweriest shop. (RoomGroup)

constraints when it should, resulting in scenes not being optimized for those constraints. In the rightmost example, a large number of objects coupled with their constraints pose a significant optimization challenge, which can lead to issues such as out of bound problems, as is the case for the example of a densely packed warehouse. We believe that these problems can be mitigated in the future by using more advanced tools and by fine-tuning the LLM for improved tool-awareness. Additionally, INTERIORAGENT 's approach to use task-specific tools for scene synthesis raises important considerations on optimal and robust utilization of multiple tools.

14. Ablation

14.1. Debugging Scene Programs

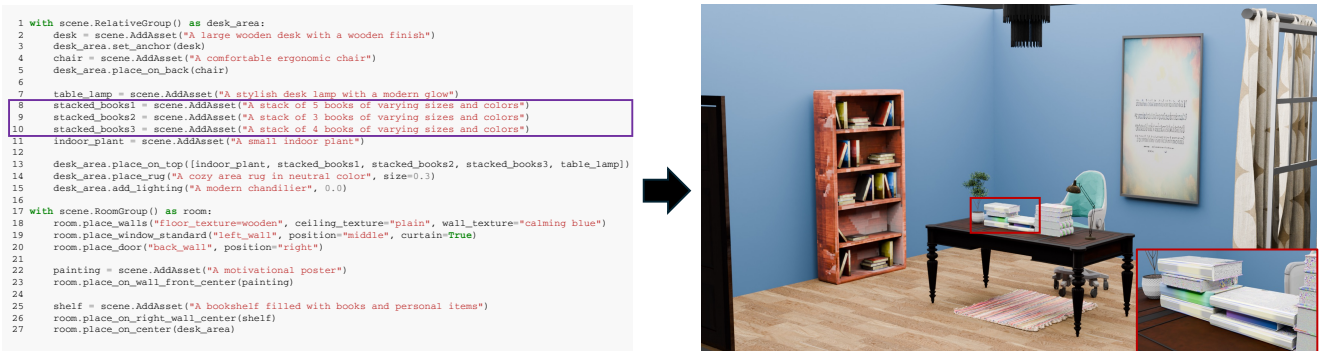
The `Debugger` performs a key role in program synthesis which is to ensure that programs synthesized by PROGRAMSYNTHESIZER are executable and free from syntax issues. In order to achieve this, it leverages a combination of `CodeRefine` and `TraceRefine` in an iterative loop. Upon generating 100 scene programs using the PROGRAMSYNTHESIZER , we observe that `CodeRefine` was used 135 times while `TraceRefine` was called 35 times. Note that `CodeRefine` is invoked every time post PROGRAMSYNTHESIZER to remove potential errors.

14.2. In-Context Examples

A few in context examples are visualized in Fig 26.

```python
class SceneMotifCoderObject(SceneProgAssetRetrieverBase):
    def __init__(self):
        super().__init__()
        self.name = "SceneMotifCoderObject"
        self.description = f"""
This tool returns 'stacked' objects based on an input description which can be used like any other objects in the scene program
"""
        self.examples = """
Following are a few examples of this tool in action:
Example 1:
scene.SceneMotifCoderObject('A table with a chair in front of it')
## Adds a new object to the scene where a chair is placed in front of a table
Example 2:
scene.SceneMotifCoderObject('A stack of 5 cups')
## Adds a new object to the scene where 5 cups are stacked on top of each other
Example 3:
scene.SceneMotifCoderObject('A grid of 5x5 chairs')
## Adds a new object to the scene where 25 chairs are arranged in a 5x5 grid
"""

    def __call__(self, query):
        code = f"""
#!/bin/bash
# Path to the Python executable
PYTHON_EXECUTABLE="/opt/miniconda3/envs/smc/bin/python"
# Path to the inference script
SCRIPT_PATH="/<path to smc>/smc/inference.py"

# Arguments for the script
DESC="{query}"
OUT_DIR="/<path to sceneprog>/sceneprog/tmp/"

cd /<path to smc>/smc
# Execute the Python script with the arguments
$PYTHON_EXECUTABLE $SCRIPT_PATH --desc "$DESC" --out_dir "$OUT_DIR"

# Wait for the program to complete and check exit status
if [ $? -eq 0 ]; then
    echo "Inference completed successfully."
else
    echo "Inference failed with exit code $?."
    exit 1
fi
"""
        with open("tmp/smc_run.sh", "w") as f:
            f.write(code)
        import os
        os.system(f"bash tmp/smc_run.sh")
        import trimesh
        mesh = trimesh.load("tmp/stacked.glb", process=False, force='mesh')
        scale = mesh.bounds[1,0] - mesh.bounds[0,0]
        return "tmp/stacked.glb", scale
```

Figure 19. **Adding SceneMotifCoder[33] to INTERIORAGENT** amounts to filling out the tool template which requires a tool name, description of its role, its input/output as well as a few examples of its demonstration. The core logic of tool is implemented under the `__call__()` method which follows the official documentation on correctly using the tool.



```python
with scene.RelativeGroup() as desk_area:
    desk = scene.AddAsset("A large wooden desk with a wooden finish")
    desk_area.set_anchor(desk)
    chair = scene.AddAsset("A comfortable ergonomic chair")
    desk_area.place_on_back(chair)

    table_lamp = scene.AddAsset("A stylish desk lamp with a modern glow")
    stacked_books1 = scene.AddAsset("A stack of 5 books of varying sizes and colors")
    stacked_books2 = scene.AddAsset("A stack of 3 books of varying sizes and colors")
    stacked_books3 = scene.AddAsset("A stack of 4 books of varying sizes and colors")
    indoor_plant = scene.AddAsset("A small indoor plant")

    desk_area.place_on_top([indoor_plant, stacked_books1, stacked_books2, stacked_books3, table_lamp])
    desk_area.place_rug("A cozy area rug in neutral color", size=0.3)
    desk_area.add_lighting("A modern chandilier", 0.0)

with scene.RoomGroup() as room:
    room.place_walls("floor_texture=wooden", ceiling_texture="plain", wall_texture="calming blue")
    room.place_window_standard("left_wall", position="middle", curtain=True)
    room.place_door("back_wall", position="right")

    painting = scene.AddAsset("A motivational poster")
    room.place_on_wall_front_center(painting)

    shelf = scene.AddAsset("A bookshelf filled with books and personal items")
    room.place_on_right_wall_center(shelf)
    room.place_on_center(desk_area)
```

Scene Program using SceneMotifCoder

Generated 3D Scene

Figure 20. INTERIORAGENT can effectively use novel tools. Here, we show INTERIORAGENT writings programs using the SceneMotifCoder tool for the caption *"Create a study room scene with a desk with 3 heaps of stacked books"*.

Figure 21. **INTERIORAGENT tool use example:** Using the Group template (1–3), we implement an ASCII art generation tool driven by an LLM prompt. The teaser illustration was produced from the caption *"forest made to look like INTERIORAGENT 3DV 2026 VANCOUVER"*. (4) INTERIORAGENT enables such expressive scenes with minimal code.

```python
class RenderingConstraint(ConstraintBase):
    def __init__(self, group, wall, paintings, target_image_path):
        from painting_detector import PaintingDetector
        self.name = 'RenderingConstraint'
        self.description = f"""
Helps in optimizing placement of paintings on the wall to match a given image target.
Inputs:
- wall: The wall name (str)
- paintings: List of painting objects (list)
- target_image_path: Path to the target image (str)
"""
        self.examples = f"""
with scene.RoomGroup() as room:
    ...
    painting = scene.AddAsset("A Beautiful Landscape")
    paintings = 3*paintings
    room.place_on_wall_back_left(paintings[0])
    room.place_on_wall_back_center(paintings[1])
    room.place_on_wall_back_right(paintings[2])

    room.RenderingConstraint("back_wall", paintings, "path/to/target/image.jpg")
"""

        self.type = 'GRADIENT'
        self.painting_detector = PaintingDetector()
        self.target_image_path = target_image_path

        self.target_centroids, self.target_bbox = self.painting_detector(self.target_image_path, resize=(1920,1080))
        self.wall = wall
        self.paintings = paintings

        super().__init__(self.name, group)

    def compute_gradients(self):
        ## Render the wall with paintings
        current_image_path = self.group.render_wall(self.wall, self.paintings)
        ## Detect centroids of each painting using Owlv2
        centroids, tmp = self.painting_detector(current_image_path, resize=(1920,1080))

        ## Use hungarian method to derieve optimal 1-1 mapping between centroids.
        perm = self.painting_detector.compute_mapping(centroids, self.target_centroids)
        mapped_centroids = [self.target_centroids[i] for i in perm]

        for i, painting in enumerate(self.paintings):
            grad = mapped_centroids[i] - centroids[i] ## pseudo gradient
            img_grad[0] *= 1/1920
            img_grad[1] *= 1/1080
            painting.grad += np.array([img_grad[0], img_grad[1], 0], dtype=np.float32)
```

Figure 22. **INTERIORAGENT tool use example:** Using the Constraints template, we implement an optimization routine for arranging wall scenery to match a target image. Frames are detected with OWLv2 using prompts such as "painting", "picture frame", "wall art", and "poster". The centers of bounding boxes from target and source renderings are extracted, matched via the Hungarian algorithm, and the scenery is shifted toward their assigned centroids.

```python
with scene.RelativeGroup() as seating_area:
    sofa = scene.AddAsset("A comfortable sofa")
    seating_area.set_anchor(sofa)
    coffee_table = scene.AddAsset("A stylish coffee table")
    seating_area.place_on_front(coffee_table)
    floor_plant = scene.AddAsset("A large floor plant")
    seating_area.place_on_right_further(floor_plant)


with scene.RoomGroup() as room:
    room.place_walls("wooden", "plain_white", "plain_white")
    sofa = scene.AddAsset("A comfortable sofa")
    room.place_on_back_wall_center(seating_area)

    painting1 = scene.AddAsset("A painting on Christianity")
    painting2 = scene.AddAsset("A painting on Buddhism")
    painting3 = scene.AddAsset("A painting on Hinduism")
    paintings = [painting1, painting2, painting3]
    room.place_on_wall_back_left(paintings[0])
    room.place_on_wall_back_center(paintings[1])
    room.place_on_wall_back_right(paintings[2])
    room.RenderingConstraint("back_wall", paintings, "target.jpg")
```

Scene Program using RenderingConstraint

w/o rendering constraint

Target for optimizing painting arrangements

Generated 3D Scene

Figure 23. **Using Rendering constraint in code:** INTERIORAGENT uses the available context to easily write a program for the caption *"Create a living room with the back wall having three paintings: symbolizing Christianity, Buddhism and Hinduism as per the following arrangement (pass image path for target)"*.

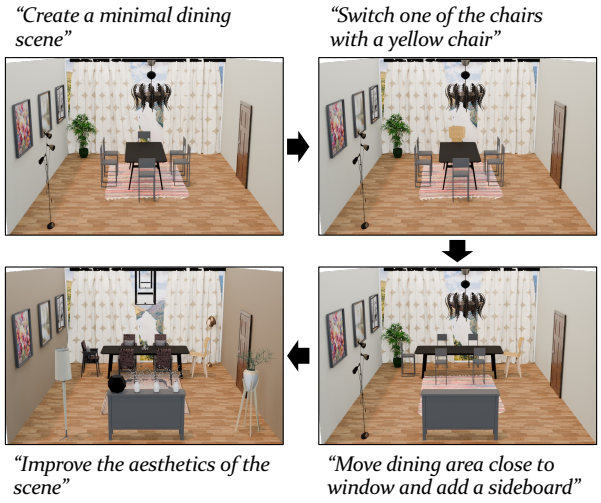


Figure 24. INTERIORAGENT allows user-driven scene customization through a chat interface. Starting from a minimal scene, INTERIORAGENT responds to user inputs to change colors, rearrange, add furniture and improve decor, while maintaining the scene balance.

Wrong retrieval | Missed visibility | Missed clearance | Optimization too difficult

Figure 25. Some failure cases of INTERIORAGENT .

Figure 26. A few in-context examples used in INTERIORAGENT .