SPATIALFACTORY: UNLEASHING MLLM SPATIAL INTELLIGENCE FOR COMPOSITIONAL SCENE SYNTHESIS

Anonymous authors
Paper under double-blind review

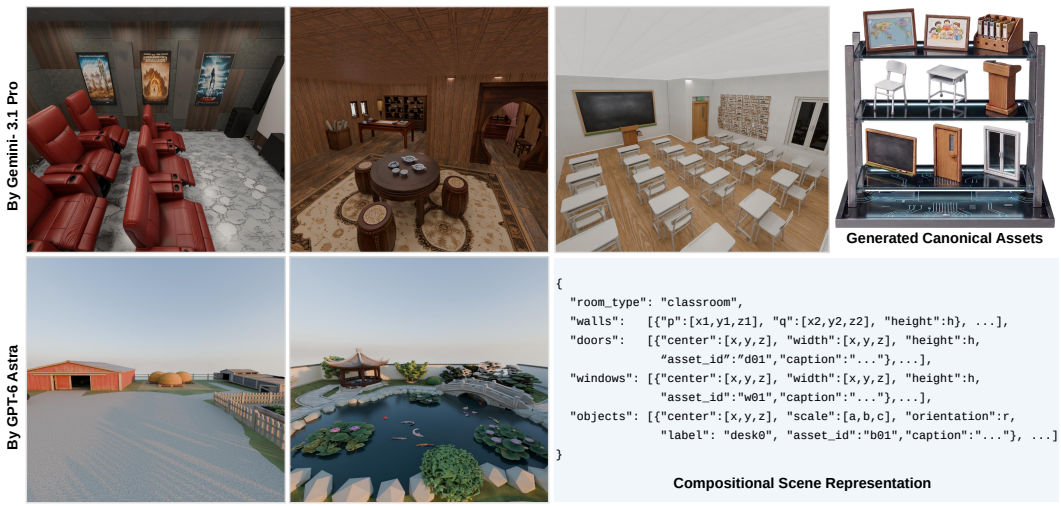


Figure 1: Diverse results and compositional scene representations produced by SpatialFactory.

ABSTRACT

High-fidelity and diverse compositional 3D scene synthesis is essential for AR/VR, robot simulation, and the curation of training data for scene understanding and generation. However, existing agentic scene synthesis workflows typically rely on simple in-context linguistic priors while lacking visual planning and imagination, severely limiting the realism and diversity of the generated scenes. To address this issue, we present SpatialFactory, an end-to-end agentic compositional scene synthesis framework built on recent MLLM advances in spatial intelligence. It introduces Multimodal Generation LMs as global top-down visual-spatial imagination engines to leverage vast 2D priors for enhanced generalization and realism, while leveraging Multimodal Understanding LMs and a 3D lifting tool to support a more stable scene parsing and reasoning process from visual hypotheses to compositional 3D scenes without the need for any specialized visual understanding models. Furthermore, to evaluate the effectiveness of the proposed scene synthesis method and the spatial reasoning performance of advanced MLLMs, we introduce a benchmark comprising 111 high-quality, professionally crafted scenes with ground-truth meshes across 12 common indoor categories, accompanied by a comprehensive evaluation pipeline. SpatialFactory significantly outperforms existing agentic methods on this benchmark, while also demonstrating strong generalization and controllability under complex floorplan constraints and detailed textual descriptions. Code and generated scenes will be publicly released.

1 INTRODUCTION

As spatial intelligence becomes increasingly central to AI research, the automated synthesis of high-quality 3D scenes has emerged as an indispensable requirement not only for immersive AR/VR applications but also for the curation of training data in spatial understanding and generation tasks Zhou

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

et al. (2026); Wang et al. (2026a); Yang et al. (2026b). To date, there has been a significant body of related work encompassing both learning-based methods Gao et al. (2023); Li et al. (2019); Paschalidou et al. (2021); Ritchie et al. (2019); Wang et al. (2019); Zhang et al. (2020); Tang et al. (2024); Bai et al. (2025); Lin & Mu (2024) and LLM/VLM/agent-based methods Feng et al. (2023); Yang et al. (2024); Sun et al. (2025); Yang et al. (2026a); Pfaff et al. (2026); Çelen et al. (2024).

However, these methods struggle to meet the realism and diversity requirements of spatial data generation. Learning-based approaches rely on scarce public layout datasets, such as 3D-FRONT Fu et al. (2021a) and 3D-FUTURE Fu et al. (2021b), which consist of a modest collection of suboptimal scenes. These datasets lack diversity in object categories and spatial arrangements, leading to homogeneous and simple generated scenes. LLM/VLM/agent-based approaches leverage in-context learning capabilities and prior knowledge acquired during pre-training to mitigate the diversity issues of learning-based methods. However, because they mainly use VLMs as visual-to-text critics while lacking a visual-spatial imagination process, they still struggle with scene diversity and are prone to collapse in complex scenes. Moreover, they are limited to simple rectangular floorplans and coarse-grained textual descriptions, which further limits their controllability and usability.

Frontier MLLMs have recently exhibited emergent spatial generation and understanding capabilities that far surpass previous levels. Specifically, Multimodal Generation LMs Google (2025); OpenAI (2025); Seedream et al. (2025); Wu et al. (2025) have acquired extensive 2D scene priors from massive image datasets and demonstrated exceptional spatial editing capabilities, offering a complementary way to compensate for scarce 3D priors through diverse global 2D visual designs in scene synthesis; meanwhile, Multimodal Understanding LMs Google (2026); ByteDance (2026); Anthropic (2026); OpenAI (2026a); Team et al. (2026); Qwen (2026) have progressed beyond high-level semantics to demonstrate strong capabilities in tasks traditionally addressed by specialized models, such as object detection Liu et al. (2024) and orientation estimation Wang et al. (2024; 2026c), and have further exhibited more flexible spatial intelligence, opening up the possibility of translating global 2D designs into compositional 3D scenes.

To explore the spatial capability boundaries of current MLLMs in compositional scene synthesis, we propose SpatialFactory, an end-to-end agentic compositional scene synthesis framework. Mimicking human imagination and planning in spatial design, SpatialFactory uses Multimodal Generation LMs as visual-spatial imagination engines to transform user-specified floorplans and detailed design intent into global top-down image plans. It then leverages Multimodal Understanding LMs combined with a simple yet effective 3D lifting tool to parse these plans into a coarse 3D layout, which is subsequently refined and edited into the final scene through agentic spatial reasoning. The entire process does not rely on specialized scene understanding models, such as object detection, semantic segmentation, or depth estimation models. To evaluate the effectiveness of the proposed scene synthesis method and the spatial reasoning performance of advanced MLLMs, we introduce a benchmark comprising 111 high-quality, professionally crafted scenes with ground-truth meshes across 12 common indoor scene categories, accompanied by a comprehensive evaluation pipeline for both scene synthesis and spatial understanding. Experimental results on our benchmark demonstrate that SpatialFactory exhibits a clear advantage over prior agentic methods on the scene synthesis benchmark, particularly under complex layouts and floorplans, while also outperforming prior pipelines that rely on specialized models on the spatial understanding benchmark.

Our contributions are summarized as follows:

- We propose **SpatialFactory**, an end-to-end agentic compositional scene synthesis framework that integrates global visual-spatial imagination with MLLM-based spatial reasoning, enabling fine-grained control over floorplans and design intent without specialized scene understanding models.
- We introduce **SpatialFactory Benchmark**, a benchmark of 111 professionally crafted scenes spanning 12 common indoor categories, where each scene is paired with fine-grained annotations together with its corresponding layout and geometric ground-truth, supporting not only scene synthesis evaluation but also a broader spectrum of scenarios such as spatial understanding.
- As an auxiliary contribution, we curate 1K generated scenes and develop a toolkit that builds full 3D scenes from compositional representations, plans scene-covering camera trajectories, and renders perspective and panoramic videos with depth, normal, and semantic maps plus per-view point clouds, meshes, and voxels for potential use in future spatial understanding research.

2 Related Work

Data-driven Scene Synthesis. Before the emergence of LLMs, scene synthesis primarily relied on data-driven modeling and rule-based optimization. Early methods employed probabilistic models to learn indoor scene layout distributions and optimization algorithms to refine spatial relationships among objects Xu et al. (2002); Yu et al. (2011); Merrell et al. (2011); Fisher et al. (2012); Liu et al. (2014); Kermani et al. (2016). With the advent of deep learning, subsequent methods adopted deep generative models, including GANs Goodfellow et al. (2014), VAEs Kingma & Welling (2013), autoregressive models Vaswani et al. (2017), and diffusion models Ho et al. (2020); Lipman et al. (2022), to learn scene layout distributions Gao et al. (2023); Li et al. (2019); Paschalidou et al. (2021); Ritchie et al. (2019); Wang et al. (2019); Zhang et al. (2020); Tang et al. (2024); Bai et al. (2025); Lin & Mu (2024). However, existing layout datasets such as 3D-FRONT Fu et al. (2021a) and 3D-FUTURE Fu et al. (2021b) offer limited diversity in scene categories and spatial arrangements, with layouts often exhibiting simple, repetitive patterns. Consequently, these models tend to generate simplistic scenes and generalize poorly beyond the training distribution.

Agentic Scene Synthesis. In recent years, the rapid advancement of LLMs has significantly propelled the development of scene synthesis. More and more researchers are leveraging the in-context learning capabilities and commonsense priors of large models to address the generalization problem of scene synthesis Feng et al. (2023); Yang et al. (2024); Sun et al. (2025); Yang et al. (2026a); Pfaff et al. (2026); Pun et al. (2025); Çelen et al. (2024). SceneWeaver Yang et al. (2026a) utilizes a RAG-like (Retrieval-Augmented Generation Lewis et al. (2020)) mechanism to obtain scene template initializations, followed by refinements using the ReAct (Reasoning and Acting Yao et al. (2022)) paradigm. Nevertheless, this methodology relies heavily on existing scene templates. SceneSmith Pfaff et al. (2026) employs an agentic generation process that decomposes scene synthesis into a hierarchical workflow. This decomposition somewhat improves generation stability. However, these methods primarily employ VLMs for visual-to-text feedback rather than visual-spatial imagination, limiting scene diversity and generation stability in complex scenes.

Image-based Scene Parsing. Image-based scene parsing pipelines decompose a scene into its constituent objects by sequentially applying a series of specialized scene understanding models for object detection, semantic segmentation, depth estimation, and pose estimation, followed by asset retrieval or generation. Among them, some works aim at 3D annotation of 2D images Yao et al. (2026), while others derive compositional 3D scene representations from local image observations Yao et al. (2025); Gu et al. (2025); Ling et al. (2025); Zhu et al. (2025). However, most of these works focus on local scene understanding and fail to produce consistently usable results when adapted to global compositional scene synthesis, due to occlusions and substantial depth and pose estimation errors. Imaginarium Zhu et al. (2025) applies this kind of pipeline to compositional scene synthesis through a generation-and-parsing paradigm. However, this pipeline may rely heavily on its fine-tuned FLUX.1-dev Black Forest Labs (2024) model, and the resulting layouts become disorganized in out-of-distribution scenes. In contrast, our method combines the spatial intelligence in MLLMs with a top-view 3D lifting method to achieve more stable parsing of complex and diverse top-view scenes without any specialized scene understanding models.

3 Method

3.1 Overview

We first introduce the problem formulation and compositional scene representation that SpatialFactory targets. As illustrated in Figure 2, SpatialFactory adopts a tree-structured agent architecture managed by a top-level agent, *Spatial Composer*, which orchestrates two cooperating modules, *Visual-spatial Imagination* and *Compositional 3D Realization*, each of which has its own tools and, when needed, lower-level sub-agents. Concretely, the Visual-spatial Imagination module (Sec. 3.3) generates a floorplan-aligned top-view visual plan that reflects the user's design intent. Conditioned on this visual plan, the Compositional 3D Realization module (Sec. 3.4) further produces a complete, compositional scene description with associated 3D assets.

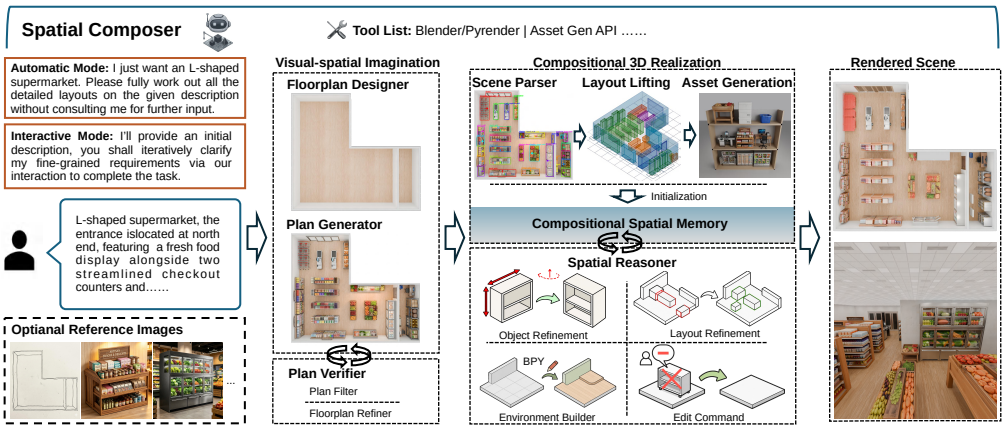


Figure 2: Overview of SpatialFactory. Driven by a top-level *Spatial Composer* agent, our framework consists of two cooperating modules: *Visual-spatial Imagination* externalizes textual intent into floorplan-aligned top-view visual plans, and *Compositional 3D Realization* converts these plans into compositional 3D scenes through a *Scene Parser*, a *3D Layout Lifting* tool, an *Asset Generation* tool, and a final *Spatial Reasoner* for scene refinement, completion, and editing.

3.2 Problem Formulation

Given a multimodal user instruction $\mathcal{I} = (T, \mathcal{R})$, where T is a textual description ranging from a simple scene-type tag (*e.g.*, "Modern-Bedroom") to a fine-grained specification of the floorplan and scene contents, and $\mathcal{R} = \{R_1, \ldots, R_M\}$ is an optional set of reference images (*e.g.*, floorplan sketches, diagrams, or furniture photos), our goal is to synthesize a compositional 3D scene

$$\mathcal{S} = \big(\mathcal{L},\ \mathcal{M},\ \mathcal{T}\big), \tag{1}$$

where $\mathcal{L}$ is a compositional scene representation that fully specifies the architectural shell together with the in-scene object layout, $\mathcal{M}$ is an asset library storing one textured 3D mesh per unique instance, and $\mathcal{T}$ is an optional architectural-texture set.

Compositional Scene Representation. We adopt a *Compositional Scene Representation* as our scene representation $\mathcal{L}$. As illustrated in Figure 2, every Compositional Scene Representation is a JSON object with four fields: **(i)** a `room_type` string that declares the high-level room category; **(ii)** a `walls` array $\mathcal{W} = \{w_j = (\mathbf{p}_j, \mathbf{q}_j, h_j)\}_{j=1}^{J}$ that defines the floor polygon by the two endpoints and the height of every segment; **(iii)** `doors` and `windows` arrays $\mathcal{U} = \{u_k = (\ell_k, \mathbf{p}_k, w_k, h_k, \alpha_k)\}_{k=1}^{K}$, where each opening is parametrized by a category label, a 3D center, a width, a height, and a unique asset identifier α_k that maps the opening to its corresponding `.glb` mesh in the asset library $\mathcal{M}$; and **(iv)** an `objects` array $\mathcal{O} = \{b_i = (\ell_i, \mathbf{p}_i, \mathbf{s}_i, \theta_i, \alpha_i)\}_{i=1}^{N}$, where each entry describes one object instance by its semantic label ℓ_i, 3D center $\mathbf{p}_i \in \mathbb{R}^3$, axis-aligned scale $\mathbf{s}_i = (a_i, b_i, c_i) \in \mathbb{R}^3_+$ (with b_i aligned to the orientation direction), orientation angle θ_i, and asset identifier α_i that maps the object to its mesh in $\mathcal{M}$. The full representation $\mathcal{L}$ aggregates the `room_type` together with $\mathcal{W}, \mathcal{U}$ and $\mathcal{O}$.

3.3 Visual-spatial Imagination

The Visual-spatial Imagination module establishes a faithful and visually grounded starting point for subsequent 3D understanding and reasoning. In this module, we incorporate two generation agents (*Floorplan Designer* and *Plan Generator*) and one verifier agent (*Plan Verifier*).

Floorplan Designer. Driven by the user instruction, the *Floorplan Designer* produces a floorplan $\mathcal{F}$ in JSON format together with a detailed scene description that conforms to the user instruction; in particular, when the user only provides a coarse scene-type description, the Floorplan Designer elaborates the in-room contents, either autonomously in automatic mode or through user interaction in interactive mode. It then invokes the rendering tool to render the empty top-view image I_{empty} of $\mathcal{F}$ and aligns the image space with the metric world space (see Appendix A.2 for details); resulting camera parameters $\boldsymbol{\xi}$ are recorded for the Compositional 3D Realization module.

Plan Generator. Given I_{empty} and the scene description, the *Plan Generator* leverages the spatial editing capability of a Multimodal Generation LM to synthesize a furnished top-view image I_{scene} that is floorplan-aligned with I_{empty}.

Plan Verifier. To guarantee that I_{scene} is floorplan-aligned with I_{empty}, we adopt a *bidirectional verification* strategy. Specifically, the *Plan Filter* and the *Plan Generator* form a multi-agent generate-and-filter paradigm: the former compares the generated I_{scene} with I_{empty}, while the latter regenerates I_{scene} whenever the two floorplans disagree noticeably. Once a coarsely consistent pair is obtained, another ReAct agent named *Floorplan Refiner* performs the reverse verification by comparing them and incrementally editing the floorplan JSON $\mathcal{F}$ to align it with I_{scene}, re-rendering I_{empty} after every edit until the two images are wall-by-wall aligned. The module produces a triplet $(\mathcal{F}, I_{\text{empty}}, I_{\text{scene}})$ together with the camera parameters $\boldsymbol{\xi}$, on top of which the rest of the pipeline operates.

3.4 Compositional 3D Realization

Conditioned on the triplet $(\mathcal{F}, I_{\text{empty}}, I_{\text{scene}})$, the Compositional 3D Realization module lifts the top-view visual plan into a 3D scene through four sub-stages: a *Scene Parser* that parses I_{scene} element-by-element into a 2D layout enriched with per-object scene graphs, a *3D Lifting* tool that lifts the parsed result into a coarse 3D layout, an *Asset Generation* tool that produces and canonicalizes one textured mesh per unique object, and a *Spatial Reasoner* that refines the scene through visual feedback, environment building, and interactive editing.

Scene Parser. As shown in Figure 2, the *Scene Parser* first performs a three-step understanding of I_{scene} before the 3D layout is recovered. **(i) Object detection.** Without invoking any external tool, the Scene Parser detects the set of newly added objects (including doors and windows) with respect to I_{empty}, and produces, for each instance, an open-vocabulary semantic label ℓ_i, an appearance-only caption, and a 2D bounding box $\mathbf{b}_i^{2D} = [x_1, y_1, x_2, y_2]$. **(ii) Scene-graph extraction.** The Scene Parser extracts a scene graph $\mathcal{G}_i$, which records each object's supporting and attaching relations and the walls it sides against. Conditioned on $\mathcal{G}_i$ and the visual evidence, the Scene Parser predicts a *discretized* orientation angle θ_i for every object; we restrict θ_i to integer multiples of 15°, which we empirically find sufficient to faithfully represent realistic furniture orientations. **(iii) Height-range estimation.** Finally, the Scene Parser infers the vertical extent $[h_i^{\text{low}}, h_i^{\text{high}}]$ of every object by jointly considering scene graph $\mathcal{G}_i$ and common-sense priors, so that the height estimate preserves object support relations. These parsed attributes are then used by the 3D Lifting tool.

3D Lifting. After the Scene Parser exposes the 2D understanding result, the Spatial Composer invokes a 3D lifting tool to produce a coarse 3D layout, which is later refined by the *Spatial Reasoner*.

Given the height range $[h_i^{\text{low}}, h_i^{\text{high}}]$ of every object and the camera parameters $\boldsymbol{\xi}$, we approximate every object as a cuboid and decompose the discrepancy between its 2D bounding box and the corresponding 3D bounding box into per-edge corrections along the x and y axes. Let $\mathbf{o} = (o_x, o_y, o_z)$ be the camera position and ρ be the pixel-to-metric ratio (Sec. 3.2); we first map every 2D box to the metric world by $[\tilde{x}_1, \tilde{y}_1, \tilde{x}_2, \tilde{y}_2] = \rho \cdot [x_1, y_1, x_2, y_2]$, which corresponds to the metric-scale region on the ground plane that the 2D detection box back-projects to through the top-view perspective camera. We then obtain the x and y extents of the coarse 3D bounding box, denoted $[\hat{x}_1, \hat{y}_1, \hat{x}_2, \hat{y}_2]$, by applying four signed corrections $\{\Delta_{\text{l}}, \Delta_{\text{r}}, \Delta_{\text{t}}, \Delta_{\text{b}}\}$ along the left, top, right and bottom:

$$[\hat{x}_1, \hat{y}_1, \hat{x}_2, \hat{y}_2] \;=\; [\tilde{x}_1 + \Delta_{\text{l}},\; \tilde{y}_1 + \Delta_{\text{t}},\; \tilde{x}_2 - \Delta_{\text{r}},\; \tilde{y}_2 - \Delta_{\text{b}}], \tag{2}$$

Each correction follows the perspective relation $\Delta_\bullet = (h_\bullet / o_z) d_\bullet$, where $d_\bullet$ is the signed distance from the camera principal point to the corresponding edge and $h_\bullet \in \{h_i^{\text{low}}, h_i^{\text{high}}\}$. Appendix A.3 details the height selection and sign convention, with an x-axis illustration.

With these, the 3D center $\mathbf{p}_i$ is taken as the midpoint of $[\hat{x}_1, \hat{y}_1, \hat{x}_2, \hat{y}_2]$ and $[h_i^{\text{low}}, h_i^{\text{high}}]$, while the vertical extent is $c_i = h_i^{\text{high}} - h_i^{\text{low}}$. The orientation-aligned in-plane extents (a_i, b_i) are estimated by linearly interpolating between the corrected x and y extents according to θ_i (full formula in Appendix A.3). The output of this stage is the coarse layout $\mathcal{O}^{\text{init}} = \{(\ell_i, \mathbf{p}_i, \mathbf{s}_i, \theta_i, \mathcal{G}_i)\}$, where $\mathcal{G}_i$ is the scene graph extracted in step (ii) and serves as the relation prior for downstream reasoning.

Asset Generation. In this stage, SpatialFactory generates one 3D mesh for every unique instance of $\mathcal{O}^{\text{init}}$ and ensures it is canonical, with all generation branches running in parallel through a

thread/process pool. For each instance, we (1) crop or segment the corresponding region from I_{scene}, (2) call a Multimodal Generation LM to translate this top-view crop into a high-resolution three-quarter view image, guided by the caption to focus on the target instance and suppress distractor objects in the crop, and (3) feed the three-quarter view to an off-the-shelf image-to-3D generator to produce the textured mesh $\mathcal{M}_i$. Optionally, the user can configure SpatialFactory to additionally synthesize the architectural-texture set $\mathcal{T}$ for the floor, walls and ceiling from I_{scene}.

Asset Canonicalization. Although advanced image-to-3D generators (*e.g.*, SAM3D Chen et al. (2025), Hunyuan3D 3.0 Tencent Hunyuan (2025)) are nominally trained to generate 3D assets following the GLTF convention (face $+z$, world-up $+y$), in practice the produced meshes are often tilted or flipped. Inspired by previous work Wang et al. (2024; 2026c), we design a fast and effective canonicalization procedure. We first render the asset from four canonical viewpoints (front / left / right / up) and ask the MLLM to identify which one is the true top view; the asset is then rotated to align the predicted top direction with $+z$. We empirically find these four views sufficient. Subsequently, we render the asset from three views (front / left / up) and ask the MLLM to predict residual tilt angles around the three axes, which are applied to the mesh to remove the small remaining tilt.

Spatial Reasoner. The Spatial Reasoner initializes a *Compositional Spatial Memory* from the parsed scene representation and generated assets. Through iterative visualization and feedback, it refines object alignment and scene-wide layout. Its *Environment Builder* then completes missing architectural elements and scene textures, while the same stage handles user edit commands in interactive mode.

Object-Level Reasoning. The parsed orientation θ_i may disagree with the forward direction of a generated asset, especially for symmetric or L-shaped objects. The *Asset Corrector* renders each canonicalized mesh under the original top-view camera and compares it with the corresponding crop of I_{scene}. Its *Yaw Corrector* aligns their orientations, while its *Scale Corrector* adjusts $\mathbf{s}_i$ until the rendered footprint matches the observed one. Both corrections update the asset state in the Compositional Spatial Memory.

Layout-Level Reasoning. The Spatial Reasoner follows a ReAct-style render-and-revise loop over the Compositional Spatial Memory. At each iteration, it examines the structured scene state and its visualizations alongside the reference visual plan, identifies issues such as collisions, out-of-bound objects, implausible placements, or inconsistent object relations, and revises the spatial layout to resolve them. The revised scene is then visualized again, allowing the next observation to assess the effect of the edit. The loop ends when no prominent issue remains or the iteration limit is reached, yielding the refined layout $\mathcal{O}$.

Environment Building and Editing. The *Environment Builder* first plans the architectural or terrain elements and scene textures still missing from the reconstructed scene. Guided by the Compositional Spatial Memory and scene renderings, it generates `bpy` (Blender Python) code to construct elements such as platforms, stairs, partitions, and outdoor terrain, and to realize the planned textures through material nodes. Each code revision is rendered and inspected before the next edit, while the resulting elements remain individually identifiable in the compositional representation. In interactive mode, the Spatial Reasoner also processes user edit commands to add, remove, or modify objects, updating the same memory. These updates produce the final compositional scene $\mathcal{S} = (\mathcal{L}, \mathcal{M}, \mathcal{T})$.

4 Experiments

4.1 SpatialFactory Benchmark

Although prior benchmarks for indoor scene synthesis already exist (*e.g.*, SceneEval Tam et al. (2026)), the annotations they contain are mostly accompanied only by coarse descriptions, and do not come with paired layout and geometric ground-truth that supports more complex and richer evaluation scenarios. In addition, their annotations omit floorplan descriptions, a long-overlooked yet crucial aspect of scene synthesis that is intimately tied to spatial layout. To enable a more rigorous evaluation of fine-grained instruction-driven scene synthesis under *complex* floorplans, and to additionally support more evaluation scenarios such as spatial layout reasoning from a given image, we introduce **SpatialFactory Benchmark**, a benchmark of *111* professionally crafted scenes spanning

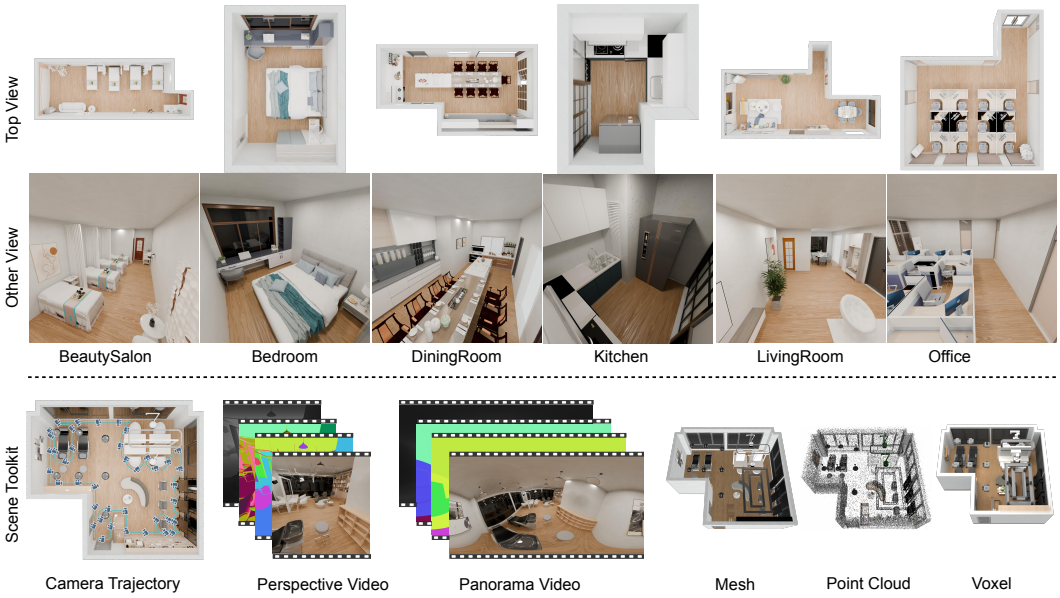


Figure 3: Top: selected SpatialFactory Benchmark scenes in top and perspective views. Bottom: scene toolkit outputs: camera trajectory, perspective and panoramic videos with depth, normal, and semantic maps, plus mesh, point cloud, and voxels.

12 common indoor scene categories. Every scene is annotated with (1) a structured layout ground-truth in our Compositional Scene Representation (including realistic and complex floorplans), (2) a textured 3D mesh for every object, (3) a fine-grained scene-and-floorplan natural-language description, and (4) a top-view rendering paired with a reusable rendering tool that supports a wide spectrum of downstream evaluation scenarios. The 12 categories include *Bedroom*, *LivingRoom*, *Balcony*, *Bathroom*, *BeautySalon*, *Classroom*, *ConferenceRoom*, *DiningRoom*, *Kitchen*, *Office*, *TeaRoom* and *WalkInCloset*. Figure 3 shows representative benchmark renderings and scene toolkit outputs.

Scene Synthesis Metrics. In text-to-scene synthesis, we evaluate each synthesized scene from two perspectives: a set of *subjective* scores given by MLLMs that score the scene along high-level aspects, and a set of *objective* metrics that quantitatively evaluate specific aspects.

Subjective Scores. Similar to prior work Yang et al. (2026a), we rate each scene with an MLLM judge along four axes on a 0–10 scale: *Functional Integrity* (**Func.**, layout fitness to the room purpose), *Spatial Logic & Physics* (**Spat.**, plausibility of object grouping, orientation, fit and physics), *Circulation & Flow* (**Circ.**, navigability and clearance), and *Aesthetic Coherence* (**Aest.**, stylistic harmony and visual balance).

Objective Metrics. Beyond subjective scores, we report seven objective metrics: *Object Recall* (**OR**↑), the recall over objects in the prompt; *Instruction Recall* (**IR**↑), the recall over per-prompt atomic instructions (covering placement, relations, attributes, etc.) and a strong indicator of fine-grained controllability; *Out-of-Bound* (**OoB**↓), the fraction of objects whose mesh falls outside the floorplan; *Collision* (**Col**↓), the fraction colliding with another object; *Navigability* (**Nav**↑), the fraction of floor area reachable for an agent of human radius; *Door/Window Clearance* (**DWC**↑), the fraction of openings with unobstructed mandatory clearance, excluding methods unable to generate windows; and *Floating* (**Float**↓), the fraction of unsupported objects.

Spatial Understanding Metrics. To explore the spatial understanding boundary of advanced Multimodal Understanding LMs (VLMs), we modify SpatialFactory into a pipeline so that, given a top-view rendering of a benchmark scene, the VLM parses the image into a structured 3D layout, which is then evaluated for both intrinsic plausibility and ground-truth consistency. We remove final physical simulation to exclude its effect and evaluate the VLM's spatial reasoning alone.

Metric Suite. We report five metrics: *OoB*↓ and *Float*↓, computed on bounding boxes rather than meshes as in the scene synthesis metrics above; *IoU-3D*↑ / *IoU-2D*↑, the average 3D / top-view

Table 1: Quantitative comparison on the SpatialFactory Benchmark for text-to-scene synthesis. All methods use gemini-3.1-pro as the underlying MLLM.

	Subjective Scores				Objective Metrics						
Method	Func.↑	Spat.↑	Circ.↑	Aest.↑	OR↑	IR↑	OoB↓	Col↓	Nav↑	DWC↑	Float↓
I-Design Çelen et al. (2024)	3.29	4.06	3.00	4.49	0.2497	0.1346	**0.0000**	0.0381	**1.0000**	/	0.0544
Imaginarium Zhu et al. (2025)	3.53	4.88	4.70	5.02	0.1547	0.1192	0.0687	0.1231	0.9999	0.9373	0.1344
Holodeck Yang et al. (2024)	4.94	4.90	5.08	5.39	0.3162	0.2050	0.0403	**0.0029**	0.9986	0.9271	0.0323
SceneWeaver Yang et al. (2026a)	5.21	5.52	4.53	5.87	0.4914	0.3173	0.0079	0.0943	0.9960	/	0.0456
SceneSmith Pfaff et al. (2026)	5.50	6.50	6.33	7.17	0.4730	0.3159	0.1092	0.0479	0.9557	0.9417	0.1846
SpatialFactory (Ours)	**7.21**	**7.28**	**6.42**	**8.22**	**0.8995**	**0.6698**	**0.0000**	0.0580	0.9817	**0.9420**	**0.0255**

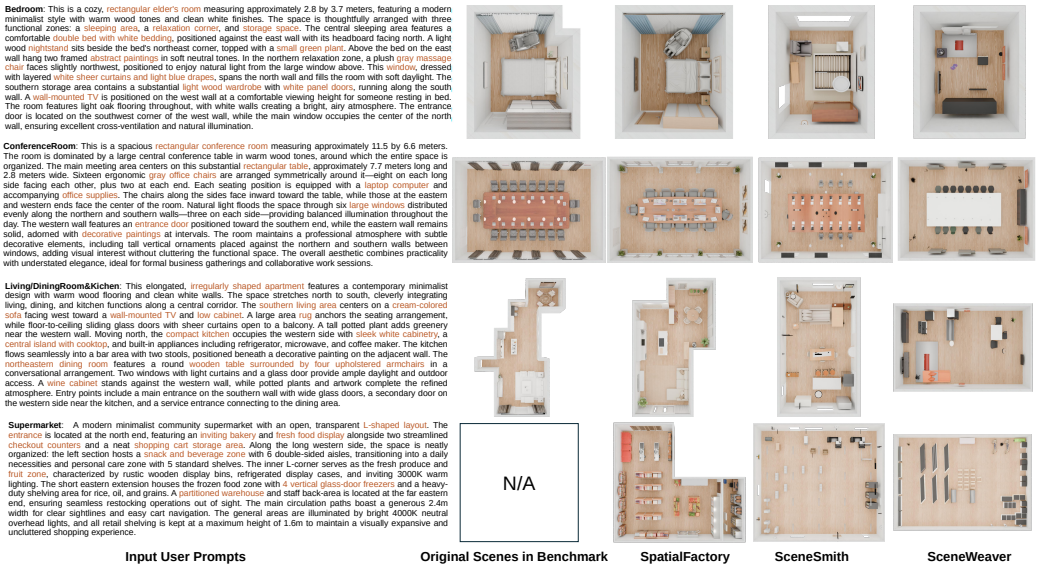


Figure 4: Qualitative comparison on text-to-scene synthesis. Each row uses a fine-grained text prompt that jointly specifies the floorplan, the per-region object layout, and the per-object texture/style. The first three rows use prompts from the SpatialFactory Benchmark; the last is a complex case from outside the benchmark. All methods use gemini-3.1-pro as the underlying MLLM. The *Original Scenes in Benchmark* column shows the benchmark renderings for reference only and is *not* fed to any method as input. As complexity grows, competing methods degrade into disorder, whereas our method maintains a coherent spatial structure and faithful adherence to the prompt.

2D OBB IoU between matched (predicted, GT) pairs; and *Relations-Hit* (**Rel.**↑), the fraction of GT relations correctly recovered. Further evaluation and metric details are provided in Appendix B.2.

4.2 Scene Synthesis Evaluation

We compare SpatialFactory with five representative baseline methods. *I-Design* Çelen et al. (2024) uses an agentic framework to first output a scene graph and then solve the layout via optimization on it. *Imaginarium* Zhu et al. (2025) is an image-guided scene parsing method that generates an image and uses a series of specialized scene understanding models to obtain a view-based scene layout. *Holodeck* Yang et al. (2024) is an LLM-driven framework that writes constraint-based scene programs. *SceneWeaver* Yang et al. (2026a) is an agentic framework that uses a RAG-like initialization followed by ReAct-based adjustment. *SceneSmith* Pfaff et al. (2026) is an agentic generation pipeline that decomposes scene synthesis into a hierarchical process, representing one of the most effective prior approaches to scene synthesis.

Table 1 reports the quantitative comparison between SpatialFactory and the baselines on the Scene Synthesis Evaluation of the SpatialFactory Benchmark. Our method achieves the best results on *all* four subjective scores and attains the best or highly competitive numbers on most objective metrics. Most notably, SpatialFactory nearly doubles the strongest baseline on both OR and IR, the two

Table 2: Evaluation across advanced VLMs.

Method	OoB↓	Float↓	IoU-3D↑	IoU-2D↑	Rel.↑
GT	0.0237	0.0067	1.0000	1.0000	1.0000
gpt-5.2	0.5963	0.1582	0.1389	0.1948	0.4256
claude-sonnet-4-6	0.0661	**0.0074**	0.2339	0.3359	0.5297
kimi-k2.5	0.1437	0.0619	0.2047	0.3011	0.3836
qwen3.5-plus	0.1195	0.0416	0.2747	0.4033	0.4775
doubao-seed-2.0-pro	0.0857	0.0502	0.3438	0.5076	0.5423
gemini-3.1-pro	0.0519	0.0246	0.4456	0.6015	0.6703
gpt-6-astra	**0.0000**	0.0103	**0.6383**	**0.7512**	**0.7564**

Table 3: Ablation and spatial understanding comparison.

Method	OoB↓	Float↓	IoU-3D↑	IoU-2D↑	Rel.↑
gemini-3.1-pro	0.0519	0.0246	0.4456	0.6015	0.6703
w/ specialist models	0.0451	0.0220	0.4428	0.5743	0.6251
w/o 3D Lifting	0.0453	0.0490	0.2463	0.3452	0.4489
Direct	0.0887	0.0391	0.2145	0.3068	0.3877
gpt-6-astra	**0.0000**	**0.0103**	**0.6383**	**0.7512**	0.7564
w/ specialist models	**0.0000**	0.0124	0.6355	0.7494	**0.7601**
w/o 3D Lifting	**0.0000**	0.0122	0.6023	0.7265	0.7470
Direct	0.0023	0.0245	0.4341	0.5400	0.6361
Imaginarium Zhu et al. (2025)	0.2946	0.0294	0.3221	0.4297	0.5012
ArtiScene Gu et al. (2025)	0.2483	0.2477	0.2906	0.3740	0.4352

metrics that most directly reflect instruction-following capability. Note that Imaginarium, which relies on a series of specialized scene understanding models, tends to produce suboptimal results in complex scenes, demonstrating the stability of our method. Figure 4 shows qualitative results. As the cases become progressively more complex from top to bottom, SpatialFactory consistently preserves a coherent global layout, while the scenes produced by other agentic methods gradually degrade into disorder. **Additional results and visualizations** are provided in Appendix C.

4.3 Spatial Understanding Evaluation

To probe MLLMs' spatial understanding capability, we benchmark seven VLMs: *gpt-6-astra* OpenAI (2026b), *gemini-3.1-pro* Google (2026), *claude-sonnet-4-6* Anthropic (2026), *gpt-5.2* OpenAI (2026a), *qwen3.5-plus* Qwen (2026), *kimi-k2.5* Team et al. (2026) and *doubao-seed-2.0-pro* ByteDance (2026). We conduct ablations with gemini-3.1-pro: *w/ specialist models* initializes 3D boxes from 2D detections using SAM3 Carion et al. (2025) masks, MoGe2 Wang et al. (2026b) depth registration and a point map; *w/o 3D Lifting* replaces the lifting tool with MLLM-predicted 3D boxes from 2D detections; and *Direct* jointly predicts objects and initial 3D boxes from the image without separate 2D detection or lifting. All three retain downstream spatial reasoning. We additionally evaluate *gpt-6-astra* under the same settings. Finally, we compare with Imaginarium Zhu et al. (2025) and ArtiScene Gu et al. (2025), two scene synthesis methods built on parsing generated images through conventional pipelines (with specialized scene understanding models).

Table 2 reports advanced VLM performance on our Spatial Understanding Evaluation pipeline. Among them, *gpt-6-astra* achieves the strongest overall results and exhibits a preliminary form of spatial generalization (*e.g.*, handling generalized detection prompts such as "a table-and-chair combination" or "the door on the right side of the room", as well as other generalized understanding behaviors that specialist detectors such as Grounding DINO Liu et al. (2024) cannot achieve).

In Table 3, *w/ specialist models* yields no consistent overall gain for either MLLM, showing that our method can achieve promising results without relying on specialist scene understanding models. Removing 3D Lifting degrades 3D and 2D layout accuracy for both MLLMs, while the joint-prediction *Direct* setting remains weaker than our staged pipeline. Scene parsing comparison with Imaginarium and ArtiScene shows that our agentic framework with 3D lifting substantially outperforms conventional specialist scene-understanding pipelines.

5 Conclusion

We present **SpatialFactory**, an interactive agentic framework that generalizes 3D scene synthesis to fine-grained, free-form instructions and complex non-rectangular floorplans, together with the **SpatialFactory Benchmark**: 111 crafted scenes annotated with structured layouts and 3D geometry, plus a comprehensive evaluation pipeline for scene synthesis and spatial understanding. Experiments show that SpatialFactory exploits MLLMs' spatial generation and understanding to bridge rich 2D priors with 3D synthesis without the need for specialized scene understanding models, mitigating the poor generalization caused by the scarcity of 3D layout data and revealing the potential of MLLMs' spatial intelligence for 3D generation and understanding. **Limitations**, **future work**, and additional **implementation details** are provided in Appendices D and A.1.

AI USE STATEMENT

Generative AI tools were used to polish manuscript text drafted by the authors, assist with migration between conference LaTeX templates and formatting checks, and support the implementation of selected code components according to detailed specifications designed by the authors. The authors conceived the research methodology and experimental protocols, conducted the experiments, reviewed AI-assisted changes, and take responsibility for the final content of this work.

REFERENCES

Anthropic. System card: Claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, 2026.

Tongyuan Bai, Wangyuanfan Bai, Dong Chen, Tieru Wu, Manyi Li, and Rui Ma. Freescene: Mixed graph diffusion for 3d scene synthesis from free prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5893–5903, 2025.

Black Forest Labs. Flux.1 [dev]. https://github.com/black-forest-labs/flux, 2024.

ByteDance. Seed 2.0 pro. https://seed.bytedance.com/zh/seed2, 2026.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, et al. Sam 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719*, 2025.

Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. I-design: Personalized llm interior designer. In *European Conference on Computer Vision*, pp. 217–234. Springer, 2024.

Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, et al. Sam 3d: 3dfy anything in images. *arXiv preprint arXiv:2511.16624*, 2025.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13142–13153, 2023.

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

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.

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

Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *International Journal of Computer Vision*, 129 (12):3313–3337, 2021b.

Lin Gao, Jia-Mu Sun, Kaichun Mo, Yu-Kun Lai, Leonidas J Guibas, and Jie Yang. Scenehgn: Hierarchical graph networks for 3d indoor scene generation with fine-grained geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7):8902–8919, 2023.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Google. Nano banana pro. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Image-Model-Card.pdf, 2025.

Google. Gemini 3.1 pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026.

Zeqi Gu, Yin Cui, Zhaoshuo Li, Fangyin Wei, Yunhao Ge, Jinwei Gu, Ming-Yu Liu, Abe Davis, and Yifan Ding. Artiscene: Language-driven artistic 3d scene generation through image intermediary. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 2891–2901, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Z Sadeghipour Kermani, Zicheng Liao, Ping Tan, and Hao Zhang. Learning 3d scene synthesis from annotated rgb-d images. In *Computer Graphics Forum*, volume 35, pp. 197–206. Wiley Online Library, 2016.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

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.

Chenguo Lin and Yadong Mu. Instructscene: Instruction-driven 3d indoor scene synthesis with semantic graph prior. *arXiv preprint arXiv:2402.04717*, 2024.

Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Zhaoshuo Li. Scenethesis: A language and vision agentic framework for 3d scene generation. *arXiv preprint arXiv:2505.02836*, 2025.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European conference on computer vision*, pp. 38–55. Springer, 2024.

Tianqiang Liu, Siddhartha Chaudhuri, Vladimir G Kim, Qixing Huang, Niloy J Mitra, and Thomas Funkhouser. Creating consistent scene graphs using a probabilistic grammar. *ACM Transactions on Graphics (TOG)*, 33(6):1–12, 2014.

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.

OpenAI. Gpt-image-2.0. https://openai.com/index/introducing-chatgpt-images-2-0/, 2025.

OpenAI. Gpt-5.2. https://openai.com/zh-Hans-CN/index/gpt-5-system-card-update-gpt-5-2/, 2026a.

OpenAI. Gpt-6 astra: A new generation of intelligence. https://openai.com/index/gpt-6-astra/, 2026b.

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.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes. *arXiv preprint arXiv:2602.09153*, 2026.

Hou In Derek Pun, Hou In Ivan Tam, Austin T Wang, Xiaoliang Huo, Angel X Chang, and Manolis Savva. Hsm: Hierarchical scene motifs for multi-scale indoor scene generation. *arXiv preprint arXiv:2503.16848*, 2025.

Qwen. Qwen 3.5. https://qwen.ai/blog?id=qwen3.5, 2026.

Daniel Ritchie, Kai Wang, and Yu-an Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6182–6190, 2019.

Team Seedream, Yunpeng Chen, Yu Gao, Lixue Gong, Meng Guo, Qiushan Guo, Zhiyao Guo, Xiaoxia Hou, Weilin Huang, Yixuan Huang, et al. Seedream 4.0: Toward next-generation multimodal image generation. *arXiv preprint arXiv:2509.20427*, 2025.

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 *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 29469–29478, 2025.

Hou In Ivan Tam, Hou In Derek Pun, Austin T Wang, Angel X Chang, and Manolis Savva. Sceneeval: Evaluating semantic coherence in text-conditioned 3d indoor scene synthesis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 7355–7365, 2026.

Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 20507–20518, 2024.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Tencent Hunyuan. Hunyuan3d-3.0. https://3d.hunyuan.tencent.com/, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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.

Ling Wang, Hao-Xiang Guo, Xinzhou Wang, Fuchun Sun, Kai Sun, Pengkun Liu, Hang Xiao, Zhong Wang, Guangyuan Fu, Eric Li, et al. Scenetransporter: Optimal transport-guided compositional latent diffusion for single-image structured 3d scene generation. *arXiv preprint arXiv:2602.22785*, 2026a.

Ruicheng Wang, Sicheng Xu, Yue Dong, Yu Deng, Jianfeng Xiang, Zelong Lv, Guangzhong Sun, Xin Tong, and Jiaolong Yang. Moge-2: Accurate monocular geometry with metric scale and sharp details. *Advances in Neural Information Processing Systems*, 38:35928–35959, 2026b.

Zehan Wang, Ziang Zhang, Tianyu Pang, Chao Du, Hengshuang Zhao, and Zhou Zhao. Orient anything: Learning robust object orientation estimation from rendering 3d models. *arXiv preprint arXiv:2412.18605*, 2024.

Zehan Wang, Ziang Zhang, Jiayang Xu, Jialei Wang, Tianyu Pang, Chao Du, Hengshuang Zhao, and Zhou Zhao. Orient anything v2: Unifying orientation and rotation understanding. *arXiv preprint arXiv:2601.05573*, 2026c.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Ken Xu, James Stewart, and Eugene Fiume. Constraint-based automatic placement for scene composition. In *Graphics Interface*, volume 2, pp. 25–34, 2002.

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *Advances in neural information processing systems*, 38:140319–140351, 2026a.

Yixuan Yang, Zhen Luo, Tongsheng Ding, Junru Lu, Mingqi Gao, Jinyu Yang, Victor Sanchez, and Feng Zheng. Optiscene: Llm-driven indoor scene layout generation via scaled human-aligned data synthesis and multi-stage preference optimization. *Advances in Neural Information Processing Systems*, 38:42499–42529, 2026b.

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

Jin Yao, Radowan Mahmud Redoy, Sebastian Elbaum, Matthew Dwyer, and Zezhou Cheng. Labelany3d: Label any object 3d in the wild. *Advances in neural information processing systems*, 38:133188–133216, 2026.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Huangyue Yu, Baoxiong Jia, Yixin Chen, Yandan Yang, Puhao Li, Rongpeng Su, Jiaxin Li, Qing Li, Wei Liang, Song-Chun Zhu, et al. Metascenes: Towards automated replica creation for real-world 3d scans. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1667–1679, 2025.

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 Trans. Graph.*, 30 (4):86, 2011.

Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. Deep generative modeling for scene synthesis via hybrid representations. *ACM Transactions on Graphics (TOG)*, 39(2):1–21, 2020.

Shengchao Zhou, Jiehong Lin, Jiahui Liu, Shizhen Zhao, Chirui Chang, and Xiaojuan Qi. Assist-3d: Adapted scene synthesis for class-agnostic 3d instance segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 13764–13772, 2026.

Xiaoming Zhu, Xu Huang, Qinghongbing Xie, Zhi Deng, Junsheng Yu, Yirui Guan, Zhongyuan Liu, Lin Zhu, Qijun Zhao, Ligang Liu, et al. Imaginarium: Vision-guided high-quality 3d scene layout generation. *ACM Transactions on Graphics (TOG)*, 44(6):1–24, 2025.

APPENDIX

APPENDIX CONTENTS

A SpatialFactory Details
A.1 Implementation
A.2 Coordinate System
A.3 3D Lifting Details
A.4 Composite Object Generation
A.5 Baseline Configuration

B SpatialFactory Benchmark Details
B.1 Scene Categories and Statistics
B.2 Evaluation Setup Details
B.3 Benchmark Gallery

C Additional Results and Visualization
C.1 Additional Results with GPT-6 Astra
C.2 Per-Category Results
C.3 Effect of Sub-Agents
C.4 User Study
C.5 Failure Cases
C.6 Runtime and Token Usage

D Limitations and Future Work

E Prompts
E.1 Prompt for MLLM Judge
E.2 Prompt for SpatialFactory

A SPATIALFACTORY DETAILS

This section provides implementation details, the coordinate system, detailed 3D lifting formulas, composite object generation, and baseline configurations.

A.1 IMPLEMENTATION

Default model stack. By default, SpatialFactory uses gemini-3.1-pro Google (2026) as the underlying MLLM for every sub-agent, gemini-3-pro-image-preview Google (2025) as the Multimodal Generation LM that synthesizes the top-view scene image, and Hunyuan-3D-3.0-Pro Tencent Hunyuan (2025) or SAM3D Chen et al. (2025) as the image-to-3D back-end.

Execution robustness. We use three complementary mechanisms to handle imperfect MLLM outputs. *(i) Bounded retries:* failed requests and outputs that do not pass validation are retried within configured budgets, while generation/filtering and ReAct refinement loops have iteration limits. *(ii) Schema-constrained output:* for structured MLLM calls, JSON Schema specifies required fields, types, and array shapes; returned values are checked again before downstream use. *(iii) Tolerant parsing:* where applicable, a JSON extractor handles fenced or incompletely fenced responses, raw arrays, and trailing commas without bypassing task-specific validation. With these mechanisms, we observed a 100% execution success rate in our experiments, excluding failures caused by external network connectivity.

Configuration. Since current MLLMs typically scale detection bounding-box pixel coordinates to the range $[0, 1000]$ during object detection, all rendered and generated images are resized to 1000×1000 accordingly, and each 2D box is stored as $[x_1, y_1, x_2, y_2]$. Object orientations are discretized to multiples of 15°. The default pipeline does not invoke specialized scene understanding models. All experiments are conducted on a single NVIDIA H20 GPU.

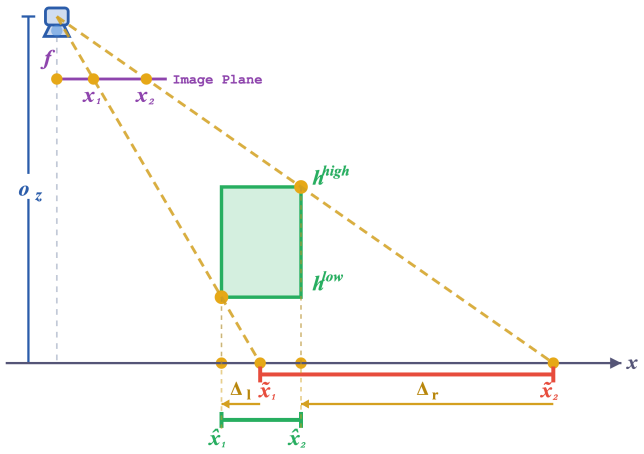


Figure A1: Illustration of 3D lifting along the x axis: the metric-scale region back-projected from the 2D detection box (red) is corrected to the x-axis extent of the coarse 3D bounding box (green).

A.2 Coordinate System

Image coordinate system. All computation, storage and MLLM interaction throughout SpatialFactory are conducted in a unified *image coordinate system*, in which the X axis points east (right), the Y axis points south (down), and the Z axis points upward. Under this convention we maintain two parallel spaces: a *pixel space* in which the rendered top-view image and every detection box live, with $x, y \in [0, 1000]$, and a *real space* that corresponds to the actual ground plane onto which the top-view image projects, in metric units. The two spaces share the same axis orientation and differ only by a single scalar — the pixel-to-metric ratio

$$\rho = \frac{c_z \cdot \tan(\mathrm{fov}_y/2)}{500}, \tag{3}$$

where c_z is the height of the top-view camera and fov_y is its vertical field of view, so that one pixel corresponds to ρ meters. To eliminate any additional offset between the two spaces, we translate the floorplan vertices and the camera position so that the camera center lands at $(500\rho,\ 500\rho,\ c_z)$ in the real space. After this translation the two origins coincide at the top-left of the top-view image, and the conversion between the two spaces reduces to a pure scaling: $x_{\text{real}} = \rho \cdot x_{\text{pixel}}$ and $y_{\text{real}} = \rho \cdot y_{\text{pixel}}$.

A.3 3D Lifting Details

Per-edge correction in 3D Lifting. In the 3D Lifting tool, we apply four signed corrections $\{\Delta_\mathrm{l}, \Delta_\mathrm{r}, \Delta_\mathrm{t}, \Delta_\mathrm{b}\}$ to the metric-scale region $[\tilde{x}_1, \tilde{y}_1, \tilde{x}_2, \tilde{y}_2]$ to obtain the x and y extents $[\hat{x}_1, \hat{y}_1, \hat{x}_2, \hat{y}_2]$ of the coarse 3D bounding box. Each correction $\Delta_\bullet$ ($\bullet \in \{\mathrm{l}, \mathrm{r}, \mathrm{t}, \mathrm{b}\}$) is computed as

$$\Delta_\bullet = \frac{h(d_\bullet)}{o_z}\, d_\bullet, \quad h(d) = \begin{cases} h_i^{\text{high}}, & d > 0, \\ h_i^{\text{low}}, & d \le 0, \end{cases} \tag{4}$$

where o_z is the camera height and $d_\bullet$ is the signed distance from the camera principal point to the corresponding edge of $[\tilde{x}_1, \tilde{y}_1, \tilde{x}_2, \tilde{y}_2]$, taken positive in the outward direction. The 3D center and vertical extent then follow as

$$\mathbf{p}_i = \left(\tfrac{\hat{x}_1+\hat{x}_2}{2},\ \tfrac{\hat{y}_1+\hat{y}_2}{2},\ \tfrac{h_i^{\text{low}}+h_i^{\text{high}}}{2}\right), \quad c_i = h_i^{\text{high}} - h_i^{\text{low}}. \tag{5}$$

Linear interpolation of (a_i, b_i) in 3D Lifting. The orientation-aligned in-plane extents (a_i, b_i) of the coarse 3D bounding box are obtained from the corrected x and y extents through a simple linear interpolation. Let $w_x = \hat{x}_2 - \hat{x}_1$, $w_y = \hat{y}_2 - \hat{y}_1$ be the corrected x and y extents, and let $n = \lfloor \theta_i / 90^\circ \rfloor$, $r = \theta_i \bmod 90^\circ$. Then

$$(a_i, b_i) = \begin{cases} \left(\frac{90^\circ - r}{90^\circ} w_x + \frac{r}{90^\circ} w_y,\ \frac{90^\circ - r}{90^\circ} w_y + \frac{r}{90^\circ} w_x\right), & n \text{ even}, \\ \left(\frac{90^\circ - r}{90^\circ} w_y + \frac{r}{90^\circ} w_x,\ \frac{90^\circ - r}{90^\circ} w_x + \frac{r}{90^\circ} w_y\right), & n \text{ odd}. \end{cases} \tag{6}$$

The two cases differ in which axis is dominant: when n is even the orientation is closer to the y axis, and a_i is dominated by w_x; when n is odd the situation is reversed.

A.4 Composite Object Generation

Real scenes often contain dense clutter, making fine-grained compositionality difficult to reconcile with generation speed and robustness. SpatialFactory therefore adopts a practical trade-off between the granularity of independently generated objects and the reliability of scene synthesis. In the *regular mode*, it detects and generates visually separable, semantically meaningful objects individually, then uses their scene-graph relations to recover a plausible arrangement.

For densely arranged objects, the *group mode* instead generates the collection as a single asset, such as a table with several objects placed on it. This avoids the cost and placement errors of generating every item separately while preserving the intended overall appearance. The trade-off is that objects within a grouped asset are not independently editable in the resulting scene. To address this, we provide a post-processing pipeline: an MLLM detects and localizes the constituent objects in the asset-generation image with 2D bounding boxes, SAM 3 Carion et al. (2025) segments their masks, and SAM3D Chen et al. (2025) generates separate 3D assets. These assets then replace the grouped asset in the compositional scene representation.

A.5 Baseline Configuration

Text-to-scene synthesis. Among the five baselines compared in Sec. 4.2 of the main paper, Holodeck, I-Design and SceneWeaver retrieve objects from publicly available asset collections such as Objaverse Deitke et al. (2023), while SceneSmith uses an asset generation model to generate objects. Note that the adapted layout-retrieval subset of MetaScenes Yu et al. (2025) constructed by SceneWeaver is not publicly released. We therefore remove its retrieval initialization to evaluate a ReAct-based agent without RAG-like initialization or visual-spatial image planning. Imaginarium Zhu et al. (2025) is evaluated using its image-based generation-and-parsing pipeline and the same benchmark prompts. For a controlled comparison, all methods use gemini-3.1-pro Google (2026) as the common underlying MLLM, a model that has been publicly available for some time and exhibits meaningful spatial intelligence. We also disable automatic scene-texture synthesis in SpatialFactory and render all methods with the same scene textures to align the visual conditions with prior methods.

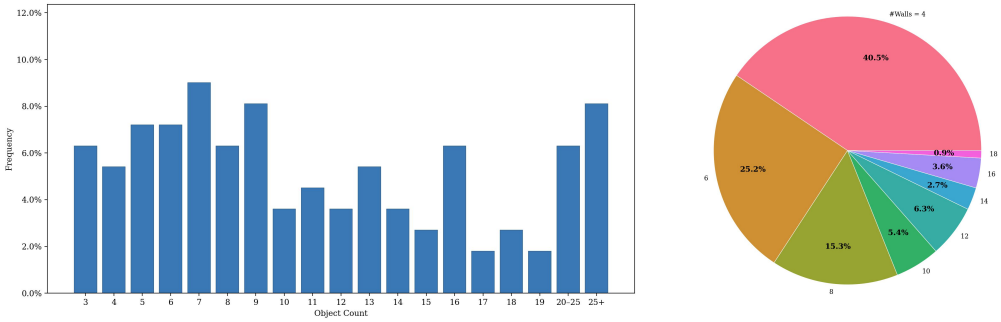
Spatial Understanding Evaluation. We disable the Environment Builder within the Spatial Reasoner, including its architectural completion and texture generation, since these steps are outside the scope of spatial understanding. Instead, this evaluation focuses on recovering 3D bounding boxes aligned with the floorplan's metric world coordinates from top-view images. Our default pipeline first detects objects in 2D and then lifts their boxes into 3D.

The three variants differ in how they obtain the initial 3D layout. In *w/ specialist models*, we first detect objects in 2D and segment them with SAM 3 Carion et al. (2025). MoGe2 Wang et al. (2026b) provides depth estimation and registration, from which we compute a point map and derive an initial 3D box for each segmented object; spatial reasoning then refines the boxes. In *w/o 3D Lifting*, the staged 2D parsing is retained, but the MLLM infers 3D boxes directly from the 2D boxes under the specified camera and world-coordinate conventions, without the lifting tool. In *Direct*, the MLLM jointly identifies objects and predicts their initial 3D boxes from the image, bypassing separate 2D detection and lifting; spatial reasoning remains in place to refine the resulting layout.

For Imaginarium Zhu et al. (2025) and ArtiScene Gu et al. (2025), we adapt their conventional image-parsing pipelines so that they output a 3D bounding box for every parsed scene element. For Imaginarium, when its object-retrieval stage fails to retrieve a required asset and thus cannot optimize its placement, we use an additional 3D generation model as a fallback to obtain the object geometry for placement optimization. The predicted boxes are then converted to our compositional scene representation and evaluated by the same SpatialFactory Benchmark metrics as our method.

Table B1: Average statistics of the SpatialFactory Benchmark by scene category. Columns report the per-category scene count, average floor area, average number of objects (Obj.s), doors and windows (DW.s), walls, and words in the scene description.

Scene	#Count	#Area (m^2)	#Obj.s	#DW.s	#Walls	#Words
Bedroom	10	14.37	9.10	2.20	6.00	201.20
LivingRoom	9	36.70	14.11	6.67	13.33	208.11
Balcony	9	7.87	5.78	3.56	4.78	191.00
Bathroom	9	4.44	4.78	1.78	6.00	182.44
BeautySalon	8	37.38	20.13	3.25	7.75	193.38
Classroom	10	21.69	16.40	3.10	6.20	198.60
ConferenceRoom	9	37.09	20.22	3.78	8.11	199.67
DiningRoom	10	18.40	11.20	4.60	7.20	203.80
Kitchen	9	5.07	10.22	2.22	6.67	194.11
Office	9	50.06	22.67	5.11	6.89	209.56
TeaRoom	9	20.32	8.56	2.78	4.22	193.00
WalkInCloset	10	6.88	7.00	1.90	5.60	188.20



(a) Object Count Distribution.

(b) Wall Count Distribution.

Figure B1: Distribution of object counts and wall counts across all 111 scenes in the SpatialFactory Benchmark.

B SpatialFactory Benchmark Details

B.1 Scene Categories and Statistics

The SpatialFactory Benchmark contains 111 scenes covering 12 indoor categories. Tab. B1 reports per-category averages for the floor area, number of objects, number of doors and windows, number of walls, and the length of the natural-language scene description. The corresponding distributions of object counts and wall counts across the entire benchmark are visualized in Fig. B1a and Fig. B1b, respectively.

B.2 Evaluation Setup Details

Subjective Scores — Full Rubrics. We rate every generated scene with an MLLM judge along four axes on a 0–10 scale. *Functional Integrity* (**Func.**) examines how effectively the layout serves the intended room purpose and whether any essential furniture or stated user requirement is missing or contradicted. *Spatial Logic & Physics* (**Spat.**) examines whether objects form sensible groupings (*e.g.*, nightstands beside beds), face the correct direction, fit within the room boundary and are free of clipping, floating or scale errors. *Circulation & Flow* (**Circ.**) examines movement pathways,

Table B2: Spatial relationships supported by our Compositional Scene Representation, grouped into object-layout, vertical object-object and horizontal object-object relations.

Type	Relation	Target
object-layout	placed on	floor
	hanging from	ceiling
	mounted on	wall
	placed against	wall
	placed with side against	wall
	installed between	wall1, wall2
	near	door / window
object-object (vertical)	placed on / support	object
	inside / contain	object
	above / below	object
object-object (horizontal)	next to	object
	{direction} near	object
	far from	object

clearance around interactable objects and the absence of obstructions or excessive dead zones. *Aesthetic Coherence* (**Aest.**) examines visual balance, proportion and stylistic harmony, and flags outlier pieces that disrupt the requested design language. The full judge prompts are listed in the prompt section of this appendix.

Objective Metrics — Full Definitions. *Instruction Recall* (**IR**↑) decomposes the user prompt into fine-grained atomic instructions and uses an MLLM to judge one by one whether each atomic instruction is realized in the synthesized scene, yielding a recall ratio over all atomic instructions; this fine-grained, instruction-by-instruction judgment keeps the metric largely free of holistic subjective bias, and notably makes IR a highly effective indicator of controllability. *Out-of-Bound* (**OoB**↓) is the fraction of objects whose mesh falls outside the floorplan polygon (extruded to the wall height). *Collision* (**Col**↓) is the fraction of objects whose mesh collides with at least one other object's mesh. *Space Reachability* (**SR**↑) is the fraction of the floor area that remains connected and reachable for an agent of standard human radius after subtracting all object footprints. *Door/Window Clearance* (**DWC**↑) is the fraction of doors and windows whose mandatory clearance volume in front of the opening is free of object intrusion, ensuring that each opening is functionally accessible. Methods that do not generate doors or windows leave this metric blank, such as SceneWeaver and I-Design. *Floating* (**Float**↓) is the fraction of objects whose mesh is unsupported.

Spatial Understanding — Metric Suite. For the Spatial Understanding Evaluation pipeline (Sec. 4.2 of the main paper), we report: *Out-of-Bound* (**OoB**↓), the fraction of predicted objects that lie outside the floorplan; *Floating* (**Float**↓), the fraction of predicted objects whose declared support surface does not actually support them; *IoU-3D*↑, the average 3D oriented bounding box (3D-OBB) IoU between matched (predicted, GT) pairs; *IoU-2D*↑, the average top-view 2D-OBB IoU between matched pairs (more tolerant to height-estimation noise); and *Relations-Hit* (**Rel.**↑), the proportion of ground-truth spatial relations (*e.g.*, "mounted on *wall_k*", "below *object_k*") that are correctly recovered by the parsed layout. The reference values obtained by feeding the ground-truth layout through the same evaluator (denoted as "GT") serve as the soft upper bound of each metric. The full set of supported spatial relations is listed in Tab. B2.

B.3 Benchmark Gallery

We present additional examples from the SpatialFactory Benchmark in Fig. B2, covering a diverse range of indoor categories. Each scene is shown together with its top-view rendering, structured layout ground-truth, and a representative free-form description.



Figure B2: Gallery of SpatialFactory Benchmark scenes covering all 12 indoor categories. Each scene is shown together with its top-view rendering, structured layout ground-truth, and a representative free-form description.



Figure C1: Additional scenes generated by SpatialFactory on prompts from the SpatialFactory Benchmark. All results use gemini-3.1-pro as the underlying MLLM.



Figure C2: Additional scenes generated by SpatialFactory on prompts *outside* the SpatialFactory Benchmark. All results use gemini-3.1-pro as the underlying MLLM.

C Additional Results and Visualization

We provide additional qualitative results and module visualizations that complement the main paper. Fig. 1 shows representative SpatialFactory results and compositional scene representations. Fig. C1 and Fig. C2 show additional scenes generated with gemini-3.1-pro from in-benchmark and out-of-benchmark prompts, respectively.

Table C1: Per-category results of SpatialFactory on the SpatialFactory Benchmark, reporting both the four subjective MLLM-judge scores (Func., Spat., Circ., Aest.) and the six objective scene-synthesis metrics (OR, IR, Col, SR, DWC, Float).

	Subjective Scores				Objective Metrics					
Scene Categories	Func.↑	Spat.↑	Circ.↑	Aest.↑	OR↑	IR↑	Col↓	SR↑	DWC↑	Float↓
Bedroom	7.8	7.4	6.7	8.7	0.9200	0.7252	0.0181	0.9275	0.9250	0.0000
LivingRoom	8.3	7.5	6.5	8.3	1.0000	0.6471	0.0588	0.8457	0.9062	0.0000
Balcony	7.6	7.3	6.3	8.5	0.9076	0.7749	0.0643	0.9709	0.8814	0.0185
Bathroom	6.0	6.9	7.0	7.0	0.8678	0.5903	0.0357	0.7817	1.0000	0.0277
BeautySalon	6.8	7.2	6.6	8.2	0.8035	0.5759	0.0285	0.9834	1.0000	0.0138
Classroom	7.0	7.3	6.3	8.4	0.8497	0.6500	0.0000	0.8696	1.0000	0.0293
ConferenceRoom	7.1	7.2	6.1	8.3	0.9305	0.6944	0.0357	0.9436	1.0000	0.0000
DiningRoom	6.8	7.3	5.9	8.0	0.9400	0.5670	0.0476	0.9616	1.0000	0.0000
Kitchen	6.7	7.3	6.3	7.7	0.7743	0.7460	0.1750	0.9563	0.8333	0.0111
Office	7.6	7.3	6.7	8.6	0.9253	0.7295	0.0178	0.9663	1.0000	0.0535
TeaRoom	8.0	7.0	6.5	8.5	0.9600	0.7553	0.1625	0.9763	0.9166	0.1251
WalkInCloset	7.0	7.8	6.5	8.0	0.9500	0.7470	0.2666	0.9715	0.8333	0.0277

C.1 Additional Results with GPT-6 Astra

SpatialFactory already produces coherent layouts with gemini-3.1-pro. With gpt-6-astra, we observe more accurate and robust spatial interpretation of top-view plans, consistent with its stronger spatial-understanding results in Sec. 4.3. Its improved Blender Python coding also helps the Environment Builder construct architectural and terrain elements, such as indoor columns and platforms and outdoor roads and varied terrain. Fig. C3 presents additional indoor and outdoor scenes generated with gpt-6-astra.

C.2 Per-Category Results

To complement the aggregated numbers reported in the main paper, Tab. C1 provides a per-category breakdown of SpatialFactory across all 12 scene types in the SpatialFactory Benchmark, under both the subjective MLLM-judge scores and the objective scene-synthesis metrics defined in Sec. B.2. SpatialFactory delivers consistently strong subjective scores across heterogeneous categories, while remaining nearly free of out-of-bound, floating and door/window blocking artifacts on most categories.

C.3 Effect of Sub-Agents

We visualize the effect of two correction modules in SpatialFactory: the *Floorplan Refiner* and the *Asset Corrector*. As shown in Fig. C4, the Floorplan Refiner revises the floorplan to improve alignment with the generated top-view visual plan, while the Asset Corrector iteratively rectifies each placed mesh's yaw and scale to match the top-view reference.

C.4 User Study

To complement the MLLM-judge and objective metrics in the main paper, we conduct a user study with 43 participants on 10 representative scenes spanning multiple room categories. All four methods use gemini-3.1-pro Google (2026) as their underlying MLLM. For each question, the methods are presented anonymously side-by-side in a randomized order, and each participant is asked to rank the four results from best to worst. We report the Top-1% (the proportion of votes that rank a method first) and Top-2% (the proportion of votes that rank a method within the top two), averaged across all questions and participants. As shown in Tab. C2, SpatialFactory is ranked first in 77.78% of all votes and within the top two in 94.02%, substantially outperforming all three baselines.

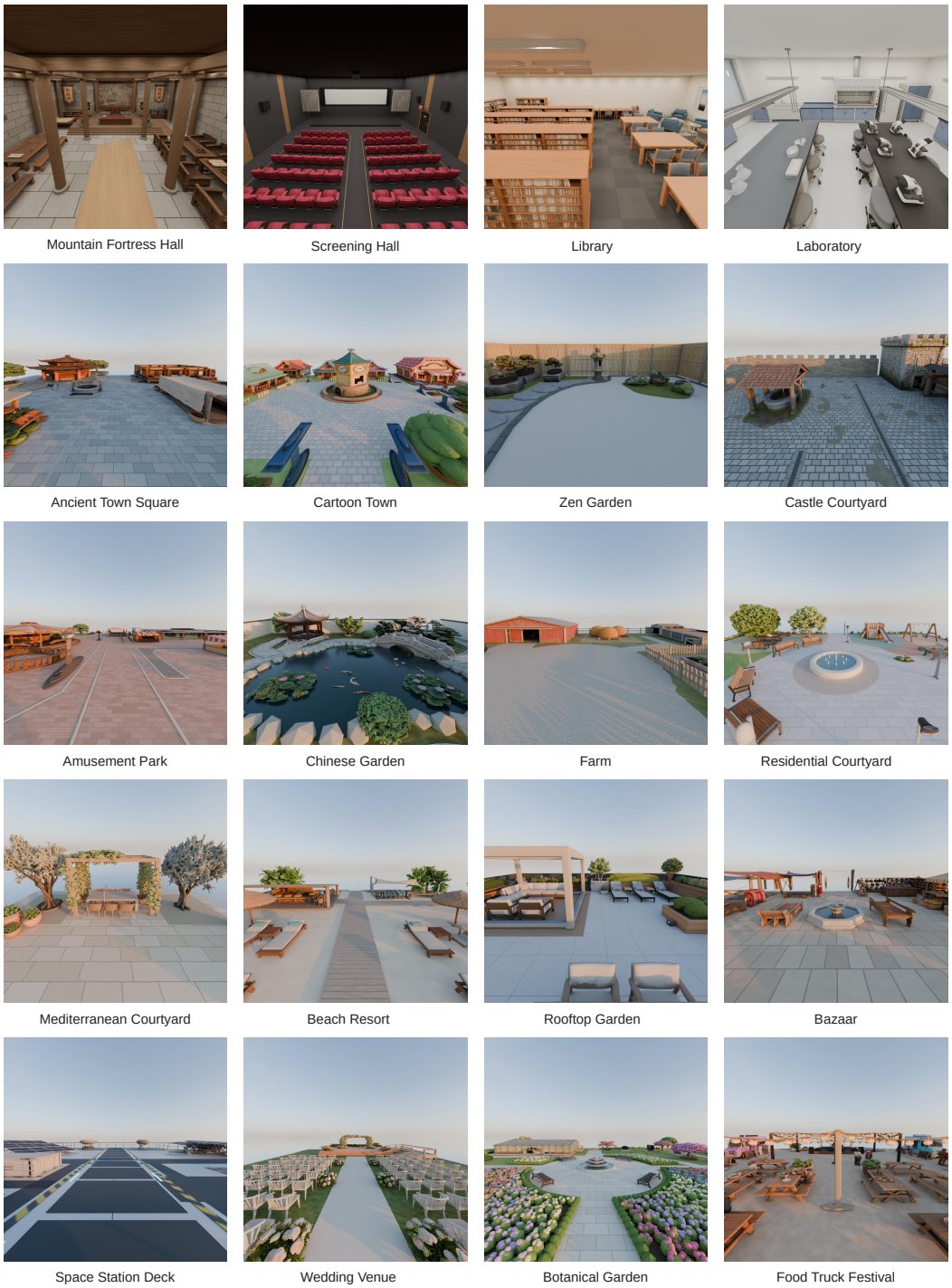


Figure C3: Additional indoor and outdoor scenes generated by SpatialFactory with gpt-6-astra.

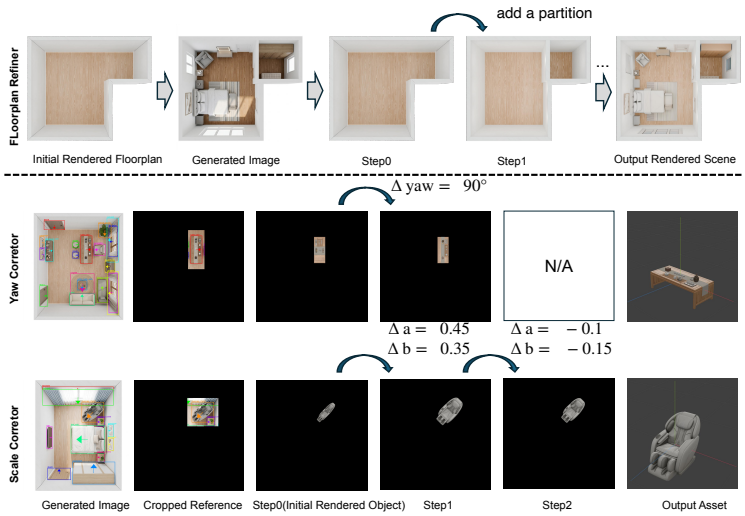


Figure C4: *Top:* the *Floorplan Refiner* edits the floorplan to match the generated top-view image on complex cases. *Bottom:* inside *Asset Corrector*, the *Yaw Corrector* and *Scale Corrector* iteratively rectify each placed mesh's orientation and scale to match the top-view reference.

Table C2: User study results from 43 participants on 10 scenes. All four methods use gemini-3.1-pro as the underlying MLLM. Top1% (Top2%) denotes the proportion of votes that rank the method first (within the top two) across all participants and questions.

Rank \ Method	SpatialFactory	SceneSmith	SceneWeaver	Holodeck
Top1%	**77.78**	15.38	5.13	1.71
Top2%	**94.02**	69.23	29.06	7.69

C.5 Failure Cases

As shown in Fig. C5, failure cases of SpatialFactory typically arise in densely populated scenes containing many objects, where the detector occasionally misses small or heavily occluded instances. In addition, doors that are visually subtle (*e.g.*, blending into the surrounding wall or partially occluded by furniture) can also be overlooked. Generated top-view images may contain inconsistent perspective cues, leading to errors in inferred 3D object placement; however, the Spatial Reasoner corrects most such deviations before final rendering. A major remaining source of visible distortion is the limited quality of current image-to-3D generators for complex meshes: the generated asset may differ substantially in shape from its counterpart in the image. We expect advances in 3D asset generation to alleviate this limitation.

C.6 Runtime and Token Usage

Across model calls, SpatialFactory uses approximately 400,000 input tokens and 20,000 output tokens per scene on average. A scene typically takes 10–20 minutes to generate. In contrast, SceneSmith may require several hours, sometimes approaching 10 hours, to complete one scene, while SceneWeaver typically requires around 2 hours. Together with the scene-quality results in the main paper, these runtimes indicate that SpatialFactory substantially improves both generation quality and speed over prior agentic methods.



Figure C5: Representative failure cases of SpatialFactory, with annotations highlighting the source of the error.

D Limitations and Future Work

SpatialFactory still has several limitations. First, the entire pipeline is built on top of MLLMs and is therefore upper-bounded by their spatial intelligence. Concretely, current Multimodal Generation LMs cannot always faithfully preserve the given floorplan in complex cases and have to rely on subsequent correction, especially under more imaginative or unusual scene designs. This limitation may diminish as their spatial generation capabilities improve. Second, realistic outdoor environments still pose challenges because of irregular terrain and complex road systems. The stronger Blender Python coding ability of gpt-6-astra partially mitigates this limitation, enabling more effective construction of outdoor structures, but highly complex landscapes remain challenging. Third, fine-scale geometry and appearance remain a major bottleneck: trees, lattice-like structures, and highly detailed textures cannot yet be reproduced reliably either through Blender Python code or by current image-to-3D asset generators, limiting the visual fidelity of the final scene. In this work, we decouple separable scene objects from architectural and terrain elements: the former are generated as individual meshes, while the latter and their materials are constructed through Blender Python code. We believe this division of labor is promising. Future improvements in mask-conditioned image-to-3D generation, such as more capable successors to SAM3D Chen et al. (2025), could recover object meshes from image regions at higher fidelity and help produce more realistic, editable, and interactive 3D scenes.

E Prompts

We provide representative prompts used in SpatialFactory.

E.1 Prompt for MLLM Judge

Prompt for MLLM judge

You are an expert in interior design and human-centric spatial planning. Your task is to evaluate the quality of the room layout based on provided top-down rendering and user preference (in triple backquotes).

User Preferences

```{user_preference}```

Please provide a detailed critique across the following four dimensions, assigning a score (0–10) and a concise justification for each.

Evaluation Criteria

1. *Functional Integrity*
   - Assess how effectively the layout serves its intended purpose (*e.g.*, bedroom, office)
   - Identify missing essentials, functional contradictions, or deviations from stated user needs
2. *Spatial Logic & Physics*
   - Ensure the objects are arranged in a logical, practical way within the room boundaries
   - Check for logical groupings (*e.g.*, nightstands by beds, desks and chairs), proper orientations (*e.g.*, TV is visible from seating areas)
   - Check for "physics" errors like furniture clipping, floating assets, or scale inaccuracies
3. *Circulation & Flow*
   - Analyze movement pathways and clearance. Identify obstructions, cluttered corridors
   - Identify excessive "dead zones" (empty space ¿ 80%). Note: Functionality and Completeness should be prioritized, empty rooms should not receive high scores for circulation and flow
4. *Aesthetic Coherence*
   - Evaluate visual balance, proportion, and stylistic harmony
   - Identify "outlier" pieces that disrupt the design language or conflict with the user's requested style

Scoring Guidelines

1. Numerical Score: Integers from 0 (completely inconsistent) to 10 (perfectly aligned).
2. Justifications: 1–3 concise sentences per aspect.
3. Output Format: Return your evaluation strictly in the following JSON format. Ensure each aspect has its own dedicated score and comment:

```json
{
  "functional_integrity": {
    "score": int,
    "justification": "Reason for Functional Integrity..."
  },
  "spatial_logic_and_physics": {
    "score": int,
    "justification": "Reason for Spatial Logic & Physics..."
  },
  "circulation_and_flow": {
    "score": int,
    "justification": "Reason for Circulation & Flow..."
  },
  "aesthetic_coherence": {
    "score": int,
    "justification": "Reason for Aesthetic Coherence..."
  },
  "overall_average_score": int,
  "holistic_summary": "A brief concluding thought on the layout
  ↪  quality."
}
```
```

E.2 PROMPT FOR SPATIALFACTORY

Prompt for Spatial Composer

You are the **Spatial Composer**, the top-level agent in SpatialFactory. Your role is to orchestrate an end-to-end scene synthesis process that transforms a user instruction, together with optional reference images, into a compositional 3D scene. The process consists of two cooperating modules: **Visual-spatial Imagination**, which externalizes the user intent into a floorplan-aligned top-view visual plan, and **Compositional 3D Realization**, which parses, lifts, generates assets for, and spatially refines the scene.
Mode. The system supports two modes.

- **Automatic mode:** execute all modules end-to-end without pausing for user confirmation, unless the input contains no scene-design intent.
- **Interactive mode:** collaborate with the user at key decision points. Before floorplan design, clarify requirements such as room shape, dimensions, style, functional zones, key furniture and layout constraints. Let the user revise the floorplan and visual plan before scene parsing; after reconstruction, route object-level edit requests to the independent Scene Editor and show updated renderings.

Coordinate system. All spatial descriptions use the image coordinate system adopted by SpatialFactory: the X axis points east/right, the Y axis points south/down, and the Z axis points upward. In the top-view image, north corresponds to $-Y$, south to $+Y$, west to $-X$, and east to $+X$. Coordinates are expressed in meters unless otherwise specified, and pixel-space coordinates are normalized to the $[0, 1000]$ image range.
Visual-spatial Imagination. First interpret the user instruction and optional reference images. If the user provides a floorplan image, parse it into a structured floorplan JSON. If the user describes the floorplan through text, convert the description into the same JSON format. If the user provides only a coarse scene type, infer a reasonable floorplan and elaborate the in-room contents according to the requested function and style.
The floorplan JSON should follow the format:

```
{
  "scene_name": "Style-SceneName",
  "environment_type": "indoor",
  "description": "Detailed scene description...",
  "floor_plan": {
    "vertices": [[x1, y1], [x2, y2], ...],
    "partition": [[xs, ys, xe, ye], ...],
    "height": 3.8,
    "shape": "rectangle"
  }
}
```

The `environment_type` field distinguishes indoor and outdoor scenes; `vertices` defines the closed floorplan polygon, `partition` stores internal wall segments, `height` stores the wall height, and `description` specifies functional zones, furniture, layout relationships and style. After the floorplan JSON is ready, invoke the **Floorplan Designer** to render the empty top-view image and align the image coordinate space with the metric world space.
Then invoke the **Plan Generator** to synthesize a furnished top-view visual plan conditioned on the empty top-view image and the scene description. The generated visual plan should remain floorplan-aligned with the rendered empty room. The **Plan Filter** checks whether the generated visual plan preserves the given floorplan; when the check fails, ask the Plan Generator to regenerate. Once a coarsely aligned pair is obtained, the **Floorplan Refiner** may edit the floorplan JSON and re-render the empty image until the empty-room top view and the generated visual plan are wall-by-wall aligned.
Compositional 3D Realization. After Visual-spatial Imagination produces $(\mathcal{F}, I_{\text{empty}}, I_{\text{scene}})$, invoke the **Scene Parser** to parse the top-view visual plan into structured 2D scene understanding results. The parser detects newly added objects with open-vocabulary labels, captions and 2D bounding boxes in the form $[x_1, y_1, x_2, y_2]$, extracts scene graphs that describe supporting, attaching and wall-side relations, predicts discretized object orientations, and estimates object height ranges.
Next invoke the **3D Lifting** tool to convert the parsed 2D layout into a coarse 3D layout. The lifting step should jointly consider the 2D bounding boxes, scene graphs, object orientation, support relations, height ranges and common-sense priors. Then invoke the **Asset Generation** tool to crop each object, generate or retrieve a textured mesh, canonicalize the asset, and attach the resulting asset identifier to the scene JSON. For composite objects, use regular mode when individual components should be separately parsed and placed, and group mode when a dense object cluster is better represented as a single generated asset.
Finally invoke the **Spatial Reasoner** to initialize a compositional spatial memory and refine the scene through visual feedback. Its Asset Corrector aligns generated meshes with their image regions; its layout-

level reasoning repairs out-of-bound objects, collisions, implausible sizes and broken support relations while prioritizing the actual 2D floorplan over distortions in the generated image. When enabled, the Environment Builder plans missing architectural or terrain elements and scene materials, constructs them with Blender Python code, and reviews top-down and custom renderings before accepting the scene.
Interactive editing. In interactive mode, floorplan and visual-plan edits must be completed before scene parsing; the generated image is then frozen for downstream stages. After reconstruction, send explicit user requests to the independent Scene Editor, which can add, modify or delete component objects and verify the changes through rendered views. Architectural and terrain edits remain the responsibility of the Environment Builder. Update the compositional scene state and render the result after each accepted edit.
Image handling. When the user uploads images, analyze both the image content and the text instruction to decide how each image should be routed. A floorplan sketch should be passed to the Floorplan Designer. A style reference, scene reference or furniture reference should be passed to the Plan Generator with an explicit explanation of its intended usage. A furniture image intended for asset appearance should also be passed to the Asset Generation or Asset Corrector tool when needed. Do not pass every image to every tool; route each image only to the sub-agent or tool that needs it.
General rules.

1. In automatic mode, execute the complete pipeline without asking the user, unless no scene-design intent is detected.
2. In interactive mode, pause at key milestones and wait for user feedback before continuing.
3. Always use the image coordinate system for spatial descriptions and JSON edits.
4. When a verifier or reasoner reports an error, analyze the cause and ask the corresponding agent or tool to regenerate, revise or re-render the result.
5. Preserve consistency among the floorplan JSON, empty top-view image, generated visual plan, parsed 2D layout, lifted 3D layout, generated assets and final rendered scene.

Prompt for Floorplan Designer.

Prompt for Floorplan Designer

You are a versatile spatial layout designer who generates indoor and outdoor scene layouts.
Coordinate system. The design uses a top-down image coordinate system. The origin $(0, 0)$ is at the northwest corner of the floorplan. The X axis points east/right, and the Y axis points south/down. Units are meters and should match real-world dimensions.

```
North (-Y)
    ^
    |
West <-> East
    |
    v
South (+Y)
```

Task. Design one floorplan with a detailed scene description based on the user input. The input may be a simple scene type pair such as `General-Bedroom`, a detailed floorplan specification, a detailed scene description, or a combination of them. If the user provides constraints, respect them closely. If the input is brief, use reasonable spatial-design creativity.
Floorplan requirements. Vertices. The `vertices` field is a closed polygon represented by a sequence of 2D points $[x, y]$, starting from $(0, 0)$. Points connect in order, and the last point connects back to the first. For example, `[[0,0],[4,0],[4,3],[0,3]]` denotes a 4m east-west by 3m north-south rectangle. Shapes may include rectangles, L-shapes, semicircles, concave or convex polygons, circular layouts, and other architecturally meaningful forms. Curves should be approximated by line segments, roughly every 15°.
Height. Normal indoor rooms should usually have height no more than 4m, while large halls can be up to 8m. All outdoor scenes, including fenced sites, and tabletop scenes use height 0; outdoor fences are separate environment elements rather than floorplan walls. If height is greater than 0, vertices represent walls. If height is 0, vertices represent boundaries only. For multi-story buildings, consider only one floor.
Partition. Avoid partitions whenever possible. Only use them for truly complex scenes with multiple enclosed rooms, such as a large apartment or hospital ward. Simple single-function scenes such as bedrooms, kitchens, bathrooms, offices and living rooms should use `partition: []`. Outdoor scenes should also use `partition: []`. Each partition is a wall segment with four values $[x_s, y_s, x_e, y_e]$, namely the start and end points. Do not output partitions as point lists such as `[[3,0],[3,5]]`. Par-

tition segments may form chains and should connect to the outer boundary or other partitions when they define enclosed spaces. Partition points must remain inside the outer polygon, and there is no need to leave gaps for doors.
Description. Write the description in English. Describe functional zones and their spatial relationships, list the main furniture and objects in each zone, and describe texture, style and overall arrangement. Skip tiny decorative items unless they define the scene. If there are partitioned rooms, describe where their entrances should be. Ensure connectivity: every enclosed space must be reachable.
Scene name. Use the format `Style-SceneName`, such as `General-Bedroom`, `ChineseAncient-SwordForge` or `Modern-LivingRoom`. The first part indicates the scene style, and the second part indicates the specific scene type. If the user provides a scene type pair, copy it directly after translating to English if needed. If the user only provides a description, infer an appropriate `Style-SceneName`.
Output format. First explain the design reasoning step by step, then output strictly between the `***` markers:

````
***
```json
[
  {
    "scene_name": "Style-SceneName",
    "environment_type": "indoor",
    "description": "...",
    "floor_plan": {
      "vertices": [[x1, y1], [x2, y2], [x3, y3]],
      "partition": [[xs1, ys1, xe1, ye1]],
      "height": 3.0,
      "shape": "rectangle/L-shape/..."
    }
  }
]
```
***
````

User input.

```
{{ user_input }}
```

Prompt for 2D Bounding-Box Parsing.

Prompt for 2D bounding-box parsing

You are given two top-down perspective images of the same room or scene:

- **Image 1:** the empty room with only walls and floor, corresponding to the rendered floorplan.
- **Image 2:** the furnished top-view visual plan generated from Image 1.

The scene description used to generate Image 2 is `{{ description }}`. Your task is to detect all new objects added in Image 2 that do not exist in Image 1.
Coordinate system.

- Origin: top-left corner $(0, 0)$.
- X axis: rightward, range 0–1000.
- Y axis: downward, range 0–1000.
- The image is normalized to 1000×1000.

The `bbox_2d` format is $[x_1, y_1, x_2, y_2]$. The order is **x first, y second**; do not use $[y, x]$ order. Here, x denotes horizontal position and y denotes vertical position. (x_1, y_1) is the top-left corner, (x_2, y_2) is the bottom-right corner, and every box must satisfy $x_1 < x_2$ and $y_1 < y_2$.
Self-check examples.

- Object in the top-right corner: $x_1 \approx 700$, $y_1 \approx 50$, $x_2 \approx 900$, $y_2 \approx 200$.
- Object in the bottom-left corner: $x_1 \approx 50$, $y_1 \approx 700$, $x_2 \approx 200$, $y_2 \approx 900$.
- A wide object spanning left-to-right should have large $x_2 - x_1$ and small $y_2 - y_1$.
- A tall object spanning top-to-bottom should have small $x_2 - x_1$ and large $y_2 - y_1$.

Label format. Use the form `{category}{id}`, such as `bed0`, `chair0`, `chair1`, `door0` and `window0`. The category should be a precise English noun. Doors use `door` and windows use `window`. The ID is an integer starting from 0 and is unique across the scene.
Required reasoning process. You must reason in the following order before outputting the final JSON.
Step 1: Doors and windows. Scan all walls. For each door or window, identify the wall it lies on and estimate its bounding box. The box should cover the frame only, not the opened area. For windows, make sure the box covers the entire window frame including all panels. If a wall has multiple separate windows, detect each individually. Do not merge adjacent windows into a single box and do not crop the box too tightly.
Step 2: Wall-mounted and ceiling objects. Scan all walls and the ceiling. For each painting, mirror, curtain, chandelier or similar object, describe what it is, what it looks like, and estimate its bounding box.
Step 3: Floor furniture, region by region. Divide the room into spatial regions such as north area, center and east wall area. For each region, list all visible objects and estimate tight bounding boxes that fully cover the visible area, including perspective-visible side faces.
Caption rules. Use English only and avoid position information such as "left", "north wall" or "corner". For individual objects, describe shape, color and material. For large furniture, emphasize distinctive top-down shapes such as L-shaped, U-shaped or round. For doors and windows, never describe them as lying flat on the floor. In a top-down view, doors and windows appear as thin rectangles along walls and are embedded upright in the wall.
Output format. The output must strictly follow the format below. Do not output JSON in the reasoning section. The reasoning should be pure text describing what was found. Output JSON only once at the end.

```
***
{Reasoning process in plain text. Analyze area by area.
List detected objects and approximate locations. Do not
write JSON here.}

Coordinate self-check:
1. Restate the coordinate system:
   "x = horizontal (0=left, 1000=right),
    y = vertical (0=top, 1000=bottom),
    format = [x1, y1, x2, y2]"
2. Pick two objects and verify whether their coordinates
   match their positions.
3. If any bbox has x and y swapped, fix it before writing
   the JSON.

```json
[
  {
    "label": "bed0",
    "bbox_2d": [x1, y1, x2, y2],
    "caption": "Large double bed with wooden headboard and white
    ↪  bedding"
  }
]
```
***
```

Prompt for Orientation and Relation Parsing.

Prompt for orientation and relation parsing

You are given a top-down perspective image of a furnished room or scene and a JSON list of detected objects with `label`, `bbox_2d` and `caption`. You are also given a labeled wall list with each wall's endpoints and inward-facing orientation.
Wall height: `{{ wall_height }}` meters.
Coordinate system. The origin is the top-left corner $(0, 0)$. The X axis points rightward/east and ranges from 0 to 1000. The Y axis points downward/south and ranges from 0 to 1000. In the image, north is upward ($-Y$), south is downward ($+Y$), east is rightward ($+X$), and west is leftward ($-X$).
Direction angles. SpatialFactory uses a non-standard angle convention. This is not the compass convention. Do not use 0° as north. In our convention, 0° points south/downward in the image, and angles rotate counter-clockwise:

Direction	Angle
South (down)	0°
Southeast (down-right)	45°
East (right)	90°
Northeast (up-right)	135°
North (up)	180°
Northwest (up-left)	225°
West (left)	270°
Southwest (down-left)	315°

All orientations must be multiples of 15°. Before writing an angle, verify it against the table. For example, facing bottom-right is southeast and should be 45°, not 135°; facing bottom-left is southwest and should be 315°, not 225°.

Wall list. Each wall has a label, two endpoints, a height and an `orientation`, where `orientation` is the angle of the inward-pointing normal from the wall surface into the room interior.

```
{{ wall_json }}
```

Input objects JSON.

```
{{ bbox_json }}
```

Required reasoning process. For each object, reason in the following order. Earlier attributes inform later ones.

1. supported_by. Decide what physically supports the object. Use `floor` for ground-supported objects, `ceiling` for chandeliers or ceiling fans in indoor scenes, `wall` for wall-mounted or embedded objects, and another object label when one object is supported by another. If object A is supported by object B, A's bounding-box center should lie within B's bounding-box area. Outdoor scenes with wall height 0 have no walls or ceilings, so objects can only be supported by `floor` or another object. Most large furniture such as wardrobes, cabinets, shelves, dressers, desks, tables and beds should be treated as floor-standing unless it is clearly wall-mounted.

2. attach. Determine the hierarchical scene structure. Use `standalone` for main functional furniture such as beds, desks, wardrobes, sofas, bookshelves, dining tables and carpets. Use `attach to the wall` for doors, windows, paintings and wall-mounted items; `attach to the ceiling` for ceiling lamps or chandeliers; and `attach to the {label}` for accessories of another standalone object.

3. side_to. Check every wall explicitly for every object. For each wall, judge whether the object is visually close to, touching or against that wall. A rough 30cm distance threshold can be used as a secondary cue, but visual evidence takes priority. The result may contain zero, one or multiple walls. Corner objects often touch two walls. The field should contain wall labels only, such as `["wall0", "wall3"]`, or an empty list.

4. backwall. The `backwall` field is either one wall from `side_to` or `none`. It indicates the wall that the object's back is pressed against. Selecting a backwall forces the object orientation to be perpendicular to that wall, so verify that the object's visual appearance supports such an orientation. Doors, windows, paintings and wall-mounted objects usually use the wall they are embedded in or hung on. Furniture beside a wall but not backed against it should use `none`. If the object visually faces a diagonal direction, do not force a cardinal wall-based orientation; set `backwall` to `none`.

When `backwall` is not `none`, let the wall inward normal be n. The object should face either n or $(n+180) \bmod 360$. For most furniture backed against a wall, doors, windows and wall-mounted objects, use n so that the visible or functional face points into the room.

5. orientation. If `backwall` is not `none`, the orientation is determined by the wall. Otherwise, infer orientation from visual evidence in the following priority: functional face, surface details, wall proximity, and defaults. A bed faces toward its foot end; a chair faces the table it belongs to; a cabinet or wardrobe faces its door-opening direction; a desk or table faces the user side; and a sofa faces the coffee table or TV direction indicated by cushion arrangement. Visible handles, cushions, screens, headboards, backrests and openings should be used as evidence. Rotationally symmetric objects such as potted plants, vases, round tables and round stools should use 0° because they have no meaningful facing direction.

Do not justify orientation using vague expressions such as "faces center", "faces outward", "faces inward" or "faces the room". The justification must name a specific visible feature and its direction.

6. chip. Set `chip=true` for thin or flat objects such as doors, windows, paintings, wall-mounted TVs, carpets, curtains and mirrors. Set `chip=false` for most furniture, folding screens and wall-mounted cabinets with depth.

Important rules. Keep all objects in the same order as the input JSON. Do not modify existing attributes such as `label`, `caption` or `bbox_2d`; only add new fields. The output JSON must contain the exact same number of objects as the input. When `backwall` is set, the orientation must be collinear with the wall orientation, namely n or $(n + 180) \bmod 360$.

Output format. Show detailed reasoning for every object using the six-step format below, then output JSON only once at the end.

```
***
**{label}**:
1. supported_by: {value} -- {reason}
2. attach: {value} -- {reason}
3. side_to: checking each wall:
   - wall0: {close/not close} -- {why}
   - wall1: {close/not close} -- {why}
   ...
   -> side_to = [...]
4. backwall: {value} -- {reason}
5. orientation: {value} -- {visual evidence and derivation}
6. chip: {value} -- {reason}

```json
[
  {
    "label": "door0",
    "caption": "...",
    "bbox_2d": [x1, y1, x2, y2],
    "orientation": 180,
    "supported_by": "floor",
    "attach": "attach to the wall",
    "side_to": ["wall2"],
    "backwall": "wall2",
    "chip": true
  }
]
```
***
```

Prompt for Height-Range and Mesh-ID Parsing.

Prompt for height-range and mesh-ID parsing

You are given a top-down perspective image of a furnished room with labeled detection boxes and a JSON list describing each detected object with `label`, `caption`, `bbox_2d`, `orientation`, `supported_by`, `attach`, `side_to`, `backwall` and `chip`.
The real wall height of this scene is `{{ wall_height }}` meters.
Important visual prior. The image is taken from a camera directly above the scene center, looking straight down. Because of this, indoor walls around the room perimeter are visible from their side or inner faces, and objects far from the scene center also reveal side faces. Doors and windows appear as elements on these visible wall faces. Use these perspective effects, including visible wall-to-object height ratios when an indoor wall provides a reliable reference, to estimate real object heights.
Task. Add `h_range` and `mesh_id` to each object. The `h_range` field is the real height interval $[h_{\text{low}}, h_{\text{high}}]$ in meters and is used by the 3D Lifting tool. The `mesh_id` field groups visually identical objects for Asset Generation.
Input JSON.
`{{ bbox_json }}`
1. h_range. Estimate each object's height range from the floor using perspective cues and reliable scene dimensions as references. When visible and comparable, a wall of height `{{ wall_height }}` meters can provide a scale reference.

- `floor`: set $h_{\text{low}} = 0$ and estimate h_{high} from visible side faces and comparable size references. Carpets and rugs are approximately $[0, 0.01]$. Doors sit on the floor and should have $h_{\text{low}} = 0$. When an indoor wall is clearly visible, its known height can help estimate the height of nearby furniture.
- `ceiling`: set $h_{\text{high}} =$ `{{ wall_height }}` and estimate h_{low}, typically near the ceiling.
- `wall`: estimate both bounds between 0 and `{{ wall_height }}`. Windows are usually elevated above the floor, except for floor-to-ceiling windows. Curtains, paintings and mirrors should be estimated from their visible extent on the wall face.

- another object label: set h_{low} near the supporting object's top surface height, and set h_{high} to h_{low} plus the object's own height.

For indoor scenes, h_{high} should not exceed `{{ wall_height }}`. When wall height is 0, the scene is open and has no wall-height upper bound; estimate positive object heights from visual evidence and real-world scale instead. Account for protruding parts: a bed with a headboard, a toilet with a tank, a chair with a backrest, or any object with handles, canopies or vertical extensions must cover the tallest visible part. Objects of the same type and similar visual size should have similar height ranges.
2. mesh_id. Assign integer appearance identifiers starting from 0. Objects with identical shape, size, texture and material should share the same `mesh_id`; objects that look different should use different IDs. Use captions as supporting evidence. Objects sharing a `mesh_id` must keep identical captions.
Important rules. Keep all objects in the same order as the input JSON. Do not modify existing attributes such as `label`, `caption`, `bbox_2d`, `orientation`, `supported_by`, `attach`, `side_to`, `backwall` or `chip`; only add `h_range` and `mesh_id`. The output JSON must contain the exact same number of objects as the input.
Output format. Show detailed reasoning for every object before outputting the JSON. For each object, explicitly state what supports it, the estimated h_{low}, the estimated h_{high} using perspective cues and an appropriate scale reference, and its `mesh_id` group. Output JSON only once at the end.

```
***
{Your detailed per-object reasoning. Do NOT write JSON here.}
```json
[
  {
    "label": "door0",
    "caption": "...",
    "bbox_2d": [x1, y1, x2, y2],
    "orientation": 0,
    "supported_by": "floor",
    "attach": "attach to the wall",
    "side_to": ["wall0"],
    "backwall": "wall0",
    "chip": true,
    "h_range": [0, 2.2],
    "mesh_id": 0
  }
]
```
***
```

Prompt for Visual Plan Generation.

Prompt for visual plan generation

You are given a top-down view of an empty floorplan. Your task is to fill this room with furniture and apply wall and floor textures to generate a floorplan-aligned top-view visual plan.
Absolute rule 1: the room shape must not change. The input floorplan is the output floorplan: same walls, corners, angles, lengths and proportions. If the input room is rectangular, the output must be rectangular with the exact same dimensions. If the input room is L-shaped, T-shaped or irregular, the output must preserve the exact same outline. Do not stretch, compress, warp, shift, rotate, crop or distort the floor area in any way. Every wall edge in the output must align with the input. The walls are immovable concrete.
Absolute rule 2: the camera must not move. The camera is fixed directly above the room center, pointing straight down, perpendicular to the floor. Do not tilt, rotate, or change its perspective projection. The output must preserve the input camera's top-down perspective view.
What you can do. Add furniture and objects inside the room, on the floor or against walls. Apply textures to floors and walls, such as wood, tile, paint or wallpaper. Add one closed entrance door embedded in an existing wall. Add closed doors to partitioned rooms if partitions exist.
What you cannot do. Do not change room shape, size, proportions or wall positions. Do not add, remove or move walls or partitions. Do not change the camera angle or viewing direction. Do not scale, crop or transform the image.
Steps.

1. Add one main entrance door from the outside. It must be closed. For outdoor scenes, decide based on context whether a door is needed.
2. Add a door to each partitioned room if any. All doors must be closed.
3. Apply textures to walls and floors matching the room style without altering wall structure.
4. Place furniture and objects matching the description below, making the room realistic and well furnished.
5. Output the final image from the exact same top-down perpendicular viewpoint. The wall layout must be identical to the input.

Final check. Before outputting, compare the output room outline with the input. If any wall has moved, stretched or changed angle, redo it. The floor shape and its pixel coverage in the image must remain unchanged. The line of sight must be perpendicular to the ground. The wall structure in the output must be pixel-identical to the input; textures may change, but geometry may not. Do not add or remove any walls or partitions inside or outside.
Scene description.

```
{{ description }}
```

Prompt for Plan Filter.

Prompt for Plan Filter

You are a professional image quality filter expert.
You are given two images:

- **Image 1:** an empty floorplan, shown as a top-down view with only the floor and walls.
- **Image 2:** a generated scene image that should preserve Image 1's viewing angle and wall structure while filling the empty room with furniture and textures. The image should be viewed from directly above the scene center with the line of sight perpendicular to the ground, so all objects should be seen from above or at a slight angle.

The image generation model commonly makes the following errors:

- The structural wall boundaries, namely the polygon outline of the room, have been scaled, deformed or shifted compared to the original floorplan. Changing floor or wall textures and materials is fine; what matters is that wall positions, lengths and proportions remain structurally identical. Adding rugs, different flooring materials or wall textures is not an error. This is a critical error.
- The viewing angle has changed, and the camera is not centered above the scene with the line of sight perpendicular to the ground. This is a critical error.
- New structural walls or partitions have been added outside or inside the given floorplan, changing the room's structural layout. Adding doors, windows, wall decorations, paintings, shelves, or wall-mounted and wall-leaning furniture such as bookshelves or cabinets to existing walls is fine and expected. These do not change the wall structure. Only judge whether new dividing walls have appeared or existing walls have been removed. This is a critical error.
- Doors are blocked or fully hidden by furniture or objects. Doors must remain clearly visible and accessible from the top-down view. This is a critical error.

Important top-view prior. In a correct top-down view, wall-mounted elements such as doors, windows and paintings naturally appear foreshortened or compressed into narrow strips along the walls. This is expected; do not mistake it for a viewing-angle error. Large 3D furniture such as bookshelves, wardrobes and tall cabinets may also show some side or front surfaces due to their height. This is physically correct. Only flag the viewing angle if the entire scene appears tilted or the camera is clearly not above the room center. Focus on whether the wall structure, including positions, lengths and proportions, is preserved, not on textures, materials or how individual objects are rendered.
Task. Determine whether the generated image meets the requirements and whether any critical error is present. Output your analysis for each criterion, then provide the final answer strictly in the format `***yes***` if the image passes, or `***no***` if any critical error is found.

Prompt for Floorplan Refiner.

Prompt for Floorplan Refiner

You are a floorplan alignment specialist. Your task is to adjust a floorplan's wall structure so that it precisely matches the wall structure visible in a generated top-view visual plan.

Background. SpatialFactory first designs a floorplan with wall vertices in metric coordinates, renders it as a 1000×1000 empty top-view image, and asks a multimodal generation LM to create a furnished visual plan under the same floorplan. However, the generation model may distort the wall structure: walls may shift, stretch, or change proportions. Your job is to adjust the floorplan vertices and partitions so that the rendered empty floorplan matches the wall positions visible in the generated visual plan.

Coordinate system. The origin $(0, 0)$ is at the northwest corner of the floorplan. The X axis points east/right and the Y axis points south/down. Units are meters. North corresponds to $-Y$, south to $+Y$, west to $-X$, and east to $+X$.

Pixel-to-real conversion. The current `pixel2real_ratio` is `{{ pixel2real_ratio }}`, so one pixel equals `{{ pixel2real_ratio }}` meters. The camera center is at pixel $(500, 500)$. Use this ratio to estimate metric offsets from the 1000×1000 comparison images.

What to compare. Focus on the wall outline, including its shape, proportions, dimensions, angles and image position. The rendered floorplan walls should overlap with the scene-image walls as closely as possible. Ignore furniture, rugs, floor textures, wall textures, colors, decorations, doors and windows. Treat doors and windows as part of the wall they are embedded in. If the outer wall boundary is traced in both images, the two outlines should overlap in both position and shape.

Adjustment strategy. Follow two steps in order. First, align shape, scale and position: translate vertices if the outline is shifted, and adjust vertex positions if the wall proportions or dimensions differ. Second, add missing walls and partitions: after the outer boundary is aligned, check whether the scene image shows internal dividing walls, partitions, alcoves, recesses, bay windows or extra outer wall segments that are missing from the current floorplan. For boundary changes, modify `vertices` to include new corners. For internal partitions, add entries to `partition` as wall segments in the form $[x_s, y_s, x_e, y_e]$. Partition endpoints should connect to the outer boundary or to other partitions.

If shape, position and wall completeness are already well aligned, state that no adjustment is needed and stop. Only call the rendering tool when a genuine mismatch is identified. When adjustment is needed, identify the discrepancy, estimate pixel-level differences, convert them to metric adjustments using `pixel2real_ratio`, output a corrected floorplan, render it, compare again, and continue until all wall structures match.

Current floorplan data.

`{{ scene_json }}`

Output format. When calling the rendering tool, pass the complete scene JSON as a string in the exact format below.

````
***
```json
[
  {
    "scene_name": "Style-SceneName",
    "description": "A detailed description of the scene...",
    "floor_plan": {
      "vertices": [[x1, y1], [x2, y2], [x3, y3]],
      "partition": [[xs1, ys1, xe1, ye1]],
      "height": 3.0,
      "shape": "rectangle"
    }
  }
]
```
***
````

Field rules. Keep the original `scene_name` unless the generated image clearly shows a different scene type. The `description` may be refined to better match the generated visual plan. The `vertices` field is a closed polygon whose points connect in order, with the last point connecting back to the first. Keep vertices in a similar coordinate range as the original. Each partition is a wall segment $[x_s, y_s, x_e, y_e]$, not a point list. If the generated image shows no internal walls, use `partition: []`. Usually keep `height` unchanged unless it is clearly wrong. Update `shape` only if the visible shape has changed.

Important notes. Do not change the origin convention. The rendered floorplan will be resized automatically to 1000×1000 for comparison. After each tool call, compare the newly rendered floorplan with the target scene image. Continue adjusting until the wall outlines match well, then stop. Typically one to three iterations are sufficient.

Prompt for Asset Image Generation.

Prompt for Asset Image Generation

This is a cropped top-down view of an object. Based on the description, complete any missing parts and generate a clear, realistic image with rich textures. The background must be black. Do not change the existing shape; only fill missing parts and enhance textures.
Critical rule. The generated image must contain exactly what is described in the object description below: nothing more and nothing less. Include all items listed in the description, such as a desk with a computer and a lamp when all three are mentioned. Do not add any object, furniture or item not mentioned in the description. Remove or ignore any extra objects visible in the input crop that are not part of the description.
For 3D objects such as tables, cabinets and furniture that only show the top surface, transform the crop into a three-quarter view, namely a 30–45 degree elevated front angle similar to a product photo taken from slightly above and in front. The goal is to show the object's full 3D structure: the top surface, front face and one side face should all be clearly visible. Show legs, handles, drawers, shelves and structural details that are hidden in a pure top-down view.
For thin or flat objects such as carpets and paintings, keep the top-down view without viewpoint shift. The generated image must show a complete object with black background and must not be cropped at the edges.
Object description.

```
This is a {{ label }}, specifically: {{ caption }}
```

Prompt for Asset World-Up Correction.

Prompt for Asset World-Up Correction

You are given four rendered views of a 3D object:

- **Image 1:** rendered from the front direction, with the camera at $+Z$ looking toward $-Z$.
- **Image 2:** rendered from the up direction, with the camera at $+Y$ looking toward $-Y$.
- **Image 3:** rendered from the left direction, with the camera at $-X$ looking toward $+X$.
- **Image 4:** rendered from the right direction, with the camera at $+X$ looking toward $-X$.

Object description.

```
The object is: {{ label }} -- {{ caption }}
```

Task. Determine which image shows the correct top-down view of this object, namely the view looking down from above the object as it would naturally sit on a floor or table. First, briefly describe what you see in each image using one or two sentences each. Then state your reasoning for which view is the top-down view. Finally, answer with exactly one line in one of the following formats:

```
***TOPDOWN=1***
***TOPDOWN=2***
***TOPDOWN=3***
***TOPDOWN=4***
```

Prompt for Asset Tilt Correction.

Prompt for Asset Tilt Correction

You are given three rendered views of a 3D object:

- **Image 1:** front view, with the camera at $+Z$ looking toward $-Z$.
- **Image 2:** top-down view, with the camera at $+Y$ looking down toward $-Y$.
- **Image 3:** right-side view, with the camera at $+X$ looking toward $-X$.

Object description.

```
The object is: {{ label }} -- {{ caption }}
```

Coordinate system. We use the glTF standard coordinate system: X points right, Y points up, and Z points front.
Task. Judge whether the object has any obvious tilt or lean that needs correction.

- Rotation around the X axis is visible as forward/backward lean in Image 3, the right-side view.
- Rotation around the Y axis is visible as left/right twist in Image 2, the top-down view.
- Rotation around the Z axis is visible as left/right lean in Image 1, the front view.
- Positive angle means counter-clockwise rotation when looking from the positive axis direction.
- If unsure, or if the tilt is very small, output 0 and be conservative.

First, briefly describe the object's posture in each view. Then state whether any tilt correction is needed and why. Finally, output exactly one line:

`***X=<float>,Y=<float>,Z=<float>***`

For example:

`***X=0,Y=-12.5,Z=3***`

Prompt for Asset Yaw Correction.

Prompt for Asset Yaw Correction

You are a 3D mesh orientation corrector.
How these images were produced. Both images are rendered from the same top-down camera, looking straight down. Both the 3D mesh in Image 1 and the real object in Image 2 have already been rotated by the object's orientation angle `{{ orientation }}` degrees from a baseline of 0°, which faces south. However, the 3D mesh's internal facing direction may differ from the real object's. Your job is to fix this residual mismatch by rotating the mesh around its own vertical axis.
You are given two images:

- **Image 1:** a 3D mesh rendered in the scene, which may need yaw correction.
- **Image 2:** the reference photo showing the real object and its target orientation.

Object description.

`The object is: {{ label }} -- {{ caption }}`

Image directions are top = north, bottom = south, left = west and right = east. The camera is directly above at (`{{ cam_x }}, {{ cam_y }}`), looking straight down.
Task. Rotate the 3D mesh around the vertical axis passing through the object's center so that its facing direction matches the reference photo. Do not simply rotate by the orientation angle `{{ orientation }}` degrees; that orientation has already been applied. Instead, judge the residual mismatch from visual details, such as handles, backrests, screens, doors or front panels.
Rotation convention. A positive angle means counter-clockwise rotation as seen from above. Call `rotate_and_render_yaw(angle)` with the rotation angle in degrees. Each call must be in the range $[-90, 90]$. Rotations are cumulative: each call rotates from the current state, not from the original state. You have at most two rotation attempts. If the first rotation makes the result worse, rotate back and stop.
Rules. Judge orientation by top-surface details visible in the top-down view. Identify asymmetric features, such as a backrest versus seat edge, keyboard versus screen hinge, drawer handles, door panels or armrests, and check whether they face the same direction in both images. Do not rely only on textures or colors. If there is no clear evidence of misalignment, output angle 0. When in doubt, do not rotate. In most cases, the correct orientation requires no rotation or only a small rotation below 45°. If you think the mesh needs an approximately 180° rotation, you are likely wrong; re-examine the images carefully before applying a large rotation.
Critical first check. Before calling the tool, carefully compare Image 1 and Image 2. If the object's orientation already roughly matches the reference, call the tool with angle 0 immediately. Do not rotate an already aligned object.
Before each tool call, explicitly state:

1. The current iteration number.
2. The specific visual features observed in both images.
3. Whether the two images' orientations currently match, based on the visual evidence.
4. If they do not match, how many degrees to rotate and why. If they match or are uncertain, output angle 0.

Stopping rule. When the orientations roughly match, call `rotate_and_render_yaw(0)`. After calling with angle 0, immediately stop. Do not call the tool again. Output a short summary and end the turn.

Prompt for Spatial Reasoner.

Prompt for Spatial Reasoner

You are the **Spatial Reasoner**. Maintain a compositional spatial memory of the current objects, assets, walls, support relations and materials. Refine the scene through observation, tool-based edits and re-rendering; do not treat a textual plan as an applied change.

State and coordinates. The structured scene records each object's label, caption, 2D detection box, 3D center `position=[x,y,z]`, extent `scale=[a,b,c]`, facing `orientation`, asset identifier and scene-graph relations (`supported_by`, `attach`, `side_to`, `backwall`). In image-world coordinates, X points right/east, Y points down/south and Z points up; 0° faces south and angles increase counter-clockwise. The extent b follows the facing direction, a is perpendicular to it, and c is the vertical extent. Blender and SSL use the same X and Z axes but reverse Y relative to image-world coordinates; convert only at the rendering or Blender-tool boundary.

Object-level alignment. Use the Asset Corrector to compare each generated mesh under the original top-view camera with the corresponding visual-plan crop. Correct clear yaw and scale discrepancies while preserving the object's identity and intended support relations. Record the corrected asset and bounding-box state before reasoning about the whole layout.

Layout-level reasoning. Inspect the projected 3D boxes on the visual plan and the 2D floorplan visualization with actual walls and object footprints. The visual plan provides approximate object identity and relations, but it may distort the floorplan. For boundary containment, circulation and unintended overlap, trust the real walls and 2D footprints even if the corrected wireframe shifts away from the generated image. Check each ordinary object's entire rotated footprint, not just its center; wall-embedded doors and windows are exceptions, not a general exemption for furniture. Also verify sizes, vertical support, wall attachment and plausible object relations. Keep changes minimal, but do not leave a known out-of-bound object unresolved.

The `visualize_adjustment` tool accepts the complete updated 3D-box state and returns both visualizations for the next judgment. At this stage, adjust the three position coordinates and three scale values when justified by evidence; preserve orientation, object identity, asset links and other scene-graph fields. The tool synchronizes height ranges from the revised center and vertical extent. After every accepted edit, inspect the new visualizations rather than assuming the proposed correction worked.

Environment building. When enabled, the Environment Builder starts from a neutral floor/wall shell with the existing object assets and compares it with the reference plan. First identify missing architectural or terrain elements and material tasks in a concise checklist. Use Blender Python (`bpy`) code and material nodes to construct or revise the floor, walls, partitions, platforms, stairs, roads, terrain and scene textures as warranted by the reference. Keep these environment elements distinct from generated component assets; do not duplicate objects already represented in the 3D-box state. Preserve labels, captions, support surfaces and material assignments so the compositional scene remains editable.

Use `plan_environment` for the checklist and `inspect_environment` to examine current geometry and materials. Use `edit_environment` to execute a concrete geometry or material change; its feedback includes a fixed top-down view and a custom view. If an important region is hidden, use `observe_environment` to choose another camera without changing the scene, and retain the conclusion with `record_review`. Repair only severe component defects or access problems through `edit_components`; an indispensable object with a simple but unusable shape may be replaced through `replace_component_with_environment`. Minor appearance defects do not justify changing an otherwise valid object. Complete the checklist and visual review through `finish_environment`; if a nonfatal limitation remains, report it rather than repeatedly observing or claiming it is fixed.

Interactive editing and completion. After reconstruction, delegate explicit user requests to the independent Scene Editor. It may add, modify or delete component objects and verify each accepted change through rendered top-down and custom views; architectural and terrain edits remain with the Environment Builder. Keep the structured object state, asset library, environment elements and scene rendering mutually consistent. Stop when no prominent issue remains or the allowed edit budget is reached, and state any unresolved limitation accurately.