Evaluating Scene-Description Alignment in Text-Guided 3D Scene Synthesis via Fine-Grained Constraint Verification

Anonymous ACL submission

Abstract

Accurately synthesizing 3D scenes from user-provided text descriptions is crucial for developing embodied agents. Despite the importance of scene-description alignment, existing evaluation methods for such text-guided 3D scene synthesis either capture only coarse similarity between the synthesized scene and the user description, or ignore the spatial reasoning for verifying object placement. None of them addressed the fine-grained constraints (*e.g.*, X needs to be in the scene in a Y manner) implied by the description from users. To address this, we introduce LEGO, a benchmark dataset that pairs each user description with human-annotated constraints and a reference scene, and LEGO-EVAL, an evaluation framework that decomposes a description into atomic constraints and verifies each one using tools that ground textual references to 3D objects and reason about their spatial relationships. We show that (i) LEGO-EVAL evaluates misalignment far more accurately than existing methods and (ii) current scene synthesis approaches achieve only at most 10% success rate in LEGO-EVAL.

1 Introduction

Training embodied agents in simulators with realistic 3D environments is necessary for narrowing the sim-to-real gap (Shridhar et al., 2020; Li et al., 2023; Puig et al., 2024; Authors, 2024; Mittal et al., 2025). For generalization to the real world, such realistic scenes are required at large scales, which is infeasible to achieve solely by manual crafting (Deitke et al., 2022; Xia et al., 2026). A growing body of work has thus explored automated text-guided 3D scene synthesis (Feng et al., 2023; Çelen et al., 2025; Yang et al., 2024b; Sun et al., 2025; Che et al., 2026), which produces 3D scenes, including their layout, objects, and spatial arrangement based on the text descriptions from users.

While it is important to generate scenes that accurately align with the user description, current methods often fail to do so (Ling et al., 2025). Such misalignment easily arises because the scene description often imposes diverse, fine-grained constraints on the scene—from individual object properties to spatial relations among them—that must be jointly satisfied during synthesis. However, training on such flawed scenes introduces misleading supervision (Figure 1), shifting the training distribution away from the user's intent and degrading agent performance (Nasiriany et al., 2026). To prevent this and improve scene synthesis, it is necessary to conduct fine-grained diagnosis of where and why such scenes fail to align with user descriptions.

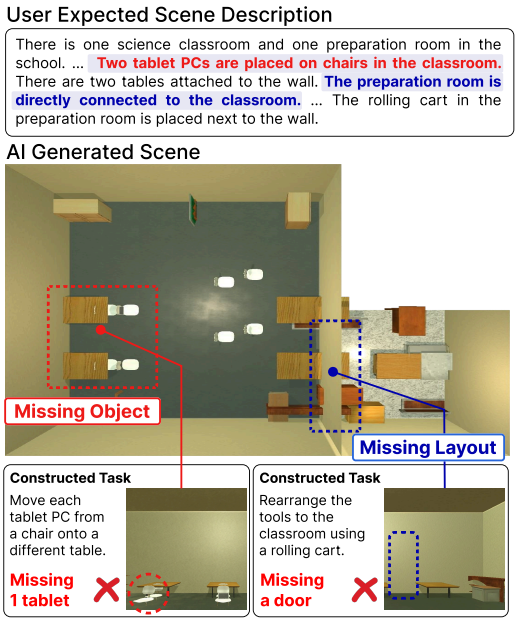

Figure 1: An example of misalignment in text-guided 3D scene synthesis. The generated scene spawns only one of two tablet PCs and omits the door, making the constructed tasks unexecutable and unlearnable.

However, existing efforts toward such fine-grained diagnosis remain limited. Recent scene synthesis methods adopt CLIPScore (Hessel et al., 2021) or VLM-based judges (Li et al., 2024) for evaluation, but CLIPScore captures only coarse-

grained text-scene similarity and VLM-based judges lack the 3D spatial reasoning required to verify object placement (Sengupta et al., 2026). Furthermore, no existing dataset for text-guided 3D scene synthesis pairs text descriptions with the fine-grained constraints they imply, a foundation needed to evaluate and improve the field.

To address these, we introduce **LEGO**, a human-curated dataset for evaluating text-guided 3D scene synthesis. LEGO contains fine-grained natural language descriptions with corresponding constraints, and a manually constructed scene. It also contains rich scene metadata (including positions, orientations, scales, and material properties of scene components), making LEGO a valuable resource not only for evaluation but also for the broader 3D scene synthesis community.

Alongside LEGO, we propose **LEGO-EVAL**, an evaluation framework that diagnoses scene-description alignment in text-guided 3D scene synthesis. LEGO-EVAL systematically decomposes the scene description into individual constraints and verifies each against the synthesized scene using a diverse set of tools, enabling precise diagnosis of the misalignment. This tool-based design moves LEGO-EVAL beyond vision-only judgments, turning synthesized scenes into a reliable foundation for embodied agent training. Our contributions are summarized as follows:

- We introduce LEGO, a human-curated dataset that pairs each natural-language scene description with fine-grained annotated constraints and a reference scene for text-guided 3D scene synthesis evaluation.
- We propose LEGO-EVAL, a tool-augmented evaluation framework that decomposes each scene description into individual constraints and verifies them one by one, enabling constraint-level diagnosis of misalignment.
- We show that LEGO-EVAL delivers substantially more accurate fine-grained evaluation than existing methods, and reveal that current synthesis approaches fall short of satisfying fine-grained scene descriptions.

2 Background

Text-guided 3D scene synthesis. The goal of text-guided 3D scene synthesis is to construct a 3D environment that faithfully reflects a user's intent. To this end, users express their requirements through a natural language description l. A synthesis method $\mathcal{F}$ processes l and produces a structured scene representation $\mathcal{S} = \mathcal{F}(l)$.

Why faithful synthesis matters. Using simulators, we train and evaluate embodied agents on tasks curated within diverse 3D scenes. Since tasks are inherently grounded in the scene, users specify their target task distribution through a set of scene descriptions l, and construct or synthesize the scenes $\mathcal{S}$ accordingly (Yang et al., 2024b; Xia et al., 2026). Any misalignment between $\mathcal{S}$ and l thus shifts the task distribution the agent is exposed to, undermining the agent's ability to learn the intended behaviors. Such misalignment makes any observed success or failure uninterpretable, confounding agent failures with environment errors.

Challenges in evaluating text-guided 3D scene synthesis. However, evaluating whether the scene $\mathcal{S}$ align with the description l is inherently difficult, as it requires verifying every constraint encoded in l. Grounding these constraints in a 3D scene is challenging, as it requires identifying the relevant scene components and reasoning about their properties and relationships—for instance, assessing *"a blue chair placed next to a black desk"* requires multi-hop grounding across object identity, attributes, and spatial relationships. This underscores the need for an evaluation framework capable of verifying diverse constraints through explicit, fine-grained grounding.

3 LEGO: Dataset for Text-Guided 3D Scene Synthesis Evaluation

In this section, we introduce **LEGO**, a human-curated dataset for fine-grained evaluation of text-guided 3D scene synthesis.

3.1 Dataset Overview

LEGO is a fully human-curated dataset spanning the entire pipeline from collecting description to scene construction and constraint annotation. As illustrated in Figure 2, each instance comprises a fine-grained natural language description with annotated constraints, a manually constructed scene, and rich scene metadata. The dataset contains 130 descriptions paired with 1,250 constraints in total, averaging 9.6 per description, with diverse constraint types as shown in Figure 3. Beyond constraints, LEGO also encompasses diverse rooms, receptacles, objects, and structural elements, as

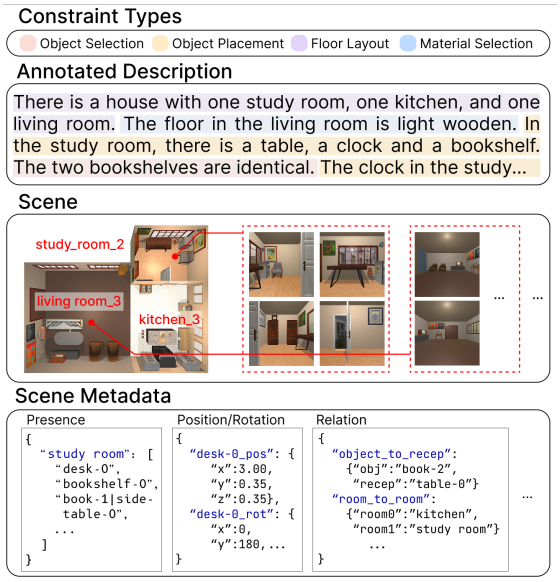

Figure 2: Overview of LEGO

detailed in Table 1, reflecting the complexity and diversity of real-world indoor environments.

3.2 Dataset Construction

Here, we present the construction process of LEGO, where four human annotators manually construct descriptions, annotate constraints, and build corresponding scenes for 3D scene synthesis. Full details are provided in Appendix A.2.

Description Construction and constraint Annotation. First, four annotators create fine-grained descriptions based on their empirical observations and real-world images of indoor spaces. Subsequently, annotators identify all constraints embedded in each description using a dedicated annotation tool. Constraints with dependencies are restructured into conditional expressions, and multiple attributes of the same entity are merged into unified statements. Each constraint is then classified into one of four types—floor layout, material selection, object selection, or object placement—and mapped to its corresponding text segment in the description.

Scene Construction and Verification. Using the collected descriptions and constraints, annotators first generate initial scenes using Holodeck (Yang et al., 2024b), which produces scenes as structural JSON representations containing object positions, rotations, and other scene properties. Object instances are sourced from Objaverse (Deitke et al., 2023), while structural components—including walls, floors, doors, windows, and materials—are drawn from ProcTHOR (Deitke et al., 2022) and

	Room	Receptacle	Object	Structural Elements
# Total	295	2,320	4,518	3,974
# Types	161	446	812	3
Avg./scene	2.27	17.85	34.75	30.57

Table 1: Statistics for rooms, receptacles, objects, and structural elements in LEGO. Room types reflect functional categories rather than layout; receptacle and object types are identity-based. Structural elements include doors, windows, and walls.

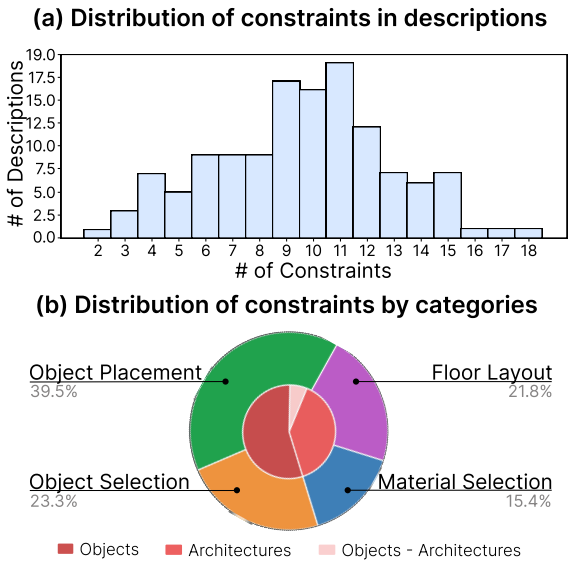

Figure 3: Statistics of LEGO.

AI2-THOR (Kolve et al., 2017) asset libraries. Annotators then manually refine the generated scenes to ensure full alignment with all constraints. Finally, annotators cross-validate each other's work across two iterations, verifying both annotation accuracy and scene-description alignment, with any discrepancies corrected before completion.

4 LEGO-EVAL: Evaluation with Tool-Augmented VLMs

We propose **LEGO-EVAL**, a tool-augmented evaluation framework for fine-grained diagnosis of scene-description alignment in text-guided 3D scene synthesis. We first describe the framework in detail, followed by its implementation details.

4.1 LEGO-EVAL

LEGO-EVAL decomposes a description l into individual constraints and evaluates each sequentially in an iterative manner, with each evaluation informed by the results of prior constraints. To do so, LEGO-EVAL leverages tools from tool set $\mathcal{T}$ to retrieve the necessary information from a generated

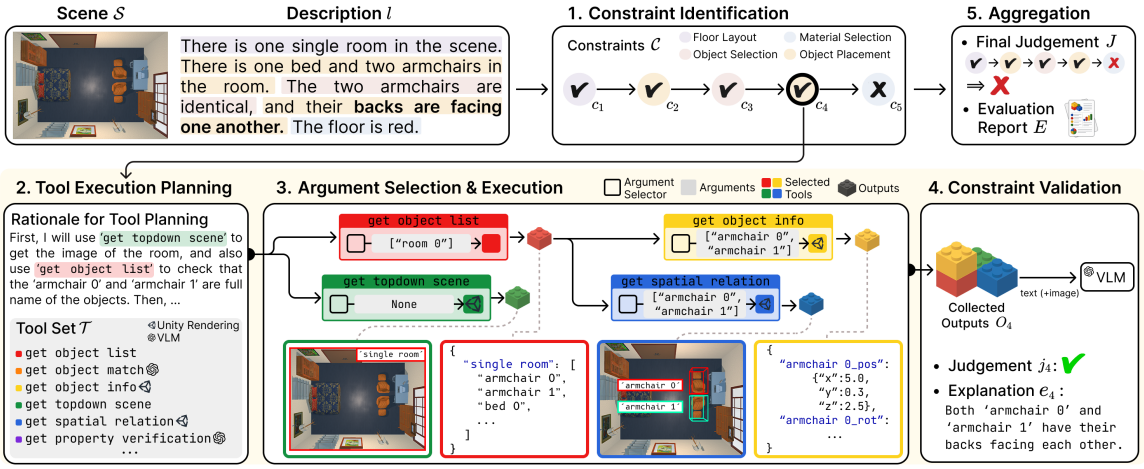

Figure 4: LEGO-EVAL. All steps are performed by a VLM judge $\mathcal{M}$, which plans tool execution, selects arguments, executes tools, evaluates each constraint, and aggregates the results into a final judgment J and a report E.

scene $\mathcal{S}$, and employs a VLM judge $\mathcal{M}$ to provide binary judgment J along with evaluation report E:

$$J, E \leftarrow \text{LEGO-EVAL}(l \mid \mathcal{S})$$

As shown in Figure 4, our evaluation framework consists of the following five steps:

Step 1: Constraint Identification. Since a description l is expressed as a natural language description, it may encode multiple atomic requirements, each of which must be satisfied in the scene and evaluated separately. Therefore, the judge $\mathcal{M}$ first decomposes l into a set of constraints $\mathcal{C} = \{c_i\}_{i=1}^{k}$. Since different constraints require different validation strategies (*e.g.*, *verifying spatial layout* vs. *checking object appearance*), each constraint c_i is assigned a constraint type τ_i that determines how it should be evaluated; τ_i belongs to one of four types[1]:

- **Floor Layout**: Spatial layout of rooms, walls, doors, and windows.
- **Material Selection**: Visual appearance of floors and walls.
- **Object Selection**: Appearance of objects, including doors and windows.
- **Object Placement**: Object placement and rotation within the scene.

Step 2: Tool Execution Planning. The set of constraints $\mathcal{C} = \{c_i\}_{i=1}^{k}$ are evaluated sequentially. For each constraint c_i, evaluating a single constraint often requires multi-hop reasoning—for instance, verifying "*the cup is on the red table*" first requires confirming the existence and color of the table before assessing the cup's placement. To capture such dependencies, the judge $\mathcal{M}$ generates a graph-structured tool execution plan, selecting a subset of tools $\mathcal{T}_i \subseteq \mathcal{T}$ based on the assigned constraint type τ_i. Furthermore, if the information needed to evaluate the constraint c_i has already been retrieved while processing prior constraints $\{c_\ell\}_{\ell<i}$, the model reuses prior results to avoid redundant tool executions.

Step 3: Tool Argument Selection & Execution. While Step 2 determines the logical order of tool execution, accurate validation further requires precise grounding of the relevant objects within the scene $\mathcal{S}$. For this, for each tool $t \in \mathcal{T}$ in the execution plan, the judge $\mathcal{M}$ selects tool arguments α by conditioning on the current constraint c_i of type τ_i, along with all prior judgements $\{j_\ell\}_{\ell<i}$ and explanations $\{e_\ell\}_{\ell<i}$. The resulting outputs are collected as $O_i = \{t(\alpha; \mathcal{S})\}_{t \in \mathcal{T}_i}$ for constraint validation.

Step 4: Constraint Validation. Given the collected tool outputs O_i, the judge $\mathcal{M}$ produces a binary judgement j_i and explanation e_i assessing whether the constraint c_i is satisfied in the scene $\mathcal{S}$. Steps 2–4 are repeated for each constraint $c_i \in \mathcal{C}$, accumulating per-constraint evaluation results $\{(j_i, e_i)\}_{i=1}^{k}$ for aggregation.

Step 5: Aggregation. After all constraints are evaluated via Steps 2-4, the judge $\mathcal{M}$ aggregates per-constraint evaluation results $\{(j_i, e_i)\}_{i=1}^{k}$ into a final judgment J and a comprehensive evaluation report E. Final judgement J is valid only if all binary judgements $\{j_i\}_{i=1}^{k}$ are valid, ensuring that no fine-grained requirement is overlooked. Further-

[1]Inspired by Holodeck (Yang et al., 2024b).

Methods	Holistic ↑				Partial ↑			
	F1	Precision	Recall	Cohen's κ	F1	Precision	Recall	Cohen's κ
CLIPScore	0.49	0.51	0.51	0.02	0.46	0.50	0.50	0.00
VLM-as-a-judge								
Gemini 2.5 Pro	0.38	0.76	0.52	0.05	0.60	0.66	0.70	0.28
GPT-4.1	0.40	0.67	0.53	0.05	0.68	0.65	0.73	0.35
GPT-5.1	0.38	0.68	0.52	0.04	0.60	0.71	0.71	0.28
SceneEval*								
GPT-4o	0.47	0.74	0.58	0.15	0.45	0.64	0.58	0.12
LEGO-EVAL								
Qwen3-VL-8B	0.67	0.74	0.69	0.38	0.71	0.74	0.69	0.42
Qwen3-VL-32B	0.67	0.79	0.70	0.39	0.80	0.80	**0.81**	0.61
GPT-4.1-mini	0.70	0.78	0.72	0.43	0.78	0.80	0.76	0.56
GPT-4.1	**0.81**	**0.84**	**0.82**	**0.63**	**0.83**	**0.86**	**0.81**	**0.66**

Table 2: Human comparison of LEGO-EVAL and existing evaluation methods.

more, the report E provides comprehensive per-constraint explanations that enable detailed analysis of scene deviations from the description l.

4.2 Implementation Details

Environment. LEGO-EVAL is built upon a AI2THOR (Kolve et al., 2017), a Unity-based 3D simulator, enabling direct access to object attributes and spatial configurations via the rendering engine.

Tool set. LEGO-EVAL is augmented with 21 diverse tools, grouped into three types:

- **Environment Interaction**: Visual information retrieval via direct Unity interface (*e.g.*, `get topdown scene`, `get spatial relation`).
- **Textual Reasoning**: Attribute retrieval from structured textual scene representation (*e.g.*, `get room list`, `get object info`).
- **Multimodal Reasoning**: Visual-to-text conversion using VLMs (Wang et al., 2024a) for reliable image-based reasoning (*e.g.*, `get object match`, `get property description`).

More details on the environment and tools are in Appendix B.1 and B.4.

5 Experiments

To validate the LEGO-EVAL, we evaluate the agreement with human judgment for different automatic evaluation methods using LEGO. Furthermore, leveraging both LEGO-EVAL and LEGO, we benchmark existing LLM-based 3D scene synthesis methods, revealing their limitations.

5.1 Comparing with Existing Evaluation

5.1.1 Setup

Evaluation Dataset. We use the descriptions, annotated constraints, and scenes from LEGO to compare the performance of evaluation methods. To enrich the dataset, we additionally construct a misaligned pair for each of the 130 original descriptions by manually crafting a scene that violates the description, resulting in 260 pairs in total.

Baselines. We compare LEGO-EVAL against CLIPScore (Hessel et al., 2021), VLM-as-a-judge, and SceneEval (Tam et al., 2025). CLIPScore computes the similarity between the description and the top-down scene image, and judges the scene as aligned if the score exceeds a threshold of 20. For VLM-as-a-judge, we provide scene images from four perspectives and use Gemini-2.5-Pro, GPT-4.1, and GPT-5.1 as base models with self-consistency across 3 samples. For SceneEval, we leverage GPT-4o as the backbone VLM[2]. Following LEGO-EVAL, all baseline methods consider a scene to be aligned with the description if all constraints are satisfied.

Metrics. We evaluate the methods using F1, precision, recall, and Cohen's κ. Cohen's κ measures agreement with human judgments beyond chance, a chance-corrected complement to F1. We compute these metrics at two levels: (1) Holistic, which measures agreement with human judgments on the full description, and (2) Partial, which measures agreement on each constraint. More details on Cohen's κ is on Appendix B.3

5.1.2 Results

As shown in Table 2, existing methods all suffer from distinct limitations. CLIPScore shows unreliable results for evaluating fine-grained alignment with 3D scenes. This may stem from the as CLIP (Radford et al., 2021) being trained on 2D

[2]SceneEval cannot assess 41% of the constraints in LEGO due to its fixed evaluation criteria; we evaluate it only on the constraints it can handle for a fair comparison.

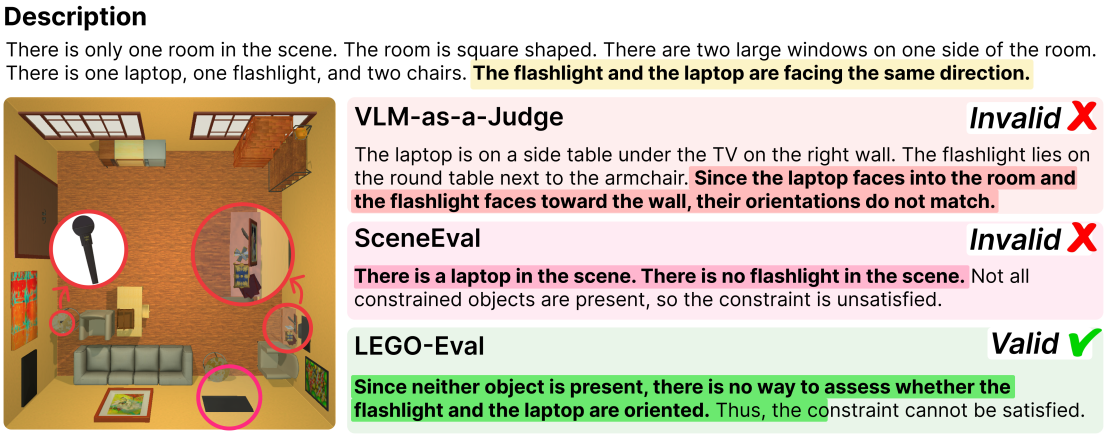

Figure 5: Comparison of evaluation results from VLM-as-a-judge, SceneEval, and LEGO-EVAL.

image-text pairs, which limits its ability to capture 3D scene structure. VLM-as-a-judge fails to identify and ground the scene components referenced in the description, leading to incorrect evaluation. SceneEval does not overcome the grounding limitation of VLM-as-a-judge, and its evaluation suffers from restricted scope—it applies rigid rule-based criteria for spatial relations such as 'beside' and only handles object-level relations.

In contrast, LEGO-EVAL significantly outperforms all baselines, achieving the highest F1 and Cohen's κ at both the holistic and partial levels. Notably, even with a lightweight backbone such as Qwen3-VL-8B, LEGO-EVAL still outperforms all baselines. This advantage stems from LEGO-EVAL's tool-augmented design, which integrates diverse tools for multi-hop grounding to locate and interpret scene components more reliably than the baselines. These results demonstrate that LEGO-EVAL provides fine-grained and reliable evaluation of text-guided 3D scene synthesis.

Case Study. Figure 5 illustrates an example comparing evaluation results from VLM-as-a-judge, SceneEval, and LEGO-EVAL, revealing that while all methods achieve accurate judgments, their reasoning processes differ significantly. Although the flashlight and laptop do not exist in the scene, the VLM-as-a-judge locates them and determines they are not facing the same direction. Similarly, SceneEval misidentifies the black painting on the wall as a laptop. In contrast, LEGO-EVAL accurately recognizes the absence of both objects and determines the constraint cannot be satisfied.

5.1.3 Ablation Study

We conduct an ablation study to verify that each tool type in our toolset contributes substantially to evaluation performance. Specifically, we disable each tool type individually and measure the resulting performance of LEGO-EVAL, while keeping tools that return lists of scene components enabled, as these are necessary for argument selection. As shown in Table 3, disabling any single tool type degrades performance, confirming that all are indispensable for comprehensive evaluation. The largest drop occurs when textual tools are disabled, underscoring that visual inputs alone—on which conventional evaluation methods often rely—are insufficient. Textual tools capture subtle scene attributes, such as colors of small objects that are hard to identify from images alone. They also convey key information about scene components extracted from retrieved images.

Tools	Holistic F1 (Δ)	Partial F1 (Δ)
w/o **M**	-0.04%	-1.02%
w/o **T**	-5.05%	-2.65%
w/o **T** + **M**	-6.46%	-2.81%
w/o **E** + **M**	-24.90%	-5.34%

Table 3: Performance drop with tool types disabled. (**M**: Multimodal Reasoning, **T**: Textual Reasoning, **E**: Environment Interaction)

5.2 Evaluation via LEGO-EVAL

5.2.1 Setup

Baselines. We evaluate five LLM-based text-guided 3D scene synthesis methods on LEGOwith LEGO-EVAL: LayoutGPT (Feng et al., 2023), Holodeck (Yang et al., 2024b), I-Design (Çelen et al., 2025), LayoutVLM (Sun et al., 2025) and MANSION (Che et al., 2026). These methods vary in functionality: Holodeck and MANSION generate complete scenes with object selection, attributes, and placement; I-Design only selects and

Method	Holistic SR ↑	Partial SR ↑				
		Floor Layout	Material Selection	Object Selection	Object Placement	Avg.
I-Design	3.8	92.7	63.7	11.0	4.1	34.2
MANSION	5.4	67.7	53.9	55.7	56.1	58.2
LayoutGPT	6.9	96.0	**65.3**	40.9	37.3	55.2
Holodeck	8.4	**96.3**	61.6	46.3	43.7	58.5
LayoutVLM	**10.0**	95.6	**65.3**	**49.8**	**46.0**	**60.6**

Table 4: Evaluation results of LLM-based 3D scene synthesis methods on LEGO.

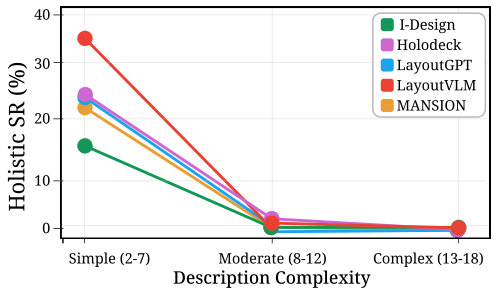

Figure 6: Description complexity vs. Holistic SR.

Method	Total	Ident.	Spa.	Attr.
VLM-as-a-judge	376	150	76	150
+ **Grounding DINO**	451	288	42	121
+ **LASER**	497	233	100	164
+ **Point-LLM**	447	203	72	172
LEGO-Eval	**119**	**53**	**20**	**46**

Table 5: Number of errors across different evaluation methods, all based on GPT-4.1

places objects but omits attributes; LayoutGPT and LayoutVLM only position a given set of objects. For fair comparison, we augment the latter three with Holodeck to produce full scenes for evaluation. Details on the methods are in Appendix D.2.

Metrics. We measure performance with two Success Rates (SR) metrics: (1) Holistic SR, the proportion of descriptions with all specified constraints satisfied; (2) Partial SR, the proportion of valid constraints. For partial SR, we report success rates by constraint type and their overall average.

5.2.2 Results

As shown in Table 4, no current method achieves a holistic SR above 10%, meaning that nearly no synthesized scene satisfies the full description. While most methods reach average partial SRs exceeding 50%, their performance on object selection and placement falls below this level. This shows that current methods still struggle with two fundamental aspects of scene synthesis—choosing the right objects and placing them appropriately.

Existing methods fail on complex descriptions. Real-world descriptions are inherently complex, as evidenced by our fully human-curated LEGO, where user-written room descriptions (Appendix C.6) contain an average of 18.2 constraints per room. To analyze how this complexity affects performance, we categorize descriptions based on the number of constraints into three groups: simple (2–7 constraints), moderate (8–12), and complex (more than 12). As Figure 6 shows, existing methods consistently fail on complex descriptions—precisely what real-world use demands. This means that scenes synthesized by current methods are likely to deviate from the real-world training distribution, undermining their usefulness as training environments.

6 Exploring the Reliability and Utility of LEGO-EVAL

Beyond measuring performance, for an evaluator to be practically useful, it must reliably diagnose why scenes fail and help correct them. Therefore, in this section, we further analyze LEGO-EVAL by examining the source of its reliability (§6.1) and its utility as a feedback signal for refinement (§6.2).

6.1 Bottlenecks for Reliable Evaluation

One might attribute the performance gap in Section 5.1 to a lack of identification support in existing methods. To test this hypothesis, we categorize judgment errors into three types—Identification, Spatial, and Attribute—and manually annotate them to pinpoint the primary bottleneck. As identification is a prerequisite for all subsequent reasoning, we treat it as the primary bottleneck and evaluate three augmented variants, each pairing the judge with a dedicated identification module: bounding box detection (Grounding DINO (Liu et al., 2023)), scene graph generation (LASER (Huang et al., 2025)), and 3D language models (Point-LLM (Guo et al., 2023)). More de-

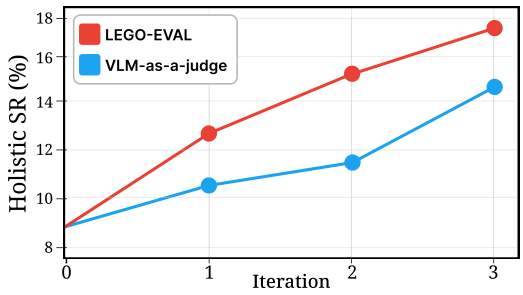

Figure 7: Comparison of refinement results using VLM-as-a-judge and LEGO-EVAL as feedback signal.

tails of the modules and experimental setup are presented in Appendix C.5.

Augmenting identification modules alone worsens evaluation. Surprisingly, all variants increase total errors rather than reducing them. Identification errors in particular worsen substantially despite being the very type each module was designed to address. This stems from the judge's inability to assess if the provided information is relevant to the constraint being verified, regardless of how much scene information is supplied. These results validate that LEGO-EVAL's advantage stems not from identification support alone, but from its constraint-driven reasoning—determining what information each constraint needs before invoking any tool.

6.2 Applying LEGO-EVAL to Improve Scene Synthesis

Beyond evaluation, fine-grained diagnosis can directly support improvement when fed back to the generator. To examine this, we use LEGO-EVAL's rationale as feedback to refine the results of Holodeck (Section 5.2) over 3 refinement iterations, with GPT-4.1 as the baseline VLM judge.

Fine-grained diagnosis improves text-guided 3D scene synthesis. As shown in Figure 7, LEGO-EVAL's feedback leads to greater improvements in Holodeck than VLM-as-a-judge, demonstrating its higher utility as a refinement signal. Since feedback quality ultimately depends on accurate grounding, even a stronger base model cannot compensate when grounding itself is unreliable. In contrast, LEGO-EVAL grounds each constraint via dedicated tools, producing detailed and interpretable feedback. Together, these establish LEGO-EVAL as a reliable signal for both evaluating and improving text-guided 3D scene synthesis.

7 Related Work

Text-guided 3D scene synthesis. Recently, LLMs and VLMs have been adopted for 3D scene synthesis, leveraging their real-world priors. Prior works generally fall into three categories: approaches that generates complete 3D environments from descriptions (Feng et al., 2023; Yang et al., 2024b; Bucher and Armeni, 2025), approaches that focus on selecting relevant objects and determining their placements (Yang et al., 2024a; Çelen et al., 2025), and layout optimization methods that refine the spatial arrangement of predefined assets (Sun et al., 2025; Ran et al., 2025).

Automatic evaluation for 3D scene synthesis. Prior work evaluates text-guided 3D scene synthesis with CLIPScore between top-down scene images and descriptions (Öcal et al., 2024; Fu et al., 2024; Deng et al., 2025), or by prompting VLMs with multi-view images (Wang et al., 2024b; Çelen et al., 2025; Sun et al., 2025; Ling et al., 2025). However, these methods fail to reliably assess fine-grained scene fidelity, such as object counts, attributes, and spatial relations (Ma et al., 2023; Li et al., 2024). Recently, SceneEval (Tam et al., 2025) predefines evaluation criteria but cannot evaluate attributes or spatial relations involving architectural elements (*e.g.*, "*A sliding window is on the orange wall*"), and restricts spatial relations to basic ones like "*left of*" or "*near*", excluding comparative expressions such as "*The table is closer to the chair than the bed*". In contrast, LEGO-EVAL supports spatial reasoning across all scene components and handles diverse natural-language relationships.

8 Conclusion

We present LEGO, a human-curated dataset for evaluating text-guided 3D scene synthesis, and LEGO-EVAL, a tool-augmented framework that evaluate scene-description misalignment at the constraint level. We show that LEGO-EVAL substantially outperforms existing evaluators, and further reveal that current scene synthesis methods achieve holistic success rates below 10%—highlighting the gap between current capabilities and real-world demands. We hope LEGO and LEGO-EVAL together drive progress toward more faithful text-guided 3D scene synthesis, providing a foundation of diverse and reliable 3D environments for training more capable embodied agents.

Limitations

Despite the contributions of our work, several limitations remain. First, LEGO-EVAL only supports scenes composed of rectangular or square rooms. This limits its ability to evaluate more diverse spatial layouts and descriptions beyond these room structures. Extending the benchmark to handle richer scene geometries remains an important direction for future work. Second, the environment interaction tools and multimodal reasoning tools used in our pipeline are simulator-dependent. Consequently, when the simulator is unavailable, evaluation must rely only on textual tools. However, even under this constraint, our method still provides more reliable results than existing baseline approaches. Finally, evaluating on LEGO requires approximately two hours. This introduces a limitation in evaluation efficiency. Nevertheless, we believe this issue can be sufficiently mitigated with improvements in hardware performance.

References

Genesis Authors. 2024. Genesis: A generative and universal physics engine for robotics and beyond.

Martin JJ Bucher and Iro Armeni. 2025. Respace: Text-driven 3d scene synthesis and editing with preference alignment. *arXiv preprint arXiv:2506.02459*.

Ata Çelen, Guo Han, Konrad Schindler, Luc Van Gool, Iro Armeni, Anton Obukhov, and Xi Wang. 2025. I-design: Personalized llm interior designer. In *European Conference on Computer Vision*, pages 217–234. Springer.

Lirong Che, Shuo Wen, Shan Huang, Chuang Wang, Yuzhe Yang, Gregory Dudek, Xueqian Wang, and Jian Su. 2026. Mansion: Multi-floor language-to-3d scene generation for long-horizon tasks. *arXiv preprint arXiv:2603.11554*.

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, and 1 others. 2023. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems*, 36:35799–35813.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. 2022. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994.

Wei Deng, Mengshi Qi, and Huadong Ma. 2025. Global-local tree search in vlms for 3d indoor scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 8975–8984.

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

Rao Fu, Zehao Wen, Zichen Liu, and Srinath Sridhar. 2024. Anyhome: Open-vocabulary generation of structured and textured 3d homes. In *European Conference on Computer Vision*, pages 52–70. Springer.

Ziyu Guo, Renrui Zhang, Xiangyang Zhu, Yiwen Tang, Xianzheng Ma, Jiaming Han, Kexin Chen, Peng Gao, Xianzhi Li, Hongsheng Li, and Pheng-Ann Heng. 2023. Point-bind point-llm: Aligning point cloud with multi-modality for 3d understanding, generation, and instruction following. *Preprint*, arXiv:2309.00615.

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

Jiani Huang, Ziyang Li, Mayur Naik, and Ser-Nam Lim. 2025. Laser: A neuro-symbolic framework for learning spatial-temporal scene graphs with weak supervision. In *International Conference on Learning Representations (ICLR)*.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, and 1 others. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, and 1 others. 2023. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR.

Chengzu Li, Caiqi Zhang, Han Zhou, Nigel Collier, Anna Korhonen, and Ivan Vulić. 2024. Topviewrs: Vision-language models as top-view spatial reasoners. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1786–1807.

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

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and 1 others. 2023. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*.

Zixian Ma, Jerry Hong, Mustafa Omer Gul, Mona Gandhi, Irena Gao, and Ranjay Krishna. 2023. Crepe: Can vision-language foundation models reason compositionally? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10910–10921.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Munoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, and 1 others. 2025. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. 2026. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*.

Başak Melis Öcal, Maxim Tatarchenko, Sezer Karaoğlu, and Theo Gevers. 2024. Sceneteller: Language-to-3d scene generation. In *European Conference on Computer Vision*, pages 362–378. Springer.

Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander Clegg, Michal Hlavac, So Yeon Min, and 1 others. 2024. Habitat 3.0: A co-habitat for humans, avatars, and robots. In *International Conference on Learning Representations*, volume 2024, pages 15306–15336.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, and 1 others. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

Xingjian Ran, Yixuan Li, Linning Xu, Mulin Yu, and Bo Dai. 2025. Direct numerical layout generation for 3d indoor scene synthesis via spatial reasoning. *arXiv preprint arXiv:2506.05341*.

Kathakoli Sengupta, Kai Ao, and Paola Cascante-Bonilla. 2026. Scenecritic: A symbolic evaluator for 3d indoor scene synthesis. *Preprint*, arXiv:2604.13035.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

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

Hou In Ivan Tam, Hou In Derek Pun, Austin T Wang, Angel X Chang, and Manolis Savva. 2025. Sceneeval: Evaluating semantic coherence in text-conditioned 3d indoor scene synthesis. *arXiv preprint arXiv:2503.14756*.

Fei Wang, Xingyu Fu, James Y Huang, Zekun Li, Qin Liu, Xiaogeng Liu, Mingyu Derek Ma, Nan Xu, Wenxuan Zhou, Kai Zhang, and 1 others. 2024a. Muirbench: A comprehensive benchmark for robust multi-image understanding. *arXiv preprint arXiv:2406.09411*.

Yian Wang, Xiaowen Qiu, Jiageng Liu, Zhehuan Chen, Jiting Cai, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. 2024b. Architect: Generating vivid and interactive 3d scenes with hierarchical 2d inpainting. *Advances in Neural Information Processing Systems*, 37:67575–67603.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, and 1 others. 2026. Sage: Scalable agentic 3d scene generation for embodied ai. *arXiv preprint arXiv:2602.10116*.

Yixuan Yang, Junru Lu, Zixiang Zhao, Zhen Luo, James JQ Yu, Victor Sanchez, and Feng Zheng. 2024a. Llplace: The 3d indoor scene layout generation and editing via large language model. *arXiv preprint arXiv:2406.03866*.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, and 1 others. 2024b. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237.

A LEGO

A.1 Data Annotation Tool

To reduce burden of human annotation, we developed a tool for constraint identification, and use Label Stuio for constraint classification, Figure 8 shows the interface of the tool for annotating the constraints within the descriptions. This tool allows the annotators to track if all constraints are accurately identified. Figure 9 shows the interface of Label Studio for annotating the constraint types and mapping each constraint with the corresponding parts of the description. The constraint type is for annotating the constraints to the correct type, and constraint number is to map the constraint to the appropriate part of the description.

A.2 Details of Human Annotation

Our data collection process follows the process below:

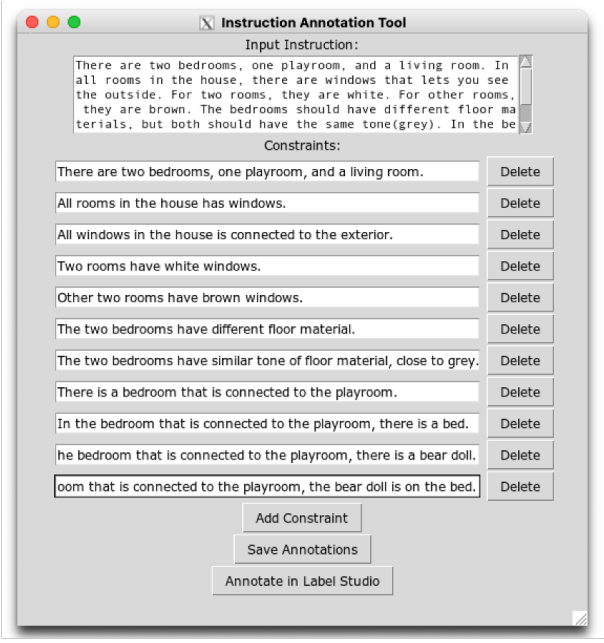

Figure 8: Our annotation tool used for annotating constraints.

Step 1: Annotator recruiting and education. To collect data, we recruit human annotators to construct the dataset. All annotators completed a two-hour education session prior to annotation. This education covers a detailed explanation about guidelines for writing quality descriptions and identifying constraints, data annotation interface, and generating scenes that align with the description. After completing the training, each annotator is assigned to create descriptions, annotate them, and generate corresponding aligned scenes.

Step 2: Data annotation. The annotation process is structured into three phases. In the first phase, we ask each annotator to create 30 to 50 fine-grained descriptions describing the scenes. Annotators are instructed to create descriptions that reflect real-world scenes relying on their empirical experiences. To avoid ambiguity, we additionally require annotators to explicitly specify the viewpoint in the descriptions. For example, when the fridge is placed against a wall, we ask annotators to use expressions such as ”facing the fridge,” so that ”left” and ”right” are interpreted from that reference perspective.

In the second phase, annotators annotate each description by identifying the constraints included in the description using our annotation tool (See Figure 8) and Label Studio(See Figure 8). As some constraints can only be satisfied when others are fulfilled, we express these dependencies explicitly. We achieve this by restructuring the constraints into conditional expressions that directly reference their prerequisites. For example, "*There is a room with a red floor. There is a bed in that room. The bed is white.*" The sentence "The bed is white" relies on the preceding context, as it refers to the bed in the room with the red floor. This constraint can then be expressed as "The bed in the room with the red floor is white." We also merge multiple attributes of the same entity into unified statements, such as combining "the three walls are yellow" and "the three walls have patterns" into "*the three walls are yellow and have patterns.*" Once the constraints have been revised, each constraint is annotated with one of four constraint types: floor layout, material selection, object selection, or object placement. We also annotate which parts of each description correspond to specific constraints, similar to entity annotation in Named Entity Recognition (NER).

In the third phase, annotators create a scene that fully validates the constraints within the description. To achieve this, we first input the descriptions into Holodeck (Yang et al., 2024b), which generates scenes in textual representations (*i.e.*, JSON scripts) that can be rendered in the 3D simulator. The textual representations contain comprehensive scene information, including the asset ID of each component as well as the exact position and rotation of all scene components. As shown in Table 4, Holodeck hardly generates scenes that fully align with the descriptions. Thus, annotators manually modify the text representation of the scene so that the scene fully aligns with the description.

Step 3: Verification. To ensure the quality of LEGO, we conduct manual verification. Specifically, annotators conduct mutual reviews of the annotated descriptions and corresponding scenes. Annotators first verify whether the constraints have been correctly identified and whether the annotations comply with the description annotation guidelines. The descriptions are re-annotated if the constraints are not accurately identified. Annotators also verify whether the generated scene fully aligns with the description. If misalignment is found, the annotators modify the scene to ensure the alignment. The procedure is repeated for two iterations.

A.3 Statistics

Figure 3 summarizes key statistics of our collected dataset. LEGO includes 130 natural language descriptions paired with manually annotated scenes, encompassing a total of 1,250 constraints. On aver-

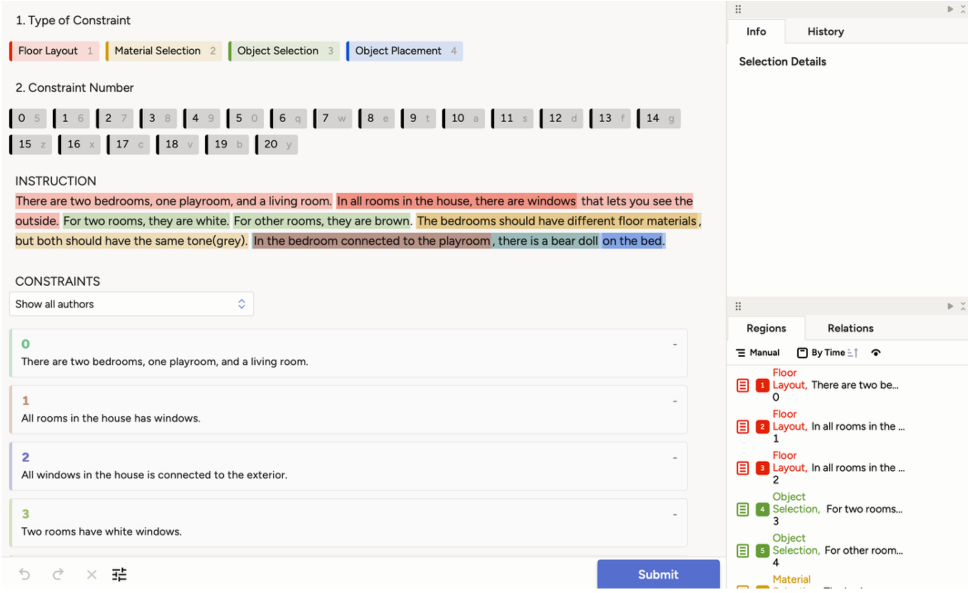

Figure 9: Interface of Label Studio for constraint classification.

Figure 10: Word cloud illustrating the diversity of instructions in LEGO.

age, each description specifies 9.6 constraints, with the majority falling between 9 and 11. These constraints span a broad range of scene elements: 55% involve objects, while 39% target architectural components. We further categorize constraints based on their semantics—approximately 40% relate to material and object selection, and the remaining 60% involve floor layout and object placement.

To illustrate the diversity of our collected descriptions, we utilize a word cloud representation in Figure 10. The visualization highlights the diverse linguistic forms and semantic categories present in the LEGO, indicating that the descriptions span across multiple domains and levels of abstraction.

Together, these distributions highlight the complexity and diversity of real-world environments captured by LEGO.

B LEGO-EVAL

We provide additional implementation details of LEGO-EVAL in the following.

B.1 Environment Setting

We use the AI2-THOR simulator (Kolve et al., 2017), which is built on the Unity engine, as our simulation environment. The simulator provides a flexible and realistic platform for embodied AI research. The environment assets are sourced from Objaverse (Deitke et al., 2023), ProcTHOR (Deitke et al., 2022), and AI2-THOR (Kolve et al., 2017), providing a wide range of object categories. Together, these assets enable comprehensive environment generation for our experiments.

B.2 Model Inference

We prompt the closed-source models, including gemini-2.5-pro, gpt-4.1, gpt-4.1-mini, and gpt-5.1 for both LLM and VLM inference. The input images are resized to 1200px resolution, and we use the default hyperparameter settings provided by the model for inference.

For the open-source model, including Gemma3-27B, Qwen2.5VL-32B, Qwen3-8B, and Qwen3-32B, the temperature is fixed at 0 to ensure deterministic outputs. The input images are resized to 335px resolution; when the number of images exceeds the model's processing capacity, two images are concatenated into a single input, with their corresponding names displayed at the top-left corner of each image.

Tool Types	Tools
Environment Interaction	`get topdown scene`, `get topdown room`, `get frontview object`, `get wall scene`, `get topdown object`, `get material image` `get multiview rendered object`, `get spatial relation`
Textual Reasoning	`get room list`, `get room info`, `get wall list`, `get wall info`, `get door list`, `get door info`, `get window list`, `get window info`, `get object list`, `get object info`
Multimodal Reasoning	`get object match`, `get property description`, `get property verification`

Figure 11: Diverse tools included in our tool set.

B.3 Metric Details

To aid readers' understanding, we provide detailed explanations of the evaluation metrics below.

- **F1 Score**: This metric evaluates the balance between precision and recall, serving as the harmonic mean of the two. A higher F1 score indicates better performance in correctly identifying positive instances while avoiding false positives and false negatives.

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1}$$

- **Cohen's Kappa** (κ): This metric measures the level of agreement between two raters, adjusting for the amount of agreement that could occur by chance. A higher κ value indicates stronger agreement, where $\kappa = 1$ denotes perfect agreement and $\kappa = 0$ corresponds to chance-level agreement. We use Cohen's kappa to measure the agreement with human judgments.

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{2}$$

  where p_o denotes the observed agreement between the two raters, and p_e denotes the expected agreement under the assumption of statistical independence between the raters.

B.4 Tool Set

As shown in Figure 11, LEGO-EVAL utilizes total 21 tools for robust evaluation. In this section, detailed descriptions for each tool are provided. For tools that output images, example outputs of the images for each tool are also provided.

B.4.1 Environment Interaction

- **get topdown scene:** This tool generates a bird's-eye (top-down) view of the environment. It produces a dictionary containing an image array that represents the entire scene layout, including walls, doors, and objects. The example output is shown in Figure 16.

- **get topdown room:** This tool generates a top-down visual representation of a specific room in the environment. By providing the room ID as input, it produces a structured dictionary containing an image array that depicts the layout of the room, including walls, doors, and objects. The example output is shown in Figure 17.

- **get frontview object:** This tool produces front-view images of specified objects within a scene. Given a list of object IDs, it centers each target object in the frame while preserving surrounding context, such as nearby objects, for richer spatial awareness. The output is a structured dictionary mapping each object ID to its corresponding front-view image array. The example output is shown in Figure 18.

- **get wall scene:** This tool generates clear wall-view images of a room from specified directions. By taking a list of wall IDs as input, it produces perspective images where objects in the middle of the room are removed to prevent occlusion, ensuring unobstructed visibility of the walls. The output is a dictionary mapping each wall to its corresponding image array. The example output is shown in Figure 19.

- **get topdown object:** This tool produces object-centric images from an overhead perspective. Given a list of object IDs, it generates top-down views where each target object is centered in the frame, while preserving surrounding context such as nearby objects. The resulting dictionary maps each object to image arrays, allowing inspection at different distances. The example output is shown in Figure 20.

- **get material image:** This tool generates visual representations of specified materials, restricted to walls and floors within the scene. By providing a list of material names, it returns a dictionary mapping each material to

an RGBA image array. The tool only includes materials that are actually present in the designated environment. The example output is shown in Figure 21.

- **get multiview rendered object:** This tool produces rendered images of specified objects using their underlying asset IDs. Given a list of object IDs, it returns a dictionary where each object ID maps to an image array showing the rendered appearance of that object. The example output is shown in Figure 22.

- **get spatial relation:** This tool generates scene images that isolate and display only specified object pairs (or sets), such as objects, windows, doors, or walls. By taking a list of tuples containing object IDs, it produces a dictionary mapping each group of objects to an image array. Values of the dictionary provide visualizations that highlight the spatial relationships among the selected elements. The example output is shown in Figure 23.

B.4.2 Multimodal Reasoning

- **get object match:** This tool provides semantic descriptions for object IDs by mapping them to their corresponding object types. It is designed to be used after retrieving rendered views with `get multiview rendered object`, ensuring accurate alignment between visual assets and their categorical labels. The output is a dictionary that links each object ID to a descriptive type.

- **get property description:** This tool generates semantic descriptions of objects by identifying their color, shape, and material from rendered images and optional textual metadata. It is intended for use after retrieving visual inputs with `get material image` and `get multiview rendered object`, ensuring accurate grounding of appearance-based attributes. The output is a dictionary summarizing the derived text information, along with reasoning.

- **get property verification:** This tool validates descriptive attributes of a subject—either an object or material—based on a given instruction. First, an LLM parses the instruction to determine which attributes should be checked (e.g., surface patterns or textures). Then, a VLM analyzes the corresponding images to extract those attributes. It is designed for use after obtaining visuals with `get material image` and `get multiview rendered object`. The output is a dictionary summarizing the verified textual information.

B.4.3 Textual Reasoning

- **get room list:** This tool extracts all room identifiers available within a given scene. It returns a dictionary which contains a list of room IDs as strings.

- **get room info:** This tool provides detailed metadata for specific rooms in a scene. Given a dictionary from `get room list` containing room IDs, it returns structured information about each room. The output includes the polygon vertices defining the room's geometry and the identified floor material.

- **get wall list:** This tool extracts the wall identifiers associated with specific rooms in a scene. Given a list of room IDs, it returns a dictionary mapping each room ID to a list of its connected wall IDs.

- **get wall info:** This tool provides comprehensive metadata for specified walls in a scene. Using a list of wall IDs (retrieved via `get wall list`), it returns the values including details such as the wall's unique ID, connected room IDs, geometric coordinates, material type, width, height, and directional orientation.

- **get door list:** This tool extracts the door identifiers connected to specific rooms within a scene. Given a list of room IDs, it returns a dictionary mapping each room ID to its associated door IDs.

- **get door info:** This tool provides detailed metadata for doors within a scene. Using a list of door IDs (obtained via `get door list`), it returns the values capture attributes such as the door's unique ID, associated asset ID, connected rooms, adjoining walls, 3D position, and openness state.

- **get window list:** This tool extracts the window identifiers associated with specific rooms in a scene. Given a list of room IDs, it returns a dictionary mapping each room ID to its corresponding window IDs.

Methods	Holistic SR				Partial SR			
	M1	M2	M3	M4	M1	M2	M3	M4
Oracle Constraint	0.12	0.05	0.05	0.14	0.66	0.57	0.35	0.65
Identified Constraint	0.13	0.07	0.05	0.12	0.63	0.57	0.35	0.64
Difference in SR	+0.01	+0.02	+0.00	-0.02	-0.03	+0.00	+0.00	-0.01

Table 6: End-to-end evaluation maintains consistent results with annotated constraint evaluation.

- **get window info:** This tool provides detailed metadata for windows in a scene. Given a list of window IDs (retrieved via `get window list`), it outputs the values include attributes such as the window's unique ID, associated asset ID, connected rooms, adjoining walls, and 3D position.

- **get object list:** This tool extracts the identifiers of objects contained within specific rooms of a scene. Given a list of room IDs, it returns a dictionary mapping each room ID to the corresponding list of object IDs present in that room.

- **get object info:** This tool provides detailed metadata about objects in a scene. Using a list of object IDs (retrieved via `get object list`), it returns the values include attributes such as the object's unique ID, associated asset ID, room assignment, 3D position, rotation, and geometric representation through coordinates or mesh vertices.

C Additional Results

We provide a series of analyses on LEGO-EVAL to better understand its behavior.

C.1 LEGO-EVAL provides end-to-end automated evaluation.

To demonstrate the effectiveness of LEGO-EVAL, we show that it performs reliably even without human-annotated constraints. Accurate, fine-grained evaluation requires capturing all relevant constraints in a given description, so we compare LEGO-EVAL's performance using its own automatically identified constraints against its performance with human-annotated ones. In our setup, constraints are extracted and categorized using GPT-4.1, then used to evaluate 3D scene generation outputs from Qwen2.5-VL-32B. These evaluations are directly compared with those conducted using human-annotated constraints. As shown in Table 6, LEGO-EVAL exhibits only minor performance differences across four 3D scene synthesis methods when using automatic versus human-provided constraints. This close alignment demonstrates that LEGO-EVAL can reliably identify and classify constraints, making it suitable as a fully automated, end-to-end evaluation tool.

C.2 LEGO-EVAL plans tool execution effectively.

We analyze how two core components of the evaluation process—tool execution planning and argument selection—contribute to overall performance, using all constraints introduced in Section 5.1. For tool planning, LEGO-EVAL generates execution plans conditioned on the given constraints. For argument selection, LEGO-EVAL selects arguments given a human-annotated tool execution plan. We test multiple LLMs for each component to observe their contributions, using Qwen2.5VL-32B as a fixed validator. We use three metrics to assess component performance: (1) Tool F1, the macro F1 score over constraints, where a tool prediction is correct if it appears in the ground-truth set; (2) Argument F1, the macro F1 over tools, with correctness based on ground-truth arguments; and (3) Graph Edit Distance (GED), measuring the number of edits needed to align predicted and ground-truth constraint graphs. Results in Table 7 show that tool execution planning correlates more strongly with overall evaluation performance than argument selection, highlighting its role as an orchestrator. When tool plans are fixed to ground truth, argument selection correlates more with performance, showing its dependence on accurate planning (See Section C.3). These results suggest that LEGO-EVAL handles tool planning well, which in turn supports accurate argument selection and evaluation.

Models	Component Performance			Evaluation Performance	
	Tool F1 ↑	GED ↓	Argument F1 ↑	Holistic F1 ↑	Partial F1 ↑
Gemma3-27B	0.50	3.01	0.38	0.61	0.68
Qwen2.5-VL-32B	0.57	2.87	0.46	0.64	0.72
Qwen3-32B	0.62	2.55	0.42	0.69	0.73

Table 7: The performance of components are highly correlated with the evaluation performance. Tool F1 and argument F1 measure prediction accuracy; GED measures graph structural similarity.

Models	Argument Selection		
	Arg. F1 ↑	Hol. F1 ↑	Part. F1 ↑
Gemma3-27B	0.38	0.65	0.66
Qwen2.5VL-32B	0.46	0.68	0.68
Qwen3-32B	0.42	0.68	0.67

Table 8: Evaluation performance of models on argument selection.

C.3 Argument selection plays a critical role in evaluation performance.

To further analyze the contribution of argument selection in our evaluation pipeline, we conduct an experiment where the tool sequence is fixed to human-annotated ground truth, and models are only responsible for selecting appropriate arguments for each tool. All other components of LEGO-EVAL, including tool execution and validation, use Qwen2.5-VL-32B as the backbone.

We evaluate three open-source models Gemma3-27B, Qwen2.5-VL-32B, and Qwen3-32B on their ability to perform argument selection. As shown in Table 8, Qwen2.5-VL-32B achieves the highest Argument F1 score, aligning with superior holistic and partial evaluation performance. Qwen3-32B also performs competitively, while Gemma3-27B lags behind across all metrics.

These results highlight that accurate argument selection is crucial for reliable evaluation. Since argument identification directly determines the quality of evidence retrieved by tools, improvements in this component significantly enhance both holistic and partial agreement with human judgments.

C.4 Tool usage distributions differ across constraint types.

Extending the analysis in Section 5.1.3, we examine which tool types are used in LEGO-EVAL across different constraint categories. As shown in Figure 12, environment interaction tools and textual reasoning tools constitute the majority of tool usage, indicating that most constraints require both visual and textual information from the generated scene for evaluation. In contrast, multimodal reasoning tools are used less frequently on average. However, their usage becomes substantially higher for object placement constraints, where they account for approximately 40% of all tool calls. This increase is primarily driven by the need for object identification and attribute verification, which rely on aligning object descriptions with visual observations. These results suggest that different constraint types require distinct combinations of tool capabilities, highlighting the importance of supporting diverse tool modalities for 3D scene and instruction alignment evaluation.

C.5 Error Analysis of VLM-as-a-Judge with Different Modules.

We provide details of the experiment conducted in Section 6.1. We conduct a error analysis of VLM-as-a-Judge using different perception modules. Specifically, we compare the vanilla VLM-as-a-Judge with three variants that incorporate additional scene understanding modules: Grounding DINO, LASER, and Point-LLM. We construct a fixed evaluation set by randomly sampling 50 instruction–scene pairs with aligned instructions and 50 with misaligned instructions from the evaluation dataset from Section 5.1, resulting in 100 total cases. As shown in Table 5, the vanilla VLM-as-a-Judge exhibits the highest number of identification and attribute errors. Although Grounding DINO is used to localize scene components mentioned in the instruction by adding bounding boxes to the input image, it unexpectedly increases identification errors. Similarly, incorporating LASER to generate scene graphs as additional inputs does not improve performance and instead leads to more errors across all categories compared to the vanilla setting. Finally, we convert scenes into point clouds and feed them into Point-LLM, which slightly re-

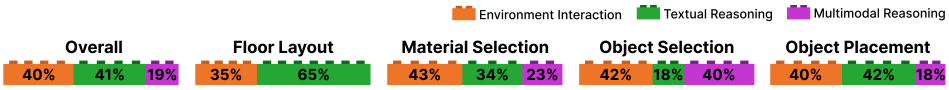

Figure 12: Distribution of tool types executed by LEGO-EVAL during evaluation.

duces spatial errors but increases both identification and attribute errors. Detailed descriptions of each method are provided in Appendix D.1.2.

C.6 Human usually give long descriptions to describe the room.

We verify whether people actually provide descriptions to an LLM for room generation that are as long as we claimed. Therefore, as shown in Figure 13, we conduct a survey where participants are asked to describe a room photo in the form of a prompt. The results show that, on average, the scene descriptions contain 18.2 constraints. This finding confirms that people indeed provide complex descriptions, and highlights the importance of LEGO-EVAL, which can robustly evaluate generated rooms based on such descriptions.

Figure 13: Our survey examples for asking human to describe the room.

D Method Details

In this section, we describe in detail the methods employed for scene evaluation and scene generation in our experiments.

D.1 Scene Evaluation

D.1.1 Overview of Existing Evaluation Methods

VLM-as-a-judge prompts VLM to evaluate the generated scene aligning with the instruction. Scene images from four perspectives are provided as model input for fair evaluation. The example input images are shown in Figure 14.

CLIPSCORE (Hessel et al., 2021) leverages pre-trained vision-language models to assess the semantic alignment between generated captions and reference images. Instead of relying on token-level matching or human references, CLIPSCORE computes similarity in the joint embedding space of CLIP. It performs binary judgment on the instruction and top-down scene image using thresholds of 15, 20, and 25. The example input images are shown in Figure 15.

SceneEval (Tam et al., 2025) introduces an evaluation framework for text-conditioned 3D indoor scene synthesis. Unlike prior metrics that primarily

measure realism or distributional similarity, SceneEval directly evaluates how generated scenes satisfy both explicit user requirements (e.g., object counts, attributes, and spatial relationships) and implicit expectations (e.g., absence of collisions, navigability, accessibility). To support evaluation, the authors release SceneEval-500, a dataset of scene descriptions with annotated ground-truth properties. Together, these metrics provide a comprehensive assessment of fidelity and plausibility in 3D scene generation.

D.1.2 VLM-as-a-Judge with Different Modules

Grounding DINO (Liu et al., 2023) is an open-set object detection model that extends the DINO detector with language grounding capabilities. It takes as input an image and a free-form text prompt describing target objects or scene elements, and outputs bounding boxes corresponding to regions that match the given text. In this work, we use Grounding DINO to localize scene elements mentioned in the instruction within the input scene images. Specifically, we apply it to detect and ground relevant objects, draw bounding boxes on the image, and then feed the annotated image into VLM-as-a-judge.

LASER (Huang et al., 2025) is a neuro-symbolic framework that learns spatial-temporal scene graphs from visual inputs under weak supervision. It constructs structured representations that encode objects, their attributes, and their relationships within a scene, enabling explicit reasoning over spatial and temporal dependencies. In our setting, LASER is used to generate scene graphs for each individual room in a multi-room environment. We then provide the scene graphs from all rooms, together with the scene images, as additional inputs to the VLM-as-a-Judge.

Point-LLM (Guo et al., 2023) is a framework that enables large language models to understand and reason over 3D point cloud data. It encodes point cloud inputs into representations that can be processed by LLMs, allowing the model to perform spatial reasoning in 3D space. In our approach, we first convert the scene graphs into point cloud representations. We then uniformly downsample the point clouds to match the input size requirements. Finally, we use Point-LLM to evaluate the resulting 3D scene representations.

Figure 14: Example image input for VLM-as-a-judge

D.2 Scene Synthesis

D.2.1 Overview of Scene Synthesis Methods

LayoutGPT (Feng et al., 2023) introduces a training-free framework that leverages large language models as visual planners for layout-based generation. LayoutGPT composes in-context visual demonstrations in CSS-like structures to inject visual commonsense into LLMs. This enables accurate translation of challenging linguistic concepts—such as numerical and spatial reasoning—into 2D image layouts and 3D indoor scene arrangements.

I-Design (Çelen et al., 2025) introduces a personalized framework for 3D indoor scene synthesis driven by natural language input. I-Design employs a team of large language model agents to interpret unstructured user descriptions and reason about object selection, style, and spatial relationships. These preferences are represented as scene graphs, which are then transformed into complete room layouts through a backtracking placement algorithm and enriched with 3D assets retrieved from object databases. The system outputs interpretable, editable pipelines including scene graphs, floor plans, and rendered views, enabling flexible, user-centered interior design exploration.

Holodeck (Yang et al., 2024b) introduces a controllable simulation platform for embodied agents in diverse 3D environments. Unlike prior systems that often limit interaction fidelity or domain variety, Holodeck enables agents to operate within

Figure 15: Example image input for CLIPSCORE

photorealistic virtual worlds featuring rich physics, object manipulation, and dynamic scenarios. The platform provides flexible interfaces for integrating natural language commands, sensory input, and reinforcement learning frameworks, making it suitable for studying grounded reasoning and task execution.

LayoutVLM (Sun et al., 2025) introduces a framework for open-universe 3D layout generation guided by natural language instructions. Unlike prior approaches that either predict precise object poses or solve rigid constraint systems, LayoutVLM combines numerical pose estimates and spatial relations within a differentiable optimization process to achieve physically plausible and semantically coherent layouts. The method leverages vision-language models with visual prompting and a self-consistent decoding procedure to generate scene layout representations from unlabeled 3D assets and rendered images.

MANSION (Che et al., 2026) introduces a framework for generating multi-floor 3D indoor environments from natural language instructions. Unlike prior work that focuses on single-floor scene synthesis or local object arrangement, MANSION explicitly models hierarchical building structures to ensure global spatial consistency across rooms and floors. It generates structurally valid and navigable environments while preserving inter-floor connectivity and semantic coherence, and further supports large-scale procedural generation with open-vocabulary scene customization for embodied AI evaluation.

D.2.2 Implementation of Scene Synthesis Methods

Scene layout generation. I-Design, LayoutVLM, and LayoutGPT do not generate structural components such as walls or floors. Therefore, we use Holodeck to generate the basic room layout (i.e., walls and floors), and the compared methods are applied solely for object selection and/or object placement within each room.

Additionally, LEGO-EVAL currently supports only square or rectangular room layouts. To adhere to this evaluation setting, we modify the generation pipeline of MANSION so that it produces only rectangular or square scene layouts.

Object Selection. For LayoutVLM, although its supplementary material provides prompts for object selection, these prompts are designed for test-time data generation and are not part of the core LayoutVLM framework. Moreover, LayoutVLM relies on a human-in-the-loop procedure for object selection, making it incompatible with fully automated benchmarking. Therefore, we do not use these prompts. Instead, we provide the model with object names and asset UIDs selected by Holodeck.

For LayoutGPT, the model selects objects to place based on a given prompt; however, it is restricted to a fixed set of 21 object categories. In contrast, LEGO covers 51.7K object categories, which cannot be enumerated in a single prompt. To address this, we include only the categories selected by Holodeck in the prompt, allowing LayoutGPT to choose objects from this reduced set.

E Examples of Tool Image Outputs

Figure 16: Example Image Output of get topdown scene.

Figure 18: Example Image Output of get frontview object.

Figure 17: Example Image Output of get topdown room.

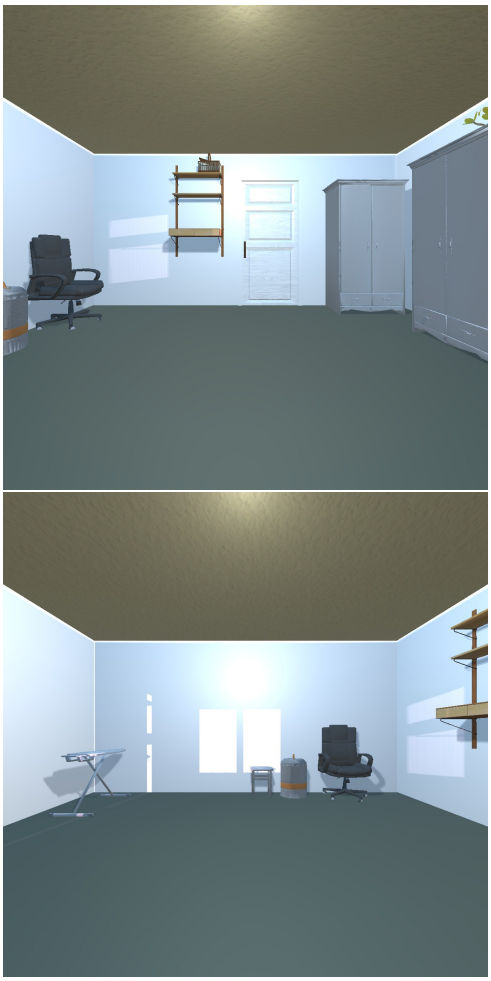

Figure 19: Example Image Output of get wall scene.

Figure 20: Example Image Output of get topdown object.

Figure 21: Example Image Output of get material image.

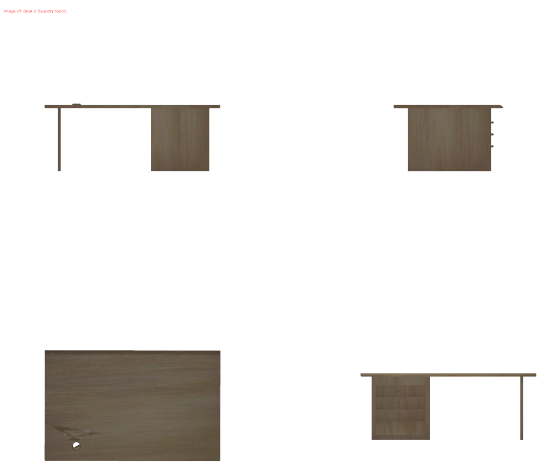

Figure 22: Example Image Output of get multiview rendered object.

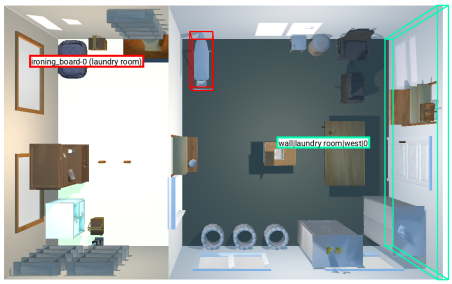

Figure 23: Example Image Output of get spatial relation.

F Prompts used in our Works

Prompts

You are an argument selector of the tools. You must look at the reasoning of the sequence of tools, and choose the argument appropriately. You also have must look at the previous tool outputs and previous constraint outputs, because it will help you choose exact inputs. The output should be a list, containing roomId(str) as elements. You must use the FULL, WHOLE ID of the room as arguments, NOT just partial. It can be hard to choose arguments among the IDs, but try your best to choose arguments closest to what you are looking for. If you really can't choose arguments, return empty list.

Tools

[Available Tool List & Definition]

Instructions

* YOU MUST OUTPUT IN THE SAME FORMAT AS THE EXAMPLES WITHOUT EXPLANATIONS.
Example:
  Chain-of-Thought:
  Arguments:
* You must use the FULL, WHOLE ID of the room as arguments, NOT just partial.
* If the exact room isn't found in the list, select the closest matching room.
* Even if the exact room is found, also checking the closest matching rooms is highly recommended.

Examples

[Example Responses for the Argument Selection of Type 1]

<Information_for_Argument_Selection id="your-turn">
User Instruction: $INSTRUCTION$
Previous Constraint Outputs: $PREVIOUS_CONSTRAINT_OUTPUTS$
Current Constraint: $CURRENT_CONSTRAINT$
Tool Sequence: $TOOL_SEQUENCE$
Reasoning behind Tool Sequence: $REASONING$
Previous Tool Outputs: $PREVIOUS_TOOL_OUTPUTS$
Current Tool to Use: $TOOL_TO_USE$
</Information_for_Argument_Selection>

<assistant_response id="your-turn">
<!-- Assistant will fill in Chain-of-Thought and Arguments here -->
</assistant_response>

Figure 24: Prompt used for Argument Selection in scene-component ID list generation tools.

Prompts

You are an argument selector of the tools. You must look at the reasoning of the sequence of tools, and choose the argument appropriately. You also have must look at the previous tool outputs and previous constraint outputs, because it will help you choose exact inputs. The output should be a list, containing Id(str) of rooms, doors, windows, walls, or objects as elements. You must use the FULL, WHOLE ID of the room, door, window, wall, or object as arguments, NOT just partial. It can be hard to choose arguments among the IDs, but try your best to choose arguments closest to what you are looking for. If you really can't choose arguments, return empty list.

Tools

[Available Tool List & Definition]

Instructions

* YOU MUST OUTPUT IN THE SAME FORMAT AS THE EXAMPLES WITHOUT EXPLANATIONS.
Example:
  Chain-of-Thought:
  Arguments:
* You must use the FULL, WHOLE ID of the room, door, window, wall, or object as arguments, NOT just partial.
* If the exact room, door, wall, or window isn't found in the list, select the closest match or use all rooms, doors, walls, or windows. Even if found, also check nearby matches.
* Likewise, for objects, check synonyms, related categories, and functional equivalents. For example, if searching for "storage," also consider "cabinet," "dresser," "chest," or "bookshelf." Always include closely related categories for the best match.
* When identifying objects that satisfy a constraint, always consider all items with similar or related names (e.g., 'desk-0', 'desk-1', 'desk-2') even if the constraint mentions only a subset. Do not assume that only the explicitly named items are relevant. Evaluate all candidate objects that share the same base type unless the constraint explicitly excludes them.

Examples

[Example Responses for the Argument Selection for Type 2]

Figure 25: Prompt used for argument selection in scene-component information retrieval tools.

```
<Information_for_Argument_Selection id="your turn">
User Instruction: $INSTRUCTION$
Previous Constraint Outputs: $PREVIOUS_CONSTRAINT_OUTPUTS$
Current Constraint: $CURRENT_CONSTRAINT$
Tool Sequence: $TOOL_SEQUENCE$
Reasoning behind Tool Sequence: $REASONING$
Previous Tool Outputs: $PREVIOUS_TOOL_OUTPUTS$
Current Tool to Use: $TOOL_TO_USE$
</Information_for_Argument_Selection>

<assistant_response id="your-turn">
<!-- Assistant will fill in Chain-of-Thought and Arguments here -->
</assistant_response>
```

Figure 26: Prompt used for argument selection in scene-component information retrieval tools.

Prompts

You are an argument selector of the tools. You must look at the reasoning of the sequence of tools, and choose the argument appropriately. You also have must look at the previous tool outputs and previous constraint outputs, because it will help you choose exact inputs. The output should be a list, containing Id(str) of rooms or walls, or name of material as elements. You must use the FULL, WHOLE ID of the room, wall, or material name as arguments, NOT just partial. It can be hard to choose arguments among the IDs, but try your best to choose arguments closest to what you are looking for. If you really can't choose arguments, return empty list.

Tools

[Available Tool List & Definition]

Instructions

* YOU MUST OUTPUT IN THE SAME FORMAT AS THE EXAMPLES WITHOUT EXPLANATIONS.
Example:
  Chain-of-Thought:
  Arguments:
* You must use the FULL, WHOLE ID of the room, wall, or material name as arguments, NOT just partial.
* If the exact room or wall isn't found in the list, select the closest matching ones.
* Even if the exact room or wall is found, also checking the closest matching rooms or walls is highly recommended.

Examples

[Example Responses for the Argument Selection of Type 3]

<Information_for_Argument_Selection id="your-turn">
User Instruction: $INSTRUCTION$
Previous Constraint Outputs: $PREVIOUS_CONSTRAINT_OUTPUTS$
Current Constraint: $CURRENT_CONSTRAINT$
Tool Sequence: $TOOL_SEQUENCE$
Reasoning behind Tool Sequence: $REASONING$
Previous Tool Outputs: $PREVIOUS_TOOL_OUTPUTS$
Current Tool to Use: $TOOL_TO_USE$
</Information_for_Argument_Selection>

<assistant_response id="your-turn">
<!-- Assistant will fill in Chain-of-Thought and Arguments here -->
</assistant_response>

Figure 26: Prompt used for Argument Selection in scene-component visual rendering tools.

Prompts

You are an argument selector of the tools. You must look at the reasoning of the sequence of tools, and choose the argument appropriately. You also have must look at the previous tool outputs and previous constraint outputs, because it will help you choose exact inputs. The output should be a list, containing object Id(str) as elements. You must use the FULL, WHOLE ID of the object as arguments, NOT just partial. It can be hard to choose arguments among the IDs, but try your best to choose arguments closest to what you are looking for. If you really can't choose arguments, return empty list.

Tools

[Available Tool List & Definition]

Instructions

* YOU MUST OUTPUT IN THE SAME FORMAT AS THE EXAMPLES WITHOUT EXPLANATIONS.
Example:
  Chain-of-Thought:
  Arguments:
* You must use the FULL, WHOLE ID of the object as arguments, NOT just partial.
* If the exact object isn't found in the list, select the closest match or check nearby matches. Do not use every single object in the room or scene as arguments.
* Check synonyms, related categories, and functional equivalents. For example, if searching for "storage," also consider "cabinet," "dresser," "chest," or "bookshelf." Always include closely related categories for the best match.
* When identifying objects that satisfy a constraint, always consider all items with similar or related names (e.g., 'desk-0', 'desk-1', 'desk-2') even if the constraint mentions only a subset. Do not assume that only the explicitly named items are relevant. Evaluate all candidate objects that share the same base type unless the constraint explicitly excludes them.

Examples

[Example Responses for the Argument Selection of Type 4]

Figure 27: Prompt used for Argument Selection in object-level visual rendering tools.

```
<Information_for_Argument_Selection id="your-turn">
User Instruction: $INSTRUCTION$
Previous Constraint Outputs: $PREVIOUS_CONSTRAINT_OUTPUTS$
Current Constraint: $CURRENT_CONSTRAINT$
Tool Sequence: $TOOL_SEQUENCE$
Reasoning behind Tool Sequence: $REASONING$
Previous Tool Outputs: $PREVIOUS_TOOL_OUTPUTS$
Current Tool to Use: $TOOL_TO_USE$
</Information_for_Argument_Selection>

<assistant_response id="your-turn">
<!-- Assistant will fill in Chain-of-Thought and Arguments here -->
</assistant_response>
```

Figure 28: Prompt used for Argument Selection in object-level visual rendering tools.

Prompts

You are an argument selector of the tools. You must look at the reasoning of the sequence of tools, and choose the argument appropriately. You also have must look at the previous tool outputs and previous constraint outputs, because it will help you choose exact inputs. The output should be a list, containing tuples as elements. The tuples must contain two object Ids(str). You must use the FULL, WHOLE ID of the object, window, or door as arguments, NOT just partial. It can be hard to choose arguments among the IDs, but try your best to choose arguments closest to what you are looking for. If you really can't choose arguments, return empty list.

Tool

[Available Tool List & Definition]

Instructions

* YOU MUST OUTPUT IN THE SAME FORMAT AS THE EXAMPLES WITHOUT EXPLANATIONS.
Example:
  Chain-of-Thought:
  Arguments:
* You must use the FULL, WHOLE ID of the object, window, door, wall as arguments, NOT just partial.
* If the exact object isn't found in the list, select the closest match. Even if found, also check nearby matches.
* Check synonyms, related categories, and functional equivalents. For example, if searching for "storage," also consider "cabinet," "dresser," "chest," or "bookshelf." Always include closely related categories for the best match.
* You are not limited to including only two objects, walls, doors, or windows in a tuple. You can include as many as needed. Grouping multiple IDs into a single tuple helps reduce the number of images generated. If you want to visualize multiple relationships in one image, make sure to include all relevant IDs together in the same tuple. Example: [('wall1', 'wall2', 'desk2')]

Examples

[Example Responses for the Argument Selection of Type 5]

Figure 28: Prompt used for Argument Selection in the spatial-relation visualization tool.

```
<Information_for_Argument_Selection id="your-turn">
User Instruction: $INSTRUCTION$
Previous Constraint Outputs: $PREVIOUS_CONSTRAINT_OUTPUTS$
Current Constraint: $CURRENT_CONSTRAINT$
Tool Sequence: $TOOL_SEQUENCE$
Reasoning behind Tool Sequence: $REASONING$
Previous Tool Outputs: $PREVIOUS_TOOL_OUTPUTS$
Current Tool to Use: $TOOL_TO_USE$
</Information_for_Argument_Selection>

<assistant_response id="your-turn">
<!-- Assistant will fill in Chain-of-Thought and Arguments here -->
</assistant_response>
```

Figure 29: Prompt used for Argument Selection in the spatial-relation visualization tool.

Prompts

You are a constraint classification engine.

Your task is to classify a given Constraint into one of the following types within the Instruction:
• Floor Layout: Describes the spatial configuration or existence of architectural elements such as rooms, walls, doors, and windows. (overall layout)

• Material Selection: Describes visual or physical surface characteristics (e.g., color, texture, material) of structural elements of walls or floors. Consider only the walls and floors, not the other objects.

• Object Selection: Describes what an object looks like (e.g., shape, style, color, material) — applies to furniture or decorations.

• Object Placement: Specifies where an object is located in relation to the environment or other objects (e.g., "on the table", "next to the sofa", "in the corner of the room").

Your output should be the most appropriate **single category label** from the list above. Respond only with the label.

[Example Responses for the Constraint Classification]

##Output
Instruction: {instruction}
constraint: {constraint}
Output:

Figure 29: Prompt used for Constraint Classification.

Prompts

You are an expert of extracting constraints from the given instruction.

Identity
Your task is to decompose a given instruction into a list of constraints. These constraints are decomposed for the purpose of validation. Hence, you must consider the order of constraints. Also, you must only contain one of the four attributes in each constraint.

Attributes:
1. Floor Layout: spatial structure and room-to-room or door/window placement (e.g., room existence, connections, relative size, or doorway/window layout)
2. Material Selection: materials, textures, and colors of walls, floors, or ceilings
3. Object Selection: object attributes such as color, shape, size, or style (not position)
4. Object Placement: spatial relationships and placement of objects within rooms or relative to each other

Rules:
1. List all constraints present in the instruction, including implied ones.
 - Example: "There is one wall that has windows. The wall that has windows is yellow."
  → (1) There is one wall that has windows.
  → (2) The wall that has windows is yellow.

2. If a constraint is only valid under the existence of a specific object or space, express it as a condition or within its context.
 - Example: "There is a room with red floor. In that room, there is a white bed."
  → (1) There is a room with red floor.
  → (2) In the room with red floor, there is a white bed.

3. Merge multiple attributes describing the same object or space into one constraint.
 - Example: "The three walls are yellow. The three walls have patterns."
  → (1) The three walls in the bedroom are yellow and have patterns.
 - Example: "The table is large. The table is red."
  → (1) The table is large and red.

4. Group mentions of rooms into one unified constraint.
 - Example: "There is bedroom and bathroom in the house. There is living room in the house."
  → (1) There is bedroom, bathroom, and one living room in the house.

5. Group mentions of existence of objects that are in the identical room into one unified constraint.
 - Example: "In the bedroom, there is a bed. In that room, there is also a table."
  → (1) In the bedroom, there is a bed and a table.

Figure 30: Prompt used for Constraint Classification.

6. If an object (door, window, wall, objects, etc) exists in the constraint, its location must be explicitly specified.
- Example: "There is bedroom in the house. The wall is white. There a toy car in the bedroom. The car is big."
  → (1) There is bedroom in the house.
  → (2) The wall in the bedroom is white.
  → (3) There a toy car in the bedroom.
  → (4) The car in the bedroom is big.

7. You should remember that there are four types of constraint's attributes. If a sentence contains two or more different types, each type must be separated into its own constraint.
- Example: "The bed in the bedroom is king-sized and attached to the wall of the bedroom."
  → (1) The bed in the bedroom is king sized.
  → (2) The bed in the bedroom is attached to the wall of the bedroom.

7-1. Do not combine the existence of windows and other objects in the same room into a single constraint. Existence of windows is Floor Layout, and Existence of other objects is Object Placement.
- Example: "The room has a window. The room has a desk and a chair."
  → (1) The room has a window.
  → (2) The room has a desk and a chair.

8. Do not create constraints for the mere existence of walls or a floor. The existence of walls and a floor is assumed for any room. Only create constraints for specific materials, textures, and colors of walls or a floor
- Example: "The room has black walls and a beige wooden floor."
  → (X)(Do NOT Create) The room has walls and a floor.
  → (X)(Do NOT Create) The room has walls.
  → (X)(Do NOT Create) The room has a floor.
  → (O)(Create like this) The room has black walls and a beige wooden floor.

[Example Responses for the Constraint Identification]

Input:
Instruction:
{instruction}

Output:
Constraints:
1. <constraint 1>
2. <constraint 2>
...
n. <constraint n>

Figure 31: Prompt used for Constraint Identification.

Prompts

You are an expert in evaluating the performance of a scene generator. Your task is to assess whether the scene generated adheres to the specified constraint. Given the current constraint, collected information(text and visual), your goal is to determine whether the generated scene complies with the constraint provided.

Identity
You must determine whether the generated scene complies with the provided constraint. Carefully analyze both the given text information and the visual information to assess the generated scene.

Instructions
* You need to provide two assessments in evaluation: 1. Evaluation and 2. Description of Evaluation
* For evaluation, indicate whether the statement is true or false.
* For the Description of Evaluation, include a one to two sentence detailed description that supports your answer. This description will be used for the next constraint assessment.
* Do not assess the instruction itself. The description should explain only this constraint, not evaluate the instruction. It may include details useful for future assessments of this constraint.
* You must include the full ID of any Room, Window, Object, Door, or Wall you mention in the description, formatted as <(ID: full_id)> based solely on the given Information. Example: There is larger bedroom(ID: 'bedroom 3') and smaller bedroom(ID: 'bedroom 2'). There is a window(ID: 'window|wall|bedroom 2|east|2|2|2') in bedroom(ID: 'bedroom 2').
* Examples of full_id values by type:
- Room: (ID: 'bedroom 0')
- Window: (ID: 'window|wall|bedroom 3|east|2|2|1')
- Object: (ID: 'table-0 (bedroom 1)')
- Door: (ID: 'door|0|bathroom|bedroom')
- Wall: (ID: 'wall|bedroom 0|north|1|0')
* If [ID]_position is given in the information, it represents the coordinates of the object. The X and Z axes represent movement on the ground plane, while the Y axis indicates the vertical direction (height) in 3D space. Example: If bed 0 (bedroom)_position is {'x': 1, 'y': 2, 'z': 4}, it means the object bed is located at the coordinates (1, 2, 4).
* Evaluation must be in the format below:
  If the scene complies with the constraint:
    Evaluation format: <<True, [One to two sentence detailed description of the answer]>>
  If the scene does not comply with the constraint:
    Evaluation format: <<False, [One to two sentence detailed description of the answer]>>

Figure 31: Prompt used in Constraint Validation for Floor Layout constraints.

* All rooms, walls, windows, objects, doors' Id is unique. The assetId of windows, objects, doors can be the same.
* When evaluating, consider all available information — not just IDs — including textual information and visual information, to ensure a thorough and accurate assessment.
* Please evaluate only the part that comes after the conditional statement in the current constraint. The condition (e.g., "in the room with 3 dolls") has already been evaluated. Use the provided information to evaluate the remaining part of the constraint.$VISUAL_ADDITIONAL_INSTRUCTION$

Examples

[Example Responses for Validation]

Output
<Information_for_Validation id="your-turn">
Instruction: $INSTRUCTION$
Current Constraint: $CURRENT_CONSTRAINT$
Information: $INFORMATION$
</Information_for_Validation>

<assistant_response id="your-turn">
Chain-of-Thought:
Evaluation:
</assistant_response>

Figure 32: Prompt used in Constraint Validation for Floor Layout constraints.

Prompts

You are an expert in evaluating the performance of a scene generator. Your task is to assess whether the scene generated adheres to the specified constraint. Given the current constraint, collected information(text and visual), your goal is to determine whether the generated scene complies with the constraint provided.

Identity
You must determine whether the generated scene complies with the provided constraint. Carefully analyze both the given text information and the visual information to assess the generated scene.

Instructions
* You need to provide two assessments in evaluation: 1. Evaluation and 2. Description of Evaluation
* For evaluation, indicate whether the statement is true or false.
* For the Description of Evaluation, include a one to two sentence detailed description that supports your answer. This description will be used for the next constraint assessment.
* Do not assess the instruction itself. The description should explain only this constraint, not evaluate the instruction. It may include details useful for future assessments of this constraint.
* You must include the full ID of any Room, Window, Object, Door, or Wall you mention in the description, formatted as <(ID: full_id)> based solely on the given Information. Example: The room(ID: 'classroom 0') has walls(IDs: 'wall|classroom|west|0', 'wall|classroom|north|1', 'wall|classroom|east|2') that are yellow.
* Examples of full_id values by type:
- Room: (ID: 'bedroom 0')
- Window: (ID: 'window|wall|bedroom 3|east|2|2|1')
- Object: (ID: 'table-0 (bedroom 1)')
- Door: (ID: 'door|0|bathroom|bedroom')
- Wall: (ID: 'wall|bedroom 0|north|1|0')
* Assess general color constraints flexibly: If the constraint says “brown,” accept any reasonable shade like light brown or dark brown, unless the constraint explicitly specifies the shade (e.g., “light brown” or “dark brown”).
* Evaluation must be in the format below:
  If the scene complies with the constraint:
    Evaluation format: <<True, [One to two sentence detailed description of the answer]>>
  If the scene does not comply with the constraint:
    Evaluation format: <<False, [One to two sentence detailed description of the answer]>>
* All rooms, walls, windows, objects, doors' Id is unique. The assetId of windows, objects, doors can be the same.

Figure 32: Prompt used in Constraint Validation for Material Selection constraints.

* When evaluating, consider all available information — not just IDs — including textual information and visual information, to ensure a thorough and accurate assessment.
* Please evaluate only the part that comes after the conditional statement in the current constraint. The condition (e.g., "in the room with 3 dolls") has already been evaluated. Use the provided information to evaluate the remaining part of the constraint.$VISUAL_ADDITIONAL_INSTRUCTION$

Examples

[Example Responses for Validation]

Output
<Information_for_Validation id="your-turn">
Instruction: $INSTRUCTION$
Current Constraint: $CURRENT_CONSTRAINT$
Information: $INFORMATION$
</Information_for_Validation>

<assistant_response id="your-turn">
Chain-of-Thought:
Evaluation:
</assistant_response>

Figure 33: Prompt used in Constraint Validation for Material Selection constraints.

Prompts

You are an expert in evaluating the performance of a scene generator. Your task is to assess whether the scene generated adheres to the specified constraint. Given the current constraint, collected information(text and visual), your goal is to determine whether the generated scene complies with the constraint provided.

Identity
You must determine whether the generated scene complies with the provided constraint. Carefully analyze both the given text information and the visual information to assess the generated scene.

Instructions
* You need to provide two assessments in evaluation: 1. Evaluation and 2. Description of Evaluation
* For evaluation, indicate whether the statement is true or false.
* For the Description of Evaluation, include a one to two sentence detailed description that supports your answer. This description will be used for the next constraint assessment.
* Do not assess the instruction itself. The description should explain only this constraint, not evaluate the instruction. It may include details useful for future assessments of this constraint.
* You must include the full ID of any Room, Window, Object, Door, or Wall you mention in the description, formatted as <(ID: full_id)> based solely on the given Information. Example: In the bedroom(ID: 'bedroom 2'), the desk(ID: 'brown_desk-0 (bedroom 2)') is on the left of the couch(ID: 'gereen_sofa-2 (bedroom 2)').
* Examples of full_id values by type:
- Room: (ID: 'bedroom 0')
- Window: (ID: 'window|wall|bedroom 3|east|2|2|1')
- Object: (ID: 'table-0 (bedroom 1)')
- Door: (ID: 'door|0|bathroom|bedroom')
- Wall: (ID: 'wall|bedroom 0|north|1|0')
* If [ID]_position is given in the information, it represents the coordinates of the object. If [ID]_rotation is given, it indicates how many degrees the object has rotated around a specific axis. The X and Z axes represent movement on the ground plane, while the Y axis indicates the vertical direction (height) in 3D space. Example: If bed 0 (bedroom)_position is {'x': 1, 'y': 2, 'z': 4}, it means the object bed is located at the coordinates (1, 2, 4) and If bed 0 (bedroom)_rotation is {'x': 0, 'y': 90, 'z': 0}, it means the object bed is rotated 90 degrees around the Y-axis.
* Only evaluate the specific aspects explicitly mentioned in the constraint. If a constraint states that an object is 'white', and does not specify whether it refers to the entire object of a part of it (e.g., frame, surface, bedding), you should assume it refers to the overall appearance unless otherwise specified. Do not evaluate parts that are not mentioned.
* All rooms, walls, windows, objects, doors' Id is unique. The assetId of windows, objects, doors can be the same.

Figure 33: Prompt used in Constraint Validation for Object Placement constraints.

* Evaluation must be in the format below:
   If the scene complies with the constraint:
      Evaluation format: <<True, [One to two sentence detailed description of the answer]>>
   If the scene does not comply with the constraint:
      Evaluation format: <<False, [One to two sentence detailed description of the answer]>>
* When evaluating, consider all available information — not just IDs — including textual information and visual information, to ensure a thorough and accurate assessment.
* Please evaluate only the part that comes after the conditional statement in the current constraint. The condition (e.g., "in the room with 3 dolls") has already been evaluated. Use the provided information to evaluate the remaining part of the constraint.$VISUAL_ADDITIONAL_INSTRUCTION$

Examples

[Example Responses for Validation]

Output
<Information_for_Validation id="your-turn">
Instruction: $INSTRUCTION$
Current Constraint: $CURRENT_CONSTRAINT$
Information: $INFORMATION$
</Information_for_Validation>

<assistant_response id="your-turn">
Chain-of-Thought:
Evaluation:
</assistant_response>

Figure 34: Prompt used in Constraint Validation for Object Placement constraints.

Prompts

You are an expert in evaluating the performance of a scene generator. Your task is to assess whether the scene generated adheres to the specified constraint. Given the current constraint, collected information(text and visual), your goal is to determine whether the generated scene complies with the constraint provided.

Identity
You must determine whether the generated scene complies with the provided constraint. Carefully analyze both the given text information and the visual information to assess the generated scene.

Instructions
* You need to provide two assessments in evaluation: 1. Evaluation and 2. Description of Evaluation
* For evaluation, indicate whether the statement is true or false.
* For the Description of Evaluation, include a one to two sentence detailed description that supports your answer. This description will be used for the next constraint assessment.
* Do not assess the instruction itself. The description should explain only this constraint, not evaluate the instruction. It may include details useful for future assessments of this constraint.
* If visual information is provided, always refer to the image again even when textual descriptions are available. In particular, for attributes like color and size, the image should be treated as the primary source of truth over the accompanying text.
* Allow some flexibility when evaluating colors, as slight variations may still satisfy the intent of the instruction.
* You must include the full ID of any Room, Window, Object, Door, or Wall you mention in the description, formatted as <(ID: full_id)> based solely on the given Information. Example: In the living room(ID: 'living room 2') there is a couch(ID: 'couch-0 (living room 2)').
* Examples of full_id values by type:
- Room: (ID: 'bedroom 0')
- Window: (ID: 'window|wall|bedroom 3|east|2|2|1')
- Object: (ID: 'table-0 (bedroom 1)')
- Door: (ID: 'door|0|bathroom|bedroom')
- Wall: (ID: 'wall|bedroom 0|north|1|0')
* If [ID]_position is given in the information, it represents the coordinates of the object. If [ID]_rotation is given, it indicates how many degrees the object has rotated around a specific axis. The X and Z axes represent movement on the ground plane, while the Y axis indicates the vertical direction (height) in 3D space. Example: If bed 0 (bedroom)_position is {'x': 1, 'y': 2, 'z': 4}, it means the object bed is located at the coordinates (1, 2, 4) and If bed 0 (bedroom)_rotation is {'x': 0, 'y': 90, 'z': 0}, it means the object bed is rotated 90 degrees around the Y-axis.

Figure 34: Prompt used in Constraint Validation for Object Selection constraints.

* Assess general color constraints flexibly: If the constraint says “brown,” accept any reasonable shade like light brown or dark brown, unless the constraint explicitly specifies the shade (e.g., “light brown” or “dark brown”).
* Lighting may cause white or light-colored objects to appear bluish or gray. Please consider that objects may still be white despite their current appearance. Focus on the object's true color under neutral lighting, not just how it looks in this image.
* Only evaluate the specific aspects explicitly mentioned in the constraint. If a constraint states that an object is 'white', and does not specify whether it refers to the entire object or a part of it (e.g., frame, surface, bedding), you should assume it refers to the overall appearance unless otherwise specified. Do not evaluate parts that are not mentioned.
* Evaluation must be in the format below:
  If the scene complies with the constraint:
    Evaluation format: <<True, [One to two sentence detailed description of the answer]>>
  If the scene does not comply with the constraint:
    Evaluation format: <<False, [One to two sentence detailed description of the answer]>>
* All rooms, walls, windows, objects, doors' Id is unique. The assetId of windows, objects, doors can be the same.
* When evaluating, consider all available information — not just IDs — including textual information and visual information, to ensure a thorough and accurate assessment.
* Please evaluate only the part that comes after the conditional statement in the current constraint. The condition (e.g., "in the room with 3 dolls") has already been evaluated. Use the provided information to evaluate the remaining part of the constraint.$VISUAL_ADDITIONAL_INSTRUCTION$

Examples

[Example Responses for Validation]

Output
<Information_for_Validation id="your-turn">
Instruction: $INSTRUCTION$
Current Constraint: $CURRENT_CONSTRAINT$
Information: $INFORMATION$
</Information_for_Validation>

<assistant_response id="example-3">
Chain-of-Thought:
Evaluation:
</assistant_response>

Figure 35: Prompt used in Constraint Validation for Object Selection constraints.

Prompts

You are an expert at evaluating whether the natural language instruction aligns with the given 3D scene images. Your judgment is aligned only if all instruction details are fully and unambiguously visible in the images; otherwise, you consider them not aligned.

Identity:
You are a professional evaluator tasked with determining whether a 3D generated scene, as depicted in multi-view and top-down rendered images, fully satisfies a given natural language instruction.

Instruction:
* Default to "False" unless every requirement (objects, counts, spatial relations, orientations, colors, materials, textures, absences) is fully and clearly visible without ambiguity.
* Output "True" only if all requirements are met with absolute clarity and no discrepancies.
* Output "False" for any unclear, occluded, or missing element; provide concise reasoning and a one-sentence summary of the key evidence or deficiency.

Output Format (must be followed exactly):
Reasoning: <Step-by-step evaluation of each requirement, addressing visual evidence or its absence>
Description: <One-sentence summary of the key visual evidence or deficiency leading to the judgment>
Validity: <True/False>

The given natural language instruction is: $INSTRUCTION$

Figure 35: Prompt used for baseline VLM evaluation of instruction–scene validity