SceneEval: Evaluating Semantic Coherence in Text-Conditioned 3D Indoor Scene Synthesis

Supplementary Material

Appendices

In this appendix, we provide details about our SceneEval-500 dataset in App. A, the implementation details of our metrics in App. B, experimental results of running SceneEval with an open-source VLM in App. C, and limitations of our work in App. D. We also provide the details of our user study in App. E, the vision language model (VLM) prompts we used in App. F, and information about scientific artifacts involved and AI assistant usage in this work in Apps. G and H.

A. Dataset Details

We provide additional details about our SceneEval-500 dataset below, including the annotation schema (App. A.1) and the data collection process (App. A.2).

A.1. Annotation Schema

SceneEval-500 contains four annotation fields: object count, object attribute, object-object relationships, and object-architecture relationships. Tab. 2 shows the annotation schema for each field along with examples. We describe the schema below.

Quantifier and quantity are used to specify the *count* of an annotation entry. For example, they are used to specify the number of objects in the scene for the object count field and the number of objects that have a specific attribute for the object attribute field. The quantifier can be one of the following: `eq` (equal), `gt` (greater than), `lt` (less than), `ge` (greater than or equal), or `le` (less than or equal). The quantity is an non-negative integer.

Object category specifies the object category of interest in an annotation entry. It is an open-vocabulary string that specifies exactly one object category using a noun (e.g., bed) or a noun phrase (e.g., office chair). In object-object relationships, multiple object categories are used to specify what objects are involved in a relationship, with the first object category being the anchor object that the relationship is based on. Fig. 5 shows the most frequent object categories in SceneEval-500.

Attribute specifies the attribute of an object in an object attribute annotation entry. It is an open-vocabulary string that specifies exactly one attribute using an adjective. For example, it can be color (e.g., red), material (e.g., wooden), shape (e.g., round), size (e.g., large), style (e.g., modern), or more specific attributes (e.g., queen-size).

Architecture type specifies the type of architectural element in an object-architecture relationship annotation entry. It can be one of the following: `wall`, `floor`, `ceiling`, `window`, `door`, or `room`. It can also be a more specific room type (e.g., bedroom, kitchen) for scenes with multiple rooms.

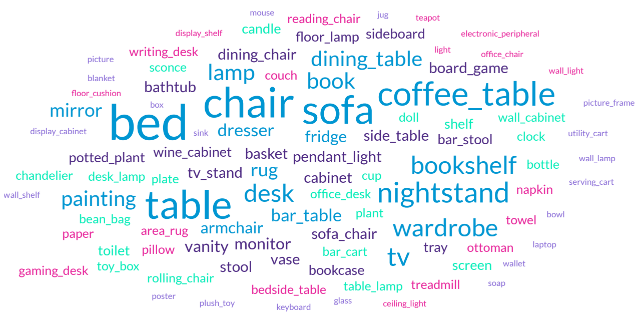


Figure 5. Word cloud showing the most frequent object categories in SceneEval-500.

Relationship specifies the relationship between objects in an object-object relationship annotation entry and the relationship between an object and an architectural element in an object-architecture relationship annotation entry. It is an open-vocabulary string that specifies exactly one relationship using a preposition (e.g., in front of, against), a verb (e.g., face, hang), or a prepositional phrase (e.g., at the foot of, at the corner of).

A.2. Manual Data Collection Process

Tab. 7 shows examples of scene descriptions and annotations in SceneEval-500. Below, we provide further details about manual curation of the initial 100 scene descriptions and annotations.

The first 100 scene descriptions and annotations in SceneEval-500 are written by the authors in English. The annotators are primarily from Asia, aged between 20 and 30, have lived in Western countries, and speak English as their second language. They have backgrounds in computer science, are familiar with the task of scene generation, and have been informed of the purpose of the dataset.

During the data collection process, the annotators were first given the definition of the difficulty levels, the annotation schema, and the target number of scenes to guide the annotation process. They were then asked to write scene descriptions that are diverse and cover a wide range of object categories and relationships, drawing inspiration from their daily lives or from online sources. At the same time, they were asked to annotate the scenes they wrote according to the annotation schema. Both the scene descriptions and

		Fidelity				Plausbility					
	Difficulty	$\uparrow$ CNT$_{\%}$	$\uparrow$ ATR$_{\%}$	$\uparrow$ OOR$_{\%}$	$\uparrow$ OAR$_{\%}$	$\downarrow$ COL$_{ob\%}$	$\downarrow$ COL$_{sc\%}$	$\uparrow$ SUP$_{\%}$	$\uparrow$ NAV$_{\%}$	$\uparrow$ ACC$_{\%}$	$\downarrow$ OOB$_{\%}$
ATISS	Easy	18.72	13.00	2.49	8.40	53.12	79.33	90.38	99.92	84.56	11.38
	Medium	12.93	8.32	1.46	7.81	50.53	73.50	91.50	99.80	85.81	10.14
	Hard	8.09	5.13	0.65	8.00	47.27	73.33	90.65	99.76	86.22	11.30
DiffuScene	Easy	20.64	15.20	4.48	†8.00	33.52	64.67	74.48	†99.51	81.32	†26.56
	Medium	14.26	11.03	4.26	†9.31	31.42	62.50	75.10	†99.41	82.72	†25.29
	Hard	8.25	6.45	2.42	†7.62	30.50	61.33	76.80	†99.41	81.32	†25.63
LayoutGPT	Easy	24.47	15.75	3.98	4.40	12.50	30.00	31.85	99.98	48.34	71.75
	Medium	14.18	9.37	1.46	5.11	11.07	25.50	29.45	100.00	48.18	72.38
	Hard	7.04	4.90	0.65	4.95	10.88	26.67	29.19	100.00	44.86	72.60
InstructScene	Easy	23.19	16.48	9.45	†10.40	56.11	89.33	77.58	†98.41	77.14	†24.31
	Medium	17.51	13.74	4.38	†12.01	52.64	84.00	80.55	†98.86	78.02	†19.03
	Hard	9.57	8.76	2.35	†8.95	56.92	84.67	84.69	†99.74	77.11	†14.50
LayoutVLM	Easy	39.36	20.88	9.45	24.40	33.17	66.00	79.23	99.37	82.49	4.52
	Medium	41.70	25.29	7.18	23.65	33.33	57.00	73.31	99.60	86.15	4.96
	Hard	30.58	17.17	4.90	14.29	29.45	50.67	78.62	99.99	87.88	5.23
Holodeck	Easy	44.47	39.19	22.89	38.00	16.20	70.67	60.83	99.61	89.75	1.57
	Medium	36.61	32.99	14.81	39.64	16.26	70.00	64.40	99.57	90.06	1.31
	Hard	26.95	22.64	8.10	35.43	15.23	76.67	63.96	99.64	89.07	1.49

Table 6. Breakdown of evaluation results by difficulty using SceneEval with the SceneEval-500 dataset on six recent scene generation methods. We report the values for each metric averaged across all scenes in each difficulty level. † indicates numbers related to the floor plan shape but the method cannot be conditioned on it. As the difficulty increases, the performance of the methods at generating scenes with the correct number of objects decreases. Compared to the other methods, Holodeck has the best overall performance across all difficulty levels.

annotations were validated by the authors to ensure quality and consistency. In addition, the scene descriptions were passed through an VLM to check for grammatical errors and typos. No personal identifiable information was collected during the whole data collection process.

B. Metric Implementation Details

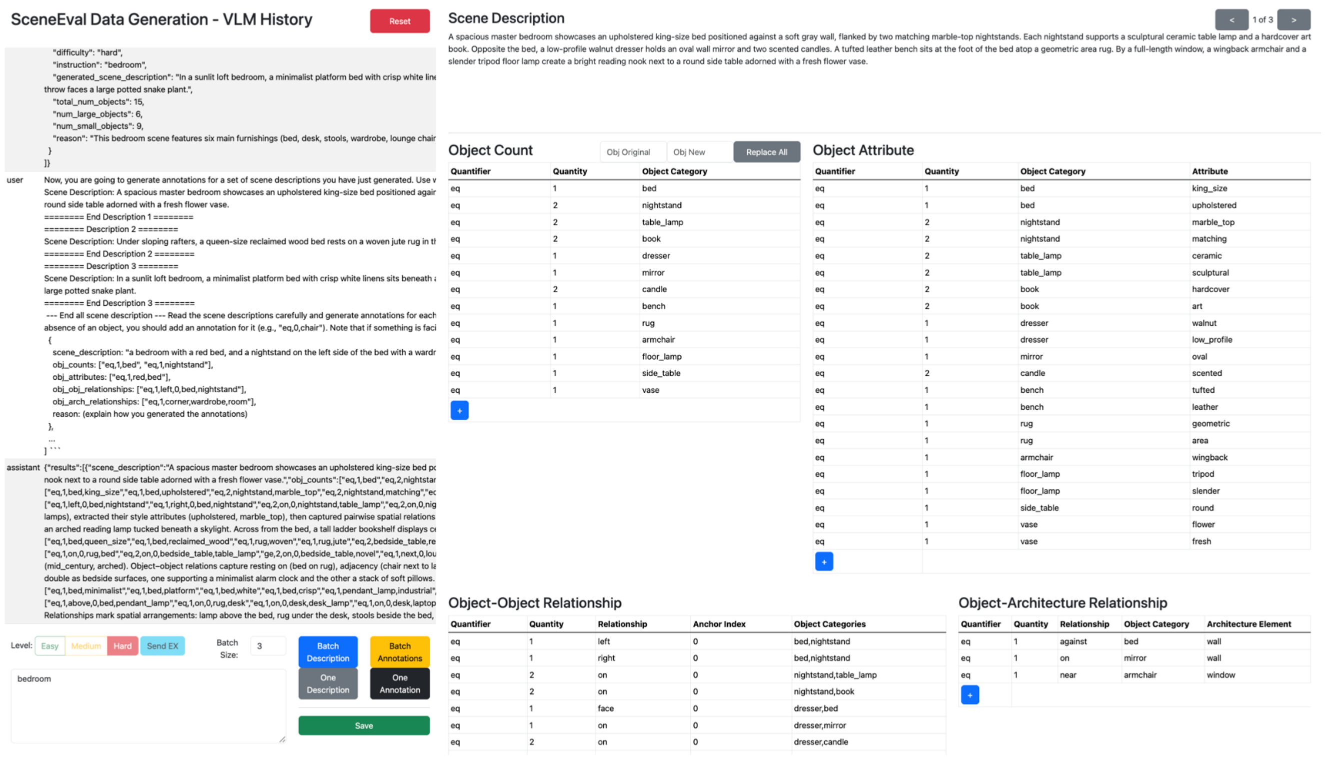
We provide details about about the object renderings used in SceneEval in App. B.1, additional implementation details for our metrics in App. B.2, and details about the predefined spatial relationships used in our object-object relationship and object-architecture relationship metrics in App. B.3.

B.1. Object Renderings

In SceneEval, we use three types of object renderings across our metrics: 1) **Front View**: The object is positioned in the center of the image, zoomed in, and rendered from the front with no other objects visible. 2) **Size Reference**: The object is rendered with a 170 cm tall human figure on the left side for size reference. 3) **Surrounding Context**: The object is positioned in the center of the image and zoomed out to show the surrounding context. These renderings help an LLM to understand the object's appearance, size, and context, respectively, for various evaluation tasks. Fig. 7 shows example renderings of these three rendering types.

Difficulty	Scene Description
Easy	A simple bedroom featuring a twin bed against the wall, with a wardrobe positioned in the corner of the room, and a desk next to the window.
Medium	This entertaining basement layout features a large gaming setup with two monitors on a desk against one wall, while a comfy bean bag chair is positioned nearby for casual seating. Across from the gaming area, a small cabinet with a mini fridge and a popcorn machine on top completes the setup.
Hard	This cozy bedroom features a full-size bed against a wall with two nightstands on each side where the left one has a small clock. A small desk sits in the corner with a comfortable chair for studying or working. Opposite the bed, a spacious dresser provides additional storage. Adjacent to the bedroom, the living room has a recliner and an ottoman facing a low coffee table with a small vase and magazines. A large bookshelf in the corner holds books and board games, while an wide couch provides space for gatherings. Just off the living room, a small gaming room with a gaming console and two beanbag chairs offers a dedicated space for entertainment, complete with a small shelf for controllers and headsets.

Table 7. Example scene descriptions for the three difficulty levels in SceneEval-500. The easy description specifies three large furniture objects. The medium one specifies three large and four small objects. The hard description specifies 14 large and 13 small objects.



```
    "difficulty": "hard",
    "instruction": "bedroom",
    "generated_scene_description": "In a sunlit loft bedroom, a minimalist platform bed with crisp white line
  throw faces a large potted snake plant.",
    "total_num_objects": 15,
    "num_large_objects": 6,
    "num_small_objects": 9,
    "reason": "This bedroom scene features six main furnishings (bed, desk, stools, wardrobe, lounge chair
   }
  ]}
user      Now, you are going to generate annotations for a set of scene descriptions you have just generated. Use v
          Scene Description: A spacious master bedroom showcases an upholstered king-size bed positioned agair
          round side table adorned with a fresh flower vase.
          ======== End Description 1 ========
          ======== Description 2 ========
          Scene Description: Under sloping rafters, a queen-size reclaimed wood bed rests on a woven jute rug in tl
          ======== End Description 2 ========
          ======== Description 3 ========
          Scene Description: In a sunlit loft bedroom, a minimalist platform bed with crisp white linens sits beneath a
          large potted snake plant.
          ======== End Description 3 ========
           --- End all scene description --- Read the scene descriptions carefully and generate annotations for each
          absence of an object, you should add an annotation for it (e.g., "eq,0,chair"). Note that if something is faci
           {
            scene_description: "a bedroom with a red bed, and a nightstand on the left side of the bed with a wardr
            obj_counts: ["eq,1,bed", "eq,1,nightstand"],
            obj_attributes: ["eq,1,red,bed"],
            obj_obj_relationships: ["eq,1,left,0,bed,nightstand"],
            obj_arch_relationships: ["eq,1,corner,wardrobe,room"],
            reason: (explain how you generated the annotations)
           },
           ...
          ] ```
assistant {"results":[{"scene_description":"A spacious master bedroom showcases an upholstered king-size bed pc
          nook next to a round side table adorned with a fresh flower vase.","obj_counts":["eq,1,bed","eq,2,nightstar
          ["eq,1,bed,king_size","eq,1,bed,upholstered","eq,2,nightstand,marble_top","eq,2,nightstand,matching","ec
          ["eq,1,left,0,bed,nightstand","eq,1,right,0,bed,nightstand","eq,2,on,0,nightstand,table_lamp","eq,2,on,0,nig
          lamps), extracted their style attributes (upholstered, marble_top), then captured pairwise spatial relations
          an arched reading lamp tucked beneath a skylight. Across from the bed, a tall ladder bookshelf displays ce
          ["eq,1,bed,queen_size","eq,1,bed,reclaimed_wood","eq,1,rug,woven","eq,1,rug,jute","eq,2,bedside_table,re
          ["eq,1,on,0,rug,bed","eq,2,on,0,bedside_table,table_lamp","ge,2,on,0,bedside_table,novel","eq,1,next,0,lou
          (mid_century, arched). Object–object relations capture resting on (bed on rug), adjacency (chair next to la
          double as bedside surfaces, one supporting a minimalist alarm clock and the other a stack of soft pillows.
          ["eq,1,bed,minimalist","eq,1,bed,platform","eq,1,bed,white","eq,1,bed,crisp","eq,1,pendant_lamp,industrial",
          ["eq,1,above,0,bed,pendant_lamp","eq,1,on,0,rug,desk","eq,1,on,0,desk,desk_lamp","eq,1,on,0,desk,laptop
          Relationships mark spatial arrangements: lamp above the bed, rug under the desk, stools beside the bed,
```

bedroom

A spacious master bedroom showcases an upholstered king-size bed positioned against a soft gray wall, flanked by two matching marble-top nightstands. Each nightstand supports a sculptural ceramic table lamp and a hardcover art book. Opposite the bed, a low-profile walnut dresser holds an oval wall mirror and two scented candles. A tufted leather bench sits at the foot of the bed atop a geometric area rug. By a full-length window, a wingback armchair and a slender tripod floor lamp create a bright reading nook next to a round side table adorned with a fresh flower vase.

Object Count

Quantifier	Quantity	Object Category
eq	1	bed
eq	2	nightstand
eq	2	table_lamp
eq	2	book
eq	1	dresser
eq	1	mirror
eq	2	candle
eq	1	bench
eq	1	rug
eq	1	armchair
eq	1	floor_lamp
eq	1	side_table
eq	1	vase

Object Attribute

Quantifier	Quantity	Object Category	Attribute
eq	1	bed	king_size
eq	1	bed	upholstered
eq	2	nightstand	marble_top
eq	2	nightstand	matching
eq	2	table_lamp	ceramic
eq	2	table_lamp	sculptural
eq	2	book	hardcover
eq	2	book	art
eq	1	dresser	walnut
eq	1	dresser	low_profile
eq	1	mirror	oval
eq	2	candle	scented
eq	1	bench	tufted
eq	1	bench	leather
eq	1	rug	geometric
eq	1	rug	area
eq	1	armchair	wingback
eq	1	floor_lamp	tripod
eq	1	floor_lamp	slender
eq	1	side_table	round
eq	1	vase	flower
eq	1	vase	fresh

Object-Object Relationship

Quantifier	Quantity	Relationship	Anchor Index	Object Categories
eq	1	left	0	bed,nightstand
eq	1	right	0	bed,nightstand
eq	2	on	0	nightstand,table_lamp
eq	2	on	0	nightstand,book
eq	1	face	0	dresser,bed
eq	1	on	0	dresser,mirror
eq	2	on	0	dresser,candle

Object-Architecture Relationship

Quantifier	Quantity	Relationship	Object Category	Architecture Element
eq	1	against	bed	wall
eq	1	on	mirror	wall
eq	1	near	armchair	window

Figure 6. Interface for semi-automatic data generation. **Top left:** Conversation history with the VLM. **Bottom left:** Controls for sending in-context examples and generating scene descriptions and annotations. **Top right:** Editable textbox containing the generated scene description. **Bottom right:** Editable tables containing the corresponding generated annotations. Each generated scene-description and annotation pair is manually validated before saving.

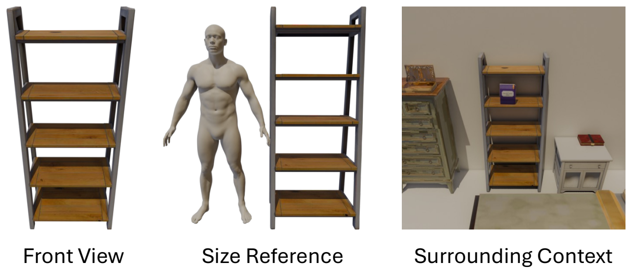


Figure 7. Example renderings of the three object rendering types used in SceneEval: Front View, Size Reference, and Surrounding Context.

B.2. Implementation Details

We provide additional details for object-object relationship and object support metrics below.

B.2.1. Object-Object Relationship

For each object-object relationship in the annotations, we first map the annotated relationship into one or more of the 13 predefined spatial relationships (see App. B.3) using an LLM. After mapping, we locate all objects in the scene that match the categories specified in the relationship. We consider all possible object combinations and compute a relationship score for each of them using the predefined spatial relationships. All mapped relationships must be satisfied for an object combination to satisfy the original specification.

B.2.2. Object Support

To evaluate whether an object is stably supported, we first give two rendered images (front view and surrounding context) to an LLM and ask it to determine the support type of the object (one of: `ground`, `object`, `wall`, or `ceiling`). Based on the type, we determine the object's support direction in its local frame (e.g., downward for `ground` and backward for `wall`) and cast rays towards that direction, from the object mesh vertices that are closest in that direction, and check for ray contacts with other geometries in the scene within 1 cm. `Wall` and `ceiling` objects are considered supported if there are any valid contact points. (e.g., a ceiling lamp hanging from one point on the ceiling). For `ground` and `object` types, we construct a convex hull from the contact points and project the object centroid in the gravity direction. The object is considered supported if the projection is within the hull. We repeat this process for all object instances and report the percentage of objects that are supported.

B.3. Predefined Spatial Relationships

B.3.1. Object-Object Relationships

Our object-object relationship metric uses a set of 13 predefined spatial relationships between objects. We describe the implementation details of these relationships below. Unless otherwise specified, we use a threshold of 0.5 to determine if a relationship is positive or negative.

Inside and Outside determine whether an object A is inside or outside another object B (e.g., a cup is inside a cabinet). We sample points within object A's bounding box and compute a score based on the percentage of points that are inside object B's bounding box.

Face determines whether an object A is facing another object B (e.g., a sofa is facing a TV). We sample points within object A's bounding box and shoot rays from these points in the direction of object A's front vector. If there are no intersections with object B, the relationship is negative. Otherwise, we take the mean coordinates of all intersection points and compute a score based on the angle between the front vector of object A and the vector from object A's centroid to the mean intersection point (ignoring the vertical axis). The score is 1.0 if the angle is 0.0, and drops to 0.0 as the angle approaches 30.0 degrees.

Side_of determines whether an object A is on one of the six sides (top, bottom, left, right, front, back) of another object B (e.g., a nightstand is on the left side of a bed). We sample points within object A's bounding box and compute a score based on the percentage of points that are on the specific side of object B's bounding box (in object B's local coordinate frame), excluding points that are inside object B's bounding box. Object B's bounding box is extended by 25% in each dimension to account for slight misalignment.

Side_region determines whether an object A is in one of the six side regions of another object B (e.g., a book is on the left side of a bookshelf). The difference between this relationship and `side_of` is that object A can be inside object B's bounding box. The implementation is the same as `side_of`, except that points inside object B's bounding box are not excluded and no extension is applied to object B's bounding box.

Long_short_side determines whether an object A is on the long or short side of another object B (e.g., a chair is on the long side of a table). The long and short sides are determined based on object B's bounding box dimensions. We sample points within object A's bounding box and compute a score based on the percentage of points that are on the long or short sides of object B.

On_top determines whether an object A is on top of another object B (e.g., a book is on top of a table). This relationship is specific for objects that are precisely placed on top of another object. The implementation is the same as `side_of`, with the top side of object B as the reference side, and no extension is applied to object B's bounding box.

Middle_of determines whether an object A is in the middle of another object B (e.g., a pillow is in the middle of a bed). We compute the distance between the centroids of object A and object B in 2D (ignoring the vertical axis) and apply a Gaussian with mean 0.0 and standard deviation 0.25 to compute the score.

Surround determines whether a group of objects N surrounds a central object B (e.g., two chairs and two armchairs surround a table). First, we calculate the ideal angle A for uniformly distributing the objects in N around object B as $A = \frac{2\pi}{|N|}$ and the mean distance D between the centroids of objects in N and object B. Next, we compute the distance deviation d_i and angle deviation a_i from the ideal distance and angle for each object i in N. Each deviation is normalized by D and A, respectively, and clipped to be within $[0, 1]$. Finally, the score s is computed as:

$$s = \frac{1}{2|N|} \sum_{i=1}^{|N|} (1 - d_i)^2 + (1 - a_i)^2 \tag{1}$$

Next_to, Near, Across, and Far determine whether an object A is within a certain distance from another object B (e.g., a TV is near a plant). `next_to` is defined as $0 \leq d \leq 0.5$, `near` is defined as $0.5 \leq d \leq 1.5$, `across` is defined as $1.5 \leq d \leq 4.0$, and `far` is defined as $d \geq 4.0$, where d is the distance in meters between the closest points of the two objects. The score is 1 if d falls within the specified range, and drops as d deviates from the range using a Gaussian with mean 0.0 and standard deviation 0.25.

B.3.2. Object-Architecture Relationships

Our object-architecture relationship metric uses a set of 10 predefined spatial relationships between objects and architecture. We describe the implementation details of these relationships below. Same as the object-object relationships, we use a threshold of 0.5 to determine if a relationship is positive or negative.

Next_to, Near, Across, and Far are defined the same as in the object-object relationships.

Inside_room determines whether an object A is inside a room (e.g., a chair is inside a living room). We sample points within object A's bounding box and cast rays towards the room's floor plane. The score is computed based on the percentage of points that intersect with the room's floor plane.

Middle_room determines whether an object A is in the middle of a room (e.g., a rug is in the middle of a room). We compute the distance between the centroid of object A and the room's centroid in 2D (ignoring the vertical axis) and apply a Gaussian with mean 0.0 and standard deviation $\frac{o}{2} + (1 - \frac{o}{r})$, where o is the longer side of the object's 2D dimensions, and r is the mean 2D room dimensions to compute the score, taking into account the object's size and the room's size.

Corner_room determines whether an object A is in a corner of a room (e.g., a plant is in a corner of a room). For every pair of walls in the room, we compute the distance scores between object A and the two walls similar to the `next_to` relationship but with a range of $0 \leq d \leq 0.8$ in meters and a Gaussian with mean 0.0 and standard deviation 0.25. We also compute the dot product between the front vectors of the two walls to determine if they are perpendicular. If they are perpendicular, the score for this pair of walls is the product of the two distance scores. The final score is the maximum score among all pairs of walls.

On_wall determines whether an object A is on a wall (e.g., a painting is on a wall). We first compute a score s_f based on the percentage of points sampled within object A's bounding box that lie in front of the wall. Next, we compute a score s_d based on the closest distance between object A and the wall similar to the `next_to` relationship but with a range of $0 \leq d \leq 0.01$ in meters and a Gaussian function of mean 0.0 and standard deviation 0.01. The final score is the product of s_f and s_d.

Against_wall determines whether an object A is against a wall (e.g., a sofa is against a wall). The implementation is the same as `on_wall`, except that the range for d is $0 \leq d \leq 0.3$ and the Gaussian function has a standard deviation of 0.1.

Hang_ceiling determines whether an object A is hanging from the ceiling (e.g., a light is hanging from the ceiling). The implementation is similar to `next_to`, except that the reference element is the ceiling, the range for d is $0 \leq d \leq 0.01$, and the Gaussian function has a standard deviation of 0.03.

C. SceneEval with Open-Source VLM

To assess the generality of our framework, we re-ran SceneEval on 500 scenes generated from the 100 manually written descriptions using Qwen2.5-VL-7B-Instruct [4], a publicly available open-source VLM that we were able to run locally with available resources.

The overall evaluation trends remain consistent with those obtained using our original model (GPT-4o [1]). Agreement with our manual evaluation across the four fidelity metrics is 80.65% (Object Count), 76.64% (Attribute), 91.11% (Object-Object Relationship), and 87.72% (Object-Architecture Relationship), with Cohen's kappas of 0.50, 0.26, 0.43, and 0.47, respectively. For the user study, we report agreement using only the subset of user-rated scenes that overlap with the 100 manual descriptions. On this subset, agreement is 77.55%, 75.00%, 78.67%, and 83.48%, with Cohen's kappas of 0.48, 0.12, 0.32, and 0.48, respectively.

These results show that while agreement scores using Qwen2.5-VL-7B-Instruct are lower than those achieved with GPT-4o (a significantly larger and more capable model) — particularly for the attribute metric, which solely relies on the VLM for evaluation — they still follow the same general trends and remain within a reasonable range. This highlights the benefit of using stronger VLMs for specific perception components (such as attribute recognition), while also demonstrating that SceneEval's evaluation pipeline is modular and not overly dependent on any single model.

D. Limitations

While our dataset and metrics provide a better coverage of important aspects of scene generation compared to existing metrics, they are not perfect. We provide a discussion of the limitations of SceneEval, SceneEval-500, and our semi-automatic data generation process below.

D.1. Limitations in SceneEval

First, our metrics currently do not consider whether objects in the generated scenes are placed according to "common sense" expectations, even if they are not explicitly specified in the input text descriptions. For example, large furniture items, like bookshelves, are typically placed against walls, except when they are used to divide spaces. Such common sense expectations are crucial for realism of the generated scenes and is an important aspect for evaluating scene generation models. Unfortunately, such expectations are less well-defined. As a result, incorporating them into the evaluation metrics is challenging and requires further research.

Second, SceneEval's execution time currently scales with the number of objects in the scene. As scenes get more complex, the time required to perform object matching and compute the metrics also increases, as there are more objects to process. Exploring parallelization and other optimization techniques to reduce the execution time is an important direction for future work.

D.2. Limitations in SceneEval-500

While our dataset includes a broader range of room types than prior work, its scale remains limited compared to datasets in other domains (e.g., image), which often contain thousands to millions of entries. Expanding the dataset further, especially through scalable automatic methods, would allow for more comprehensive evaluations and a deeper analysis of model capabilities.

Additionally, the authors who created the initial 100 descriptions and annotations, and who validated and corrected the semi-automatically generated ones, are primarily from Asia and have lived in Western countries. None are trained interior designers or architects. As a result, the dataset may reflect cultural assumptions and expectations that are not universally representative.

D.3. Semi-Automatic Data Generation Limitations

Despite the advantages of semi-automatic data generation over manually writing descriptions and annotations, it is not without its challenges. During data generation, we encountered four main failure modes that limit scalability and prevent fully automatic generation:

Missing annotations. The most common issue is the omission of object attributes and object-object relationships. Specifically, 101 missing attributes (e.g., "wooden") were identified and manually added during the generation of 210 medium and hard scene descriptions. Additionally, we manually added 154 missing object-object relationships. Many cases involved the omission of one among multiple constraints for an object; for instance, given the description "On the desk, a pen is right of a laptop," the annotation "pen on desk" was missing. Such omissions are especially common in hard scenes, whose descriptions are longer and contain more attributes and relations.

Hallucination. During consecutive generation with a lengthy conversation history, the VLM often erroneously includes annotations from previously generated descriptions. This issue becomes more frequent after generating more than five descriptions and their annotations. It also affects the anchor index field, where the VLM may output incorrect values (e.g., specifying an index of 14 when only two objects are involved in the relationship).

Inconsistency in object attributes. The VLM occasionally merges multiple attributes into a single non-atomic entry. For example, it may produce "round wooden" as one attribute instead of "round" and "wooden". This violates our expectation that each attribute is independent, and requires manual splitting during validation.

Decreasing diversity. In addition to increased hallucination, we observed that with longer generation history, the VLM tends to produce scene descriptions with reduced diversity in object selection or spatial arrangement, requiring manual intervention such as resetting the conversation history or prompting with specific scene ideas.

These limitations currently prevent fully automatic dataset generation. Improving the generation process remains an important direction for future work toward building larger and more diverse datasets with reduced manual effort.

E. User Study Details

Fig. 8 shows the interface used in our user study. Participants are presented with a 3D scene in an interactive viewer, enabling them to freely inspect specific details while evaluating the listed scene properties. Participants are instructed to carefully examine both the scene and the expected properties, selecting *True* or *False* to indicate whether each property is satisfied. Once complete, they save their responses to a dedicated cloud storage location.

F. VLM Prompts

SceneEval uses an VLM to assist in parts of the evaluation framework. We provide the system prompt in App. F.1 and the evaluation task prompts in App. F.2. Additionally, we provide the prompts for semi-automatic data generation in App. F.3.

F.1. System Prompt

The system prompt provides the VLM with the overall context about the tasks and the role it plays in the evaluation framework.

```
system: >
  You are an expert in interior design.
  You have seen thousands of interior designs and have a
  ↪  good understanding of the spatial arrangement of
  ↪  objects in a room.
  Now, you are working as an evaluator for a design
  ↪  company.
  Use your expertise in interior design to evaluate the
  ↪  spatial arrangement of objects in the given scene
  ↪  according to the task instructions.
  When you are required to include object descriptions
  ↪  in your response, respond exactly as they are
  ↪  provided in the task instructions word for word.
  When you are required to give a specific side for a
  ↪  response to a relationship, use only the sides
  ↪  provided in the task instructions.
```

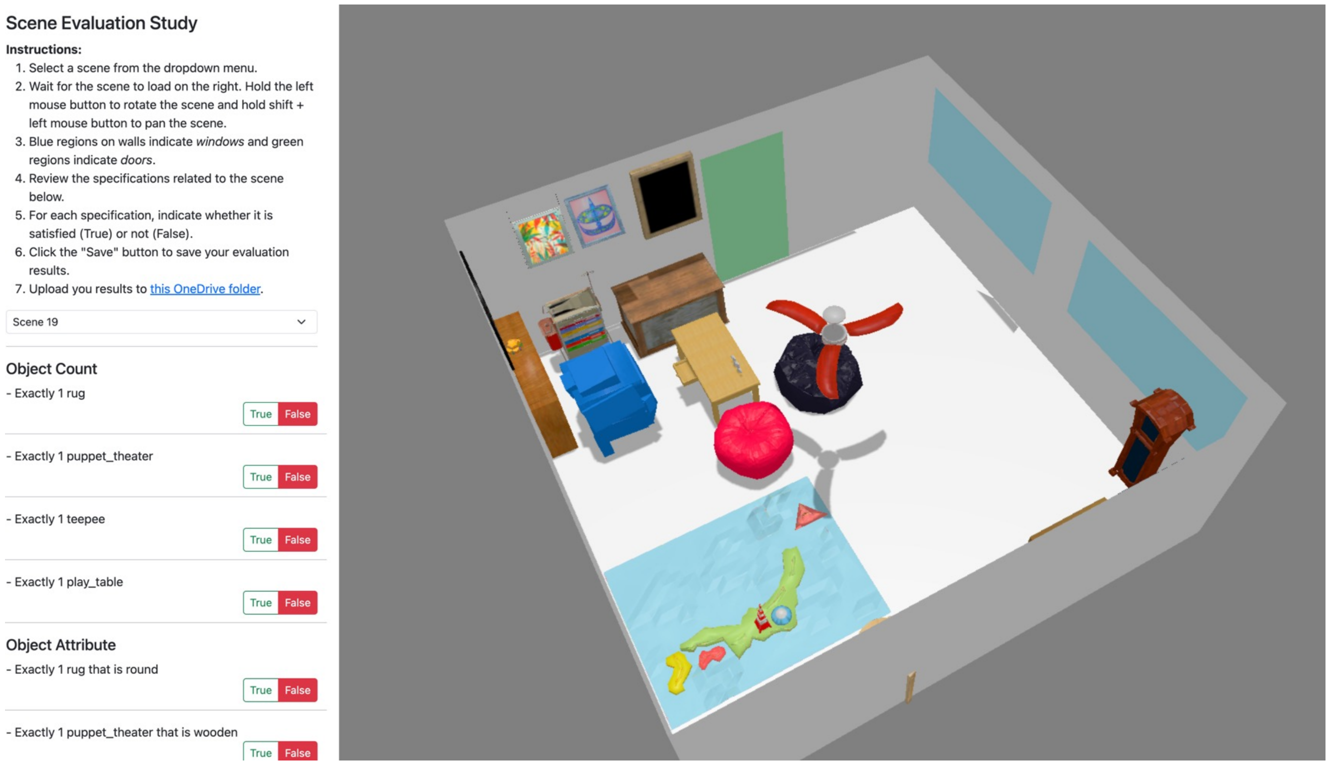
F.2. Evaluation Task Prompts

There are six tasks in SceneEval that use an VLM for assistance. We provide the prompts for each task below.

F.2.1. Object Matching

This task asks the VLM to match objects in the scene to the object categories specified in the ground truth annotation. Given a front-view image of an object and the object categories, the VLM is asked to determine if the object belongs to any of the specified categories and provide a justification.

~~~
obj_matching: >
  The user specified the scene to contain objects of
  ↪  certain categories.
  To facilitate further evaluation, you need to match
  ↪  the objects in the scene to the object categories
  ↪  specified by the user.
  You are provided an image of one of the objects in the
  ↪  scene.
  Does the object in the image belong to any of the
  ↪  object categories specified by the user?
  Respond in the given response schema. Here are two
  ↪  example responses:
  ```
  provided_categories: ["chair", "table", "lamp"]
  matched: True
  matched_category: "chair"
  reason: "The object in the image is a chair."
  ```
  ```
  provided_categories: ["chair", "table", "lamp"]
  matched: False
  matched_category: ""
  reason: "The object in the image is a sofa, which does
  ↪  not match any of the specified categories."
  ```
~~~


Scene Evaluation Study

Instructions:

1. Select a scene from the dropdown menu.
2. Wait for the scene to load on the right. Hold the left mouse button to rotate the scene and hold shift + left mouse button to pan the scene.
3. Blue regions on walls indicate *windows* and green regions indicate *doors*.
4. Review the specifications related to the scene below.
5. For each specification, indicate whether it is satisfied (True) or not (False).
6. Click the "Save" button to save your evaluation results.
7. Upload you results to this OneDrive folder.

Scene 19

Object Count

- Exactly 1 rug

True False

- Exactly 1 puppet_theater

True False

- Exactly 1 teepee

True False

- Exactly 1 play_table

True False

Object Attribute

- Exactly 1 rug that is round

True False

- Exactly 1 puppet_theater that is wooden

True False

Figure 8. Interface used for the user study. **Top left:** Study instructions and interface guidelines. **Bottom left:** Scene properties to evaluate. **Right:** Interactive 3D viewer with free camera control for detailed inspection.

```
  If the object in the image does not belong to any of
  ↪  the object categories specified by the user,
  ↪  respond with "matched: False" and
  ↪  "matched_category: "".
  Here is the list of object categories that the user
  ↪  specified to match against:
  "<TARGET_CATEGORIES>"
```

F.2.2. Object Attribute

This task asks the VLM to determine if the objects in the scene satisfy the attribute requirements in the annotations. For each object of interest, the VLM is provided with two images: one from the front view and one with a human model on the side for scale. The VLM is asked to determine if the object satisfies the attribute requirements and provide a reason for its decision.

```
obj_attribute: >
  The user specified the scene to contain objects with
  ↪  certain attributes.
  You are provided with images of instances of objects
  ↪  in the scene with the same category.
  There are two images for each object instance: one
  ↪  from the front view and one with a 170cm human
  ↪  model for scale.
  The images are in the following order: obj1_front,
  ↪  obj1_scale, obj2_front, obj2_scale, ...
  Given these images, how many of these objects satisfy
  ↪  the attribute requirements specified by the user?
  Note that the human model is included in the images
  ↪  solely for scale reference and should not be
  ↪  considered as part of the evaluation.
  Respond in the given response schema. Here is an
  ↪  example response:
  ```
  category: "chair",
  num_instances: 3,
  [
    {
      "instance": 0,
      "attribute": "red",
      "satisfied": True,
      "reason": "This chair is red."
    },
    {
      "instance": 1,
      "attribute": "red",
      "satisfied": False,
      "reason": "This chair is blue."
    },
    ...
  ]
  ```
  The attribute requirements are as follows:
  "<OBJ_ATTRIBUTES>"
  Here are the renderings of "<OBJ_COUNT>" instances of
  ↪  object with category "<OBJ_CATEGORY>" in the
  ↪  scene.
```

F.2.3. Object Support Type

This task asks the VLM to identify the support type of objects in the scene. For each object, the VLM is provided with two images: one from the front view and one slightly zoomed out to show the surrounding area. The VLM is asked to pick the support type of the object from the predefined types and provide a reason for its decision.

```
obj_support_type: >
  Objects in the scene are placed on the ground, on
  ↪  wall, on ceiling, or on other objects.
```

```
  You are given two images of an object in the scene:
  ↪  one from the front view and one slightly zoomed
  ↪  out to show the surrounding area.
  Using the images, identify the support type of the
  ↪  object.
  The support type of an object is the surface on which
  ↪  the object is placed.
  Here are the support types for objects:
  - ground: The object is placed on the ground. (e.g.,
  ↪  "table on the ground")
  - wall: The object is placed on the wall. (e.g.,
  ↪  "painting on the wall")
  - ceiling: The object is placed on the ceiling. (e.g.,
  ↪  "lamp hanging from the ceiling")
  - object: The object is placed on a surface of another
  ↪  object. (e.g., "book on the table")
  Respond in the given response schema. Here is two
  ↪  example responses:
  ```
  support_type: "ground",
  reason: "The table is placed on the ground."
  ```
  ```
  support_type: "wall",
  reason: "The painting is placed on the wall."
  ```
  If the object appears to be a ceiling light, carefully
  ↪  consider the image as it may be difficult to see
  ↪  that the object is hanging from the ceiling.
```

F.2.4. Object Functional Sides

This task asks the VLM to identify the functional sides of objects in the scene. The functional sides of an object are the sides that need to be accessible for the object to be used properly. The VLM is provided with descriptions of the objects in the scene and asked to identify the functional sides of each object with a justification.

```
obj_functional_sides: >
  Objects in the scene have functional sides that are
  ↪  important for their placement and use.
  The functional sides of an object are the sides that
  ↪  need to be accessible for the object to be used
  ↪  properly.
  Here, only consider these four sides of an object:
  ↪  ["front", "back", "left", "right"].
  If an object is placed in a scene, at least one of its
  ↪  functional sides should be accessible for the
  ↪  object to be considered properly placed.
  If an object has multiple functional sides, this means
  ↪  that the object can be used from any of these
  ↪  sides and there is no difference in importance
  ↪  between them.
  Otherwise, only consider the most important functional
  ↪  side as the sole functional side of the object.
  Here are some examples of different cases:
  - Objects that have equal importance for their
  ↪  functional sides:
    - bed: ["front", "left", "right"]
    - dining_table: ["front", "back", "left", "right"]
  - Objects that have a significant front side:
    - desk: ["front"]
    - sofa: ["front"]
  - Objects that can be moved so all sides are
  ↪  functional:
    - dining_chair: ["front", "back", "left", "right"]
    - stool: ["front", "back", "left", "right"]
  You are provided with descriptions of the objects in
  ↪  the scene.
  The task is to identify the functional sides of each
  ↪  of the objects.
  Note that for small objects like cups and books that
  ↪  are placed on a surface, do not consider their
  ↪  functional sides and respond with an empty list.
  Respond in the given response schema. Here is an
  ↪  example response:
  ```
  [
    {
      "obj_description": "bed.n.01 - bed description",
      "functonal_sides": ["front", "left", "right"],
      "reason": "These three sides of a bed have equal
      ↪ importance for accessibility and as long as
      ↪ one of them is accessible, the bed is
      ↪ considered properly placed."
    },
    {
      "obj_description": "chair.n.01 - chair
      ↪ description",
      "functional_sides": ["front", "back", "left",
      ↪ "right"],
      "reason": "All four sides of a chair are
      ↪ functional because it can be moved and used
      ↪ from any side."
    },
    {
      "obj_description": "cup.n.01 - cup description",
      "functional_sides": [],
      "reason": "Cups are small objects that do not have
      ↪ functional sides."
    }
    ...
  ]
  ```
  The descriptions of the objects in the scene are as
  ↪  follows:
  "<OBJ_DESCRIPTIONS>"
```

F.2.5. Object Relationship Mapping

This task asks the VLM to map open-vocabulary object-object relationships in the annotations to predefined spatial relationship types. For each input relationship, the VLM can choose multiple relationship types if multiple types are required to fully describe the relationship. The prompt provides the definition of the predefined spatial relationship types, with examples and guidelines for mapping the relationships. The VLM is asked to provide the mapped relationship types for each input relationship along with the necessary information.

```
obj_relationship_mapping: >
  The user specified the scene to contain certain
  ↪  relationships between objects.
  An object-object relationship is a spatial
  ↪  relationship between two or more objects in the
  ↪  scene.
  In which, an anchor object is the object that is used
  ↪  as a reference point to compare against.
  Here are some examples to illustrate the concept of
  ↪  anchor object:
  - "chair next to the table": the table is the anchor
  ↪  object.
  - "lamp near the sofa": the sofa is the anchor object.
  You are provided with manually annotated relationships
  ↪  between objects in the scene.
  The task is to map the mentioned relationships into
  ↪  one or more of the predefined spatial relationship
  ↪  type here:
  - inside_of: The target object is inside the anchor
  ↪  object. (e.g., "cup inside the cabinet")
  - outside_of: The target object is outside the anchor
  ↪  object. (e.g., "toy outside the box")
  - face_to: The target object is facing the anchor
  ↪  object. (e.g., "sofa facing the TV")
  - side_of: The target object is at one of the six
  ↪  sides (left, right, front, back, top, bottom) of
  ↪  the anchor object. (e.g., "nightstand left of the
  ↪  bed")
```

```
- side_region: The target object is inside the anchor
↪  object at one of the six sides (left, right,
↪  front, back, top, bottom). (e.g., "book on the
↪  left side of the shelf", "phone on the left side
↪  of the table")
- long_short_side_of: The target object is
↪  specifically at a long or short side of the anchor
↪  object. (e.g., "book at the long side of the
↪  table")
- on_top: The target object is on top of the anchor
↪  object at its top-most surface and not inside it.
↪  (e.g., "book on top of the table", but not
↪  applicable for "book on a "bookshelf" because the
↪  book is technically inside the bookshelf - use
↪  inside_of instead)
- middle_of: The target object is in the middle of the
↪  anchor object. (This only compares the objects in
↪  2D, e.g., "pillow in the middle of the bed")
- surround: Multiple target objects (can be different
↪  types) are circled around one anchor object.
↪  (e.g., "four chairs surrounding the table")
- next_to: The target object is next to the anchor
↪  object within 0 to 0.5m (e.g., "chair next to the
↪  table")
- near: The target object is near the anchor object
↪  within 0.5 to 1.5m. (e.g., "sofa near the TV")
- across_from: The target object is far from the
↪  anchor object within 1.5 to 4m. (e.g., "lamp
↪  across the room from the sofa")
- far: The target object is far from the anchor object
↪  beyond 4m. (e.g., "painting far from the bed")
- None: None of the predefined spatial relationships
↪  above match the relationship
You can choose multiple relationship types for a
↪  single input relationship if it requires multiple
↪  types to fully describe the relationship.
Here is an example of relationships that require
↪  multiple types to fully describe them:
- "table at the foot of the bed" needs both "side_of"
↪  and "next_to" relationship types to fully describe
↪  it.
Here are some additional guidelines for mapping the
↪  relationships:
- When choosing side_of and side_region, you must also
↪  specify the side of the anchor object (left,
↪  right, front, back, top, bottom).
- When choosing long_short_side_of, you must also
↪  specify the side of the anchor object (long,
↪  short).
- For side ambiguous relationships, like "next to" or
↪  "adjacent to", simply choose the distance-based
↪  relationship (next_to, near, across_from, far).
- When you choose multiple types for a single
↪  relationship, and some of the types require
↪  specifying a side, you must specify the side for
↪  all types in the same order as the types are
↪  listed.
  - Use "None" for the side when a type does not
  ↪  require specifying a side.
  - For example, if you choose both "side_of" and
  ↪  "next_to" for a relationship, you must specify
  ↪  the sides as ["front", None].
- Even if the relationship type does not require
↪  specifying a side, you must still provide a side
↪  as "None" in the response at the corresponding
↪  index.
- When the anchor object is not specified (i.e., when
↪  the anchor index is -1), put the first object in
↪  the relationship as the anchor object in your
↪  response.
- The other_object_counts are the number of objects
↪  that are part of the relationship for each object
↪  category in other_objects in the same order.
- When none of the predefined spatial relationships
↪  match the relationship, put "None" as the
↪  relationship type and provide a reason. (Do not
↪  put an empty list.)
Respond in the given response schema. Here is an
↪  example response:
```
```
[
  {
    "relationship": "beneath - objects: box, bed, with
    ↪  the object with index: 0 being the anchor",
    "anchor_object": "bed",
    "other_objects": ["box"],
    "other_object_counts": [1],
    "relationship_types": ["side_region"],
    "sides": ["bottom"],
    "reason": "Box beneath the bed is considered as
    ↪  the box being inside the bed at the bottom
    ↪  side."
  },
  {
    "relationship": "next_to - objects: lamp, chair,
    ↪  with the object with index: 0 being the
    ↪  anchor",
    "anchor_object": "lamp",
    "other_objects": ["chair"],
    "other_object_counts": [1],
    "relationship_types": ["next_to"],
    "sides": [None],
    "reason": "Chair next to the lamp is considered as
    ↪  the chair being next to the lamp."
  },
  {
    "relationship": "at the foot of - objects: bed,
    ↪  table, with the object with index: 0 being the
    ↪  anchor",
    "anchor_object": "bed",
    "other_objects": ["table"],
    "other_object_counts": [1],
    "relationship_types": ["side_of", "next_to"],
    "sides": ["front", None],
    "reason": "Table at the foot of the bed is
    ↪  considered as the table being at the front
    ↪  side of the bed and next to it."
  },
  {
    "relationship": "surround - objects: table,
    ↪  chair:0, chair:1, chair:2, sofa, with the
    ↪  object with index: 0 being the anchor",
    "anchor_object": "table",
    "other_objects": ["chair", "sofa"],
    "other_object_counts": [3, 1],
    "relationship_type": ["surround"],
    "side": [None],
    "reason": "Three chairs and a sofa surrounding the
    ↪  table is considered as the chairs and the sofa
    ↪  surrounding the table."
  },
  {
    "relationship": "diagnoally across - objects:
    ↪  table, chair, with the object with index: 0
    ↪  being the anchor",
    "anchor_object": "table",
    "other_objects": ["chair"],
    "other_object_counts": [1],
    "relationship_types": None
    "sides": [None],
    "reason": "No appropriate relationship type found
    ↪  for this relationship - chair diagonally
    ↪  across the table."
  }
  ...
]
```
The annotated relationships between objects in the
↪ scene are as follows:
"<RELATIONSHIPS>"
```

### F.2.6. Architectural Relationship Mapping

This task asks the VLM to map open-vocabulary relationships between objects and architectural elements in the annotations to predefined spatial relationship types. Similar to the object relationship mapping task, the VLM is pro-
```

vided with the definitions of the predefined spatial relationship types, with examples and guidelines for mapping the relationships. The VLM is asked to provide the mapped relationship types for each input relationship along with the necessary information.

```
arch_relationship_mapping: >
  The user specified the scene to contain certain
  ↪  relationships between objects and architectural
  ↪  elements.
  An architectural element is a structural component of
  ↪  a building, such as a wall, floor, ceiling, or
  ↪  room.
  Here are some examples of relationships between
  ↪  objects and architectural elements:
  - "painting on the wall"
  - "bookshelf against the wall"
  You are provided with manually annotated relationships
  ↪  between objects and architectural elements in the
  ↪  scene.
  The task is to map the mentioned relationships into
  ↪  one of the predefined spatial relationship type:
  - inside_room: The target object is inside the room.
  ↪  (e.g., "sofa inside the room")
  - middle_of_room: The target object is in the middle
  ↪  of the room. (e.g., "table in the middle of the
  ↪  room")
  - next_to: The target object is next to an
  ↪  architectural element within 0 to 0.5m. (e.g.,
  ↪  "chair next to the wall")
  - near: The target object is near an architectural
  ↪  element within 0.5 to 1.5m. (e.g., "lamp near the
  ↪  door")
  - across_from: The target object is far from an
  ↪  architectural element within 1.5 to 4m. (e.g.,
  ↪  "art across from the wall")
  - far: The target object is far from an architectural
  ↪  element beyond 4m. (e.g., "table far from the
  ↪  window")
  - on_wall: The target object is on the wall (must be
  ↪  directly in front of the wall). (e.g., "painting
  ↪  on the wall")
  - against_wall: The target object is against the wall
  ↪  (must be directly in front of the wall). (e.g.,
  ↪  "bookshelf against the wall")
  - corner_of_room: The target object is at the corner
  ↪  of the room. (e.g., "chair at the corner of the
  ↪  room")
  - hang_from_ceiling: The target object is hanging from
  ↪  the ceiling. (e.g., "lamp hanging from the
  ↪  ceiling")
  - None: None of the predefined spatial relationships
  ↪  above match the relationship
  Here are some additional guidelines for mapping the
  ↪  relationships:
  - When specifying the architectural element type,
  ↪  select from the following: ["wall", "floor",
  ↪  "ceiling", "room", "window", "door"].
  - When choosing floor or room, you must also specify
  ↪  the specific floors from the provided list of
  ↪  floor IDs.
    - If the IDs are not informative enough and you
    ↪  cannot determine the specific floor, choose all
    ↪  floors in the scene.
    - If the relationship is not specific to a floor,
    ↪  choose all floors in the scene.
  - When none of the predefined spatial relationships
  ↪  match the relationship, put "None" as the
  ↪  relationship type and provide a reason.
  Respond in the given response schema. Here is an
  ↪  example response:
  ```
  [
    {
      "relationship": "on - object: painting, with
      ↪ respect to architectural element: wall"
      "target_object": "painting",
      "architectural_element_type": "wall",
      "relationship_type": "on_wall",
      "specific_floors": [],
      "reason": "The painting is on the wall."
    },
    {
      "relationship": "along - object: bookshelf, with
      ↪ respect to architectural element: wall"
      "target_object": "bookshelf",
      "architectural_element_type": "wall",
      "relationship_type": "against_wall",
      "specific_floors": [],
      "reason": "The bookshelf is along a wall means
      ↪ that it is in front of and against the wall."
    },
    {
      "relationship": "corner - object: chair, with
      ↪ respect to architectural element: bedroom"
      "target_object": "chair",
      "arch_element_type": "room",
      "relationship_type": "corner_of_room",
      "specific_floors": ["floor_bedroom_001", ...]
      "reason": "The chair is at the corner of the
      ↪ room."
    }
    ...
  ]
  ```
  The annotated relationships between objects and
  ↪  architectural elements in the scene are as
  ↪  follows:
  "<RELATIONSHIPS>"
  Here are all the floors in the scene that you can
  ↪  choose from:
  "<FLOOR_IDS>"
```

F.3. Semi-Automatic Data Generation

The semi-automatic data generation process uses a VLM to generate scenes descriptions and annotations given a set of manually created entries as in-content examples.

F.3.1. System Prompt

The system prompt provides the VLM with the task description, the annotation schema, and other relevant guidelines.

```
system: >
  You are tasked to generate annotations for scenes
  ↪  given a text description of the scene.
  There are four types of annotations that you need to
  ↪  generate:
  - Object Count: The number of objects in the scene.
  ↪  The schema is:
    - {eq, lt, gt, le,
    ↪  ge},<instance_count>,<obj_reference>
    - e.g., "eq,3,chair" means that there are exactly 3
    ↪  chairs in the scene
    - <obj_reference> is the category name of the
    ↪  object, e.g., "chair", "table", "lamp", and not
    ↪  the object instance name
      - The object category must not include a
      ↪  descriptive attribute
      - e.g., "fridge" instead of "mini_fridge", as
      ↪  "mini" can be captured in the object
      ↪  attributes
      - On the other hand, "floor_lamp" is acceptable as
      ↪  it is a specific type of lamp and because
      ↪  "floor" is not an attribute
  - Object Attribute: The attributes of the objects in
  ↪  the scene. The schema is:
    - {eq, lt, gt, le,
    ↪  ge},<instance_count>,<obj_reference>,<attribute>
    - e.g., "eq,1,chair,red" means that there is exactly
    ↪  1 red chair in the scene
    - All <obj_reference> must refer to an object
    ↪  category that is mentioned in the object count
    ↪  for the same scene
```

```
- Object-Object Relationship: The relationships
↪  between objects in the scene. The schema is:
  - {eq, lt, gt, le,
  ↪  ge},<instance_count>,<relationship>,
  ↪  <anchor_index>,<obj_reference_0>,
  ↪  <obj_reference_1>,<obj_reference_n>
  - e.g., "eq,1,front,0,desk,chair" means that the
  ↪  object at index 0 (desk) is the anchor object
  ↪  and a chair is in front of it, and there is
  ↪  exactly 1 such relationship in the scene
  - <obj_refernce> is category name of the object,
  ↪  e.g., "chair", "table", "lamp", and not the
  ↪  object instance name
  - All <obj_reference> must refer to an object
  ↪  category that is mentioned in the object count
  ↪  for the same scene
  - Note that the <relatioship> must be broken into
  ↪  pairwise relationships, except for "surround" as
  ↪  in "three chairs surrounding the table"
    - e.g., "two chairs next to the table" should be
    ↪  broken into as "eq,2,next,0,table,chair"
    - e.g., "two nightstands on two sides of the bed"
    ↪  should be broken into two relationships:
    ↪  "eq,1,left,0,bed,nightstand" and
    ↪  "eq,1,right,0,bed,nightstand"
  - The <anchor_index> is the index of the anchor
  ↪  object in the scene description where the
  ↪  corresponding object is the reference point for
  ↪  the relationship
    - The index starts from 0 and is based on the
    ↪  order of the objects in the scene description
    - The anchor object can be any object in the scene
    - e.g., in the relationship
    ↪  "eq,1,left,0,bed,nightstand", the anchor
    ↪  object is the bed (index 0) and the nightstand
    ↪  is to the left of it
    - e.g., in the relationship
    ↪  "eq,1,face,0,desk,chair", the anchor object is
    ↪  the desk (index 0) and the chair is facing it
    - Fix the anchor index to be 0 and instead
    ↪  re-arrange the obj_references as needed
- Object-Architecture Relationship: The relationships
↪  between objects and architectural elements in the
↪  scene. The schema is:
  - {eq, lt, gt, le,
  ↪  ge},<instance_count>,<relationship>,
  ↪  <obj_reference>,<arch_reference>
  - e.g., "eq,1,against,bookshelf,wall" means that the
  ↪  bookshelf is against the wall, and there is
  ↪  exactly 1 such relationship in the scene
  - <obj_reference> is the category name of the
  ↪  object, e.g., "chair", "table", "lamp", and not
  ↪  the object instance name
  - All <obj_reference> must refer to an object
  ↪  category that is mentioned in the object count
  ↪  for the same scene
  - <arch_reference> can be one of the following:
  ↪  ["wall", "floor", "ceiling", "room", "window",
  ↪  "door"] or a specific room type, e.g.,
  ↪  "bedroom", "living_room", "kitchen", etc
    - It cannot be a specific instance of a category,
    ↪  e.g., "wall_1", "floor_2", "ceiling_3",
    ↪  "room_4", or "kitchen_5", etc
  - You do not need to specify something is on the
  ↪  floor as it is expected and adding one for each
  ↪  object is redundant
The difference between object-object relationships and
↪  object-architecture relationships is that the
↪  former is between two objects (or more if
↪  "surround"), while the latter is between an object
↪  and an architectural element.
For example, door, window, wall, floor, and ceiling
↪  are architectural elements and any relationship
↪  with them must be an object-architecture
↪  relationship.
Otherwise, if the relationship only involves objects,
↪  it is an object-object relationship.
{eq, lt, gt, le, ge} are the quantifier operators for
↪  equal, less than, greater than, less than or equal
↪  to, and greater than or equal to respectively.
```

F.3.2. In-Content Example

This prompt provides the VLM with a set of scene descriptions and their corresponding annotations as in-content examples.

```
show_examples: >
  Here are <NUM_EXAMPLES> manually annotated examples of
  ↪  the scene descriptions and their corresponding
  ↪  annotations:
  <IN_CONTEXT_EXAMPLES>
  Note that the annotations are in the same format as
  ↪  described above.
  The annotations are exhaustive and cover all the
  ↪  objects in the scene.
  Observe and learn from the examples, I will then ask
  ↪  you to generate annotations for a new scene
  ↪  description.
```

F.3.3. Generate Scene Description

This task asks the VLM to generate scene descriptions. There are two versions of the prompt: one for generating a single scene description and one for generating multiple scene descriptions.

```
generate_description: >
  Let's start a new cycle of generating scene
  ↪  descriptions and annotations.
  Your first task is to generate a scene description.
  There are three difficulty levels: easy, medium, and
  ↪  hard.
  - Easy:
    - Total number of all objects: <= 4
    - At most 4 large furniture objects (e.g., bed,
    ↪  sofa, table, chair)
    - Zero small objects (e.g., lamp, vase, book)
  - Medium:
    - Total number of all objects: 5 to 8
    - 0 to 3 small objects (e.g., lamp, vase, book)
    - Remaining objects are large furniture objects
    ↪  (e.g., bed, sofa, table, chair)
  - Hard:
    - Total number of all objects: >= 9
    - No limit on the number of small objects (e.g.,
    ↪  lamp, vase, book)
  The examples you have seen are of the same difficulty
  ↪  level as the one you are going to generate.
  Use your knowledge of the world and the examples you
  ↪  have seen to generate a scene description.
  But you do not need to base your scene description on
  ↪  the examples you have seen.
  Be creative, you do not need to follow the examples.
  Avoid viewpoint-dependent descriptions like left wall,
  ↪  right wall, etc as these are not meaningful in the
  ↪  context of a 3D environment.
  Only use ASCII characters in the scene description, no
  ↪  special characters.
  Have variety in the way you write. It can be started
  ↪  with "A" or "There is" at first but you should
  ↪  move away from them as you generate more
  ↪  descriptions.
  Use different sentence structures and avoid repetitive
  ↪  phrases.
  While having variety, you should also maintain
  ↪  clarity.
  Describe object styles and how they are arranged in
  ↪  the scene - object-object relationships and
  ↪  object-architecture relationships, as needed.
  Here are specific instructions from the user:
  --- Begin user instructions ---
  <INSTRUCTION>
  --- End user instructions ---
  Now, generate a <DIFFICULTY> scene description for the
  ↪  user.
  Respond in the given response schema. Here is an
  ↪  example response:
  ```
```
```

```
  {
    difficulty_level: "easy", (copy the difficulty level
    ↪  from the user)
    instruction: "Generate a simple bedroom", (copy the
    ↪  instruction from the user)
    generated_scene_description: "A bedroom with a red
    ↪  bed, and a nightstand on the left side of the
    ↪  bed with a wardrobe in the corner",
    total_num_objects: 3,
    num_large_objects: 3,
    num_small_objects: 0,
    reason: (explain how you generated the scene
    ↪  description)
  }
  ```
batch_generate_descriptions: >
  Let's start a new cycle of generating scene
  ↪ descriptions and annotations.
  Your first task is to generate some scene
  ↪ descriptions.
  There are three difficulty levels: easy, medium, and
  ↪ hard.
  - Easy:
    - Total number of all objects: <= 4
    - At most 4 large furniture objects (e.g., bed,
    ↪ sofa, table, chair)
    - Zero small objects (e.g., lamp, vase, book)
  - Medium:
    - Total number of all objects: 5 to 8
    - 0 to 3 small objects (e.g., lamp, vase, book)
    - Remaining objects are large furniture objects
    ↪ (e.g., bed, sofa, table, chair)
  - Hard:
    - Total number of all objects: >= 9
    - No limit on the number of small objects (e.g.,
    ↪ lamp, vase, book)
  The examples you have seen are of the same difficulty
  ↪ level as the ones you are going to generate.
  Use your knowledge of the world and the examples you
  ↪ have seen to generate scene descriptions.
  But you do not need to base your scene descriptions on
  ↪ the examples you have seen.
  Be creative, you do not need to follow the examples.
  Avoid viewpoint-dependent descriptions like left wall,
  ↪ right wall, etc as these are not meaningful in the
  ↪ context of a 3D environment.
  Only use ASCII characters in the scene description, no
  ↪ special characters.
  Have variety in the way you write. It can be started
  ↪ with "A" or "There is" at first but you should
  ↪ move away from them as you generate more
  ↪ descriptions.
  Use different sentence structures and avoid repetitive
  ↪ phrases.
  While having variety, you should also maintain
  ↪ clarity.
  Describe object styles and how they are arranged in
  ↪ the scene - object-object relationships and
  ↪ object-architecture relationships, as needed.
  Here are specific instructions from the user:
  --- Begin user instructions ---
  <INSTRUCTION>
  --- End user instructions ---
  Now, generate <NUM_DESCRIPTIONS> <DIFFICULTY> scene
  ↪ descriptions for the user.
  Respond in the given response schema. Here is an
  ↪ example response:
  ```
  [
    {
      difficulty_level: "easy", (copy the difficulty
      ↪  level from the user)
      instruction: "Generate a simple bedroom", (copy
      ↪  the instruction from the user)
      generated_scene_description: "A bedroom with a red
      ↪  bed, and a nightstand on the left side of the
      ↪  bed with a wardrobe in the corner",
      total_num_objects: 3,
      num_large_objects: 3,
      num_small_objects: 0,
      reason: (explain how you generated the scene
      ↪  description)
    },
    ...
  ]
  ```
```

### F.3.4. Generate Annotations

This task asks the VLM to generate annotations for the scene descriptions. There are two versions of the prompt: one for generating annotations for a single scene description and one for generating annotations for multiple scene descriptions.

```
generate_annotations: >
  Now, you are going to generate annotations for the
  ↪ scene description you have just generated.
  Use what you have learned from the in-context
  ↪ examples.
  The user may have edited the scene description, so
  ↪ here is the final scene description:
  --- Begin scene description ---
  <SCENE_DESCRIPTION>
  --- End scene description ---
  Read the scene description carefully and generate
  ↪ annotations for it.
  Before generating the annotations, go over the
  ↪ definitions of the four types of annotations
  ↪ again.
  After that, go over the examples again and understand
  ↪ how the annotations should be generated.
  The annotations should be exhaustive and cover all the
  ↪ objects in the scene.
  The annotations should be in the same format as
  ↪ described above.
  Use underscore instead of space or hyphen in all
  ↪ annotations.
  If the description mentions absence of an object, you
  ↪ should add an annotation for it (e.g.,
  ↪ "eq,0,chair").
  Note that if something is facing another object, not
  ↪ only can you use "front", you can also say
  ↪ "facing".
  Respond in the given response schema. Here is an
  ↪ example response:
  ```
  {
    scene_description: "a bedroom with a red bed, and a
    ↪  nightstand on the left side of the bed with a
    ↪  wardrobe in the corner",
    obj_counts: ["eq,1,bed", "eq,1,nightstand"],
    obj_attributes: ["eq,1,red,bed"],
    obj_obj_relationships:
    ↪  ["eq,1,left,0,bed,nightstand"],
    obj_arch_relationships:
    ↪  ["eq,1,corner,wardrobe,room"],
    reason: (explain how you generated the annotations)
  }
  ```
batch_generate_annotations: >
  Now, you are going to generate annotations for a set
  ↪ of scene descriptions you have just generated.
  Use what you have learned from the in-context
  ↪ examples.
  The user may have edited the scene descriptions, so
  ↪ here are the final scene descriptions:
  --- Begin all scene description ---
  <SCENE_DESCRIPTIONS>
  --- End all scene description ---
  Read the scene descriptions carefully and generate
  ↪ annotations for each of them.
  Before generating the annotations, go over the
  ↪ definitions of the four types of annotations
  ↪ again.
  After that, go over the examples again and understand
  ↪ how the annotations should be generated.
```
```

```
The annotations should be exhaustive and cover all the
↪  objects in the scenes.
The annotations should be in the same format as
↪  described above.
Use underscore instead of space or hyphen in all
↪  annotations.
If the description mentions absence of an object, you
↪  should add an annotation for it (e.g.,
↪  "eq,0,chair").
Note that if something is facing another object, not
↪  only can you use "front", you can also say
↪  "facing".
Order the annotations to be consistent with the order
↪  of the scene descriptions.
Respond in the given response schema. Here is an
↪  example response:
```
[
  {
    scene_description: "a bedroom with a red bed, and
    ↪ a nightstand on the left side of the bed with
    ↪ a wardrobe in the corner",
    obj_counts: ["eq,1,bed", "eq,1,nightstand"],
    obj_attributes: ["eq,1,red,bed"],
    obj_obj_relationships:
    ↪ ["eq,1,left,0,bed,nightstand"],
    obj_arch_relationships:
    ↪ ["eq,1,corner,wardrobe,room"],
    reason: (explain how you generated the
    ↪ annotations)
  },
  ...
]
```
```

G. Scientific Artifacts

The licenses for the datasets and code used in this work are as follows. For datasets, 3D-FRONT [17] and 3D-FUTURE [18] are available under their respective terms of use[1],[2]. Objaverse [13] is available under the ODC-By v1.0 license. For code, ATISS [41] is available under its NVIDIA Source Code License[3]. DiffuScene [51] is under its terms of use[4]. Holodeck [59], Long-CLIP [65] and Qwen2.5-VL [4] are available under Apache License 2.0. InstructScene [34] and LayoutGPT [15] are available under the MIT License. LayoutVLM [48] does not have a license specified in its repository. GPT-4o [1] and o4-mini [39] is under the OpenAI terms of use[5]. Our use of these datasets and code is in compliance with their respective licenses.

H. AI Assistant Usage

GPT-4o [1] is used in this work in parts of the evaluation framework, and o4-mini [39] is used in our data generation process. For experiments involving an open-source model, we used Qwen2.5-VL [4]. We also used GitHub Copilot[6] and ChatGPT[7] to assist in writing code and checking for grammatical errors and typos in the paper.

[1] 3D-FRONT Terms of Use
[2] 3D-FUTURE Terms of Use
[3] ATISS NVIDIA Source Code License
[4] DiffuScene Terms of Use
[5] https://openai.com/policies/terms-of-use/
[6] https://github.com/features/copilot
[7] https://chat.openai.com/