A. Details of HOLODECK

A.1. Efficiency and Cost

To create an interactive house of k rooms, HOLODECK uses $3+3\times k$ API calls. More specifically, utilizing OpenAI's gpt-4-1106-preview model incurs an approximate cost of \$ 0.2 per room. With our current implementation, HOLODECK can generate a single room in about 3 minutes. This includes the time for API calls and layout optimization using a MacBook equipped with an M1 chip.

A.2. Floor & Wall Modules

In the LLM outputs in the Floor Module, the following details are provided for each room:

- **room type**: the room's name, e.g., kitchen, bedroom.
- **floor material**: a description of the floor's appearance.
- **wall material**: a description of the wall's appearance.
- **vertices**: four tuples $\{(x_i, y_i), i \in [1, 2, 3, 4]\}$, representing the coordinates of the room's corners.

Material Selection. We have an image representation for each of 236 materials, consistent with the material setup in PROCTHOR [7][9]. Using CLIP[10] [42], we calculate the similarity between the material descriptions provided by the Large Language Model (LLM) and these images. The material with the highest similarity score is selected. Additionally, we utilize the 148 colors from Matplotlib [21] to refine the material selection by choosing the color closest to the description with CLIP.

Wall height. We have the LLM suggest a suitable wall height based on the user's input. For example, it may recommend a high ceiling for commercial spaces like museums.

A.3. Doorway & Window Modules

In HOLODECK, we take advantage of the diverse collection of doors and windows introduced in PROCTHOR [7], featuring a diverse collection of 40 doors (refer to examples in Figure 13) and 21 windows (see Figure 14). The LLM provides essential information to aid in the selection of doors:

- **room 1 & room 2**: the two rooms connected by the door, for example, bedroom and kitchen.
- **connection type**: one of the three connection types: *doorframe* (frame without a door), *doorway* (frame with a door), and *open* (no wall separating the rooms).
- **size**: the size of the door: *single* (one meter in width) or *double* (two meters in width).
- **door style**: a description of the door's appearance.

We have an image for each door, and we utilize CLIP to select the door that most closely matches the description.

We have the LLM provide the following data about windows:

[9]Procthor splits the set of materials into wall and floor materials. For HOLODECK, we merge them in one pool for retrieval.

[10]We employ OpenCLIP [22] with ViT-L/14, trained on the LAION-2B dataset [46], for all CLIP-related components in this paper.

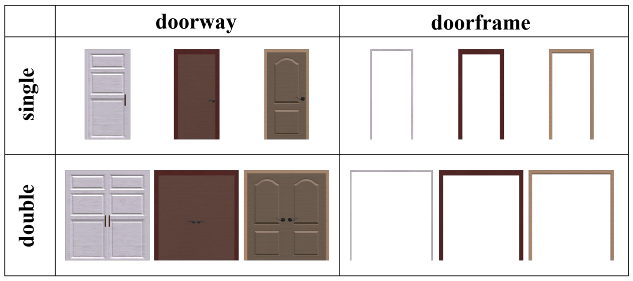


Figure 13. Examples of different doors in HOLODECK.

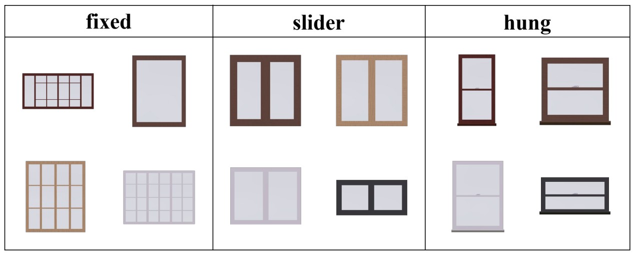


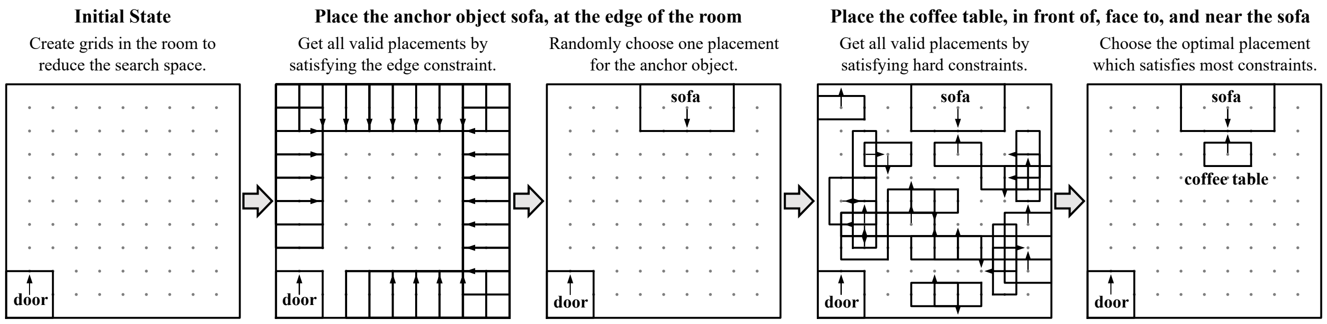
Figure 14. Examples of different windows in HOLODECK.

- **room type**: the room where the window will be installed.
- **direction**: the wall's direction (*south*, *north*, *east*, or *west*) where the window will be placed.
- **type**: one of the three window types: *fixed*, *slider* or *hung*.
- **size**: the width and height of the window.
- **quantity**: the number of windows installed on each wall.
- **height**: the distance from the floor to the window's base.

A.4. Object Selection Module

In Objaverse, each 3D asset $o \in \mathcal{O}$ is associated with the following metadata - a textual description of the asset t, the 3D bounding box size of the asset (w, d, h), and a set of 2D images I captured from three different angles (0°, 45°, and $-45°$). For each object proposed by LLM o', we have the LLM output a detailed description of the object (t') and its 3D bounding box size (w', d', h') for retrieval purposes. To evaluate the similarity between a candidate 3D asset in the repository $o = \left(t, (w, d, h), I\right)$ and the object proposed by the LLM o' $\left(t', (w', d', h')\right)$, we use three metrics:

- **Visual Similarity** ($\mathcal{V}$) measures the CLIP similarity between the 2D renderings of the candidate asset and the textual description of the LLM-proposed object: $\mathcal{V}(o, o') = \max_{i\in I} \text{CLIP}(i, t')$.
- **Textual Similarity** ($\mathcal{T}$) measures the similarity between the textual description of the candidate 3D asset and the textural description of the LLM-proposed object. This metric is crucial in improving the accuracy of the retrieval process since it ensures that we retrieve the asset within the correct category. We use the sentence transformer (SBERT) [44] with all-mpnet-base-v2 checkpoint to calculate the scores: $\mathcal{T} = \text{SBERT}(t, t')$.
- **Size Discrepancy** ($\mathcal{S}$) measures the discrepancy in the size of the 3D bounding box size of the candidate as-


Initial State
Create grids in the room to reduce the search space.
Place the anchor object sofa, at the edge of the room
Get all valid placements by satisfying the edge constraint.
Randomly choose one placement for the anchor object.
Place the coffee table, in front of, face to, and near the sofa
Get all valid placements by satisfying hard constraints.
Choose the optimal placement which satisfies most constraints.
door
sofa
coffee table
Figure 15. Example of using the DFS-based Constraint Satisfaction algorithm to place the objects.

set and the LLM-proposed object. There are similar objects with different sizes in the asset repository, and the size of objects is an important factor in designing scenes, e.g., we need a larger sofa for a large living room. The size matching score is computed as: $\mathcal{S}(o, o') = \left(|w - w'| + |h - h'| + |d - d'|\right)/3$. Two objects of similar size will have a smaller value of $\mathcal{S}$.

The overall matching score $\mathcal{M}\left(o, o'\right)$ is a weighted sum of the above metrics:

$$\mathcal{M}\left(o, o'\right) = \alpha \cdot \mathcal{V}\left(o, o'\right) + \beta \cdot \mathcal{T}(o, o') - \gamma \cdot \mathcal{S}(o, o') \quad (1)$$

with weights $\alpha = 100$, $\beta = 1$, and $\gamma = 10$. The asset with the highest matching score is selected.

A.5. Layout Design Module

In this module, we position the set of objects O chosen in Sec A.4, applying spatial relational constraints provided by the LLM. We define various constraints for floor objects:

- **Global constraint**: edge; middle.
- **Distance constraint**: near (object); far (object).
- **Position constraint**: in front of (object); side of (object).
- **Alignment constraint**: center align with (object).
- **Direction constraint**: face to (object).

The LLM can combine these constraints to form a constraint list C_o for each object $o \in O$. For instance, as shown in Figure 15, the constraints for a "coffee table" are [middle, in front of (sofa), face to (sofa), near (sofa)].

For floor object placement, we employ two solvers: Depth-First-Search (DFS) Solver and Mixed Integer Linear Programming (MILP) [2] Solver.

Depth-First-Search Solver. In the DFS solver, each object is defined by five variables $(x, y, w, d, \text{rotation})$. (x, y) is the 2D coordinates of the object's center, w and d are the width and depth of the 2D bounding box of the object, and rotation can be one of 0°, 90°, 180°, and 270°. The constraints listed above are treated softly, allowing certain violations when finding a layout. Beyond these soft constraints, we implement hard constraints essential for object placement: these constraints prevent object collisions and ensure that objects remain within the designated room boundaries. Violation of these hard constraints results in the object not being placed. Figure 15 demonstrates that our DFS solver initiates grids to establish a finite search space. It first explores different placements for the anchor object selected by the LLM. Subsequent steps involve optimizing the placement for the remaining objects, adhering to the hard constraints, and satisfying as many soft constraints as possible.[11] The algorithm can yield multiple solutions, with the final selection meeting the most constraints.

Mixed Integer Linear Programming (MILP) Solver is particularly effective for structured layout design. It optimizes a linear objective function subject to linear constraints with some non-discrete variables. This approach is well-suited for our layout optimization problem in HOLODECK.

In our MILP formulation, each object's position is determined by four variables: $(x, y, \text{rotate}_{90}, \text{rotate}_{180})$. The variables rotate_{90} and rotate_{180} are boolean, indicating rotations of 90 and 180 degrees, respectively. For example, if rotate_{90} and rotate_{180} are both true, it signifies a 270-degree rotation of the object. We translate all previously mentioned constraints into linear ones for the MILP problem. For instance, to align Object A with Object B at the center, a constraint in the form of $A_x = B_x$ or $A_y = B_y$ is implemented, where A_x, A_y and B_x, B_y represent the centers of Objects A and B, respectively. Note that the constraint is non-linear due to the OR operator. To model this linearly in MILP, we can introduce binary auxiliary variables and additional constraints to capture the logic of the OR condition. For solving the MILP, we utilize GUROBI [15], a state-of-the-art solver known for its efficiency and robustness.

In MILP solver, all constraints specified in the previous section are applied as hard constraints except that the Distance constraints (*near* and *far*) are uniquely modeled as part of the objective. For a visual comparison of these solvers' outcomes in HOLODECK, refer to Figure 24.

Wall & Small Objects. The placement of wall objects is determined by two specific attributes:

[11]The evaluation of an object's placement is based on the number of constraints satisfied. Placements that satisfy a greater number of constraints receive higher weights. However, any placement that violates hard constraints is rejected.

Asset ID: f1440a39fdab4f5282ad4a37fdcaa8c5

Category	video game console	**Front View**	180°
Synset	game_console.n.01	**Materials**	plastic metal glass
Width	10.2 cm		
Length	23.9 cm	**ONFLOOR**	False
Height	1.4 cm	**ONOBJECT**	True
Volume	341.16 cm^3	**ONWALL**	False
Mass	0.4 kg	**ONCEILING**	False
Description	This is a Nintendo Switch, a popular hybrid video game console that can be used both as a stationary and portable device. It has joy-con controllers attached on either side of the screen, one in blue and the other in red.		

Figure 16. Example of an asset's attributes annotated by GPT-4-V.

- **Above (Floor Object)**: This denotes the floor object directly underneath the wall object.
- **Height**: Specifies the exact distance from the floor to the base of the wall object, measured in centimeters.

To place small surface objects on top of larger objects, we first have LLM propose the placements and utilizeRandomSpawn[12] function in AI2-THOR. This method allows for randomized and efficient positioning of small objects on larger surfaces.

A.6. GPT-4-V for 3D Asset Annotation

We annotate the 3D assets used in HOLODECK with OpenAI's GPT-4-V API to enhance the accuracy of object retrieval and placement. As illustrated in Figure 16, GPT-4-V takes a set of four images as inputs, each showing an object from orthogonal rotations (0°, 90°, 180°, and 270°) and outputs the following attributes for the 3D object:

- **Category**: a specific classification of the object, such as "chair", "table", "building", etc.
- **Synset**: the nearest WordNet [34] synset will be used as the object type in object navigation tasks.
- **Width, Length, Height**: physical dimensions in centimeters, defining the object's bounding box sizes.
- **Volume**: approximate volume in cubic centimeters (cm^3).
- **Mass**: estimated object mass in kilograms (kg).
- **Front View**: an integer denoting the view representing the front of the object, often the most symmetrical view.
- **Description**: a detailed textual description of the object.
- **Materials**: a list of materials constituting the object.
- **Placement Attributes:** Boolean values (ONCEILING, ONWALL, ONFLOOR, ONOBJECT) indicating typical placement locations. For example, "True" for a ceiling fan's placement on the ceiling.

[12] AI2-THOR RandomSpawn Documentation

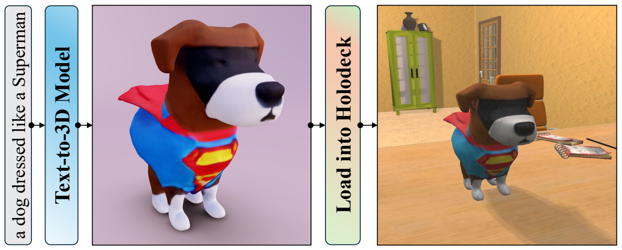


Figure 17. HOLODECK can import any 3D objects, including text-to-3D models generated (e.g., the object in this figure is generated by LumaAI [30]) to enhance object diversity.

A.7. Importing Objaverse Assets into AI2-THOR

The transformation of Objaverse assets into interactive objects in AI2-THOR involves a complex, multi-step pipeline.

Initially, the process starts with downloading and converting various 3D models into a mesh format optimized for runtime loading. We then generate visibility points on the mesh surface, enabling AI2-THOR to determine object visibility. This is followed by 3D decomposition, where the mesh is split into simpler convex meshes to facilitate rapid and realistic collision detection. The final step involves compressing textures (i.e., albedo, normal, and emission) and the model format to streamline performance.

Handling many assets in numerous scenes is challenging, mainly due to the large mesh counts of Objaverse assets and the traditional compile-time asset packaging approach of game engines like Unity. To address this, we implement caching layers for objects, reducing the loading time for repeated use in different scenes. Additionally, we develop a system to unload objects from memory, allowing efficient management of thousands of 3D objects at runtime.

Besides the objects from Objaverse, our automated pipeline can process any 3D objects, including those generated by text-to-3D models, as shown in Figure 17.

A.8. Rendering Options

As shown in Figure 18, HOLODECK scenes are rendered by Unity as default to train the embodied agents more efficiently. Users can also render HOLODECK scenes in Blender to achieve better realism.

A.9. Prompt

The complete prompt templates of HOLODECK's modules are provided in Figure 20 and 21. The prompt for annotating 3D assets using GPT-4-V is shown in Figure 22.

B. Qualitative Examples

In Figure 23, we showcase an additional 20 scenes generated by HOLODECK. These 20 scene types are chosen from the MIT dataset [41], distinct from examples in the main paper.

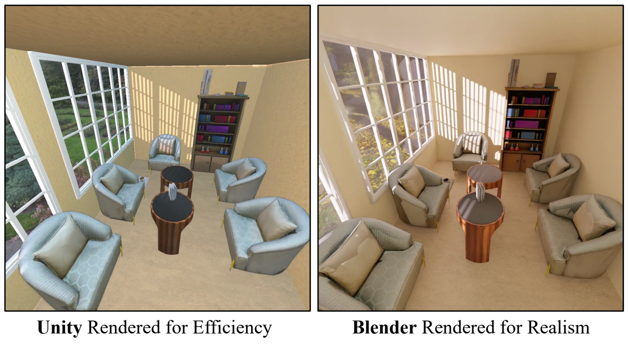

Unity Rendered for Efficiency
Blender Rendered for Realism

Figure 18. HOLODECK renders scenes with Unity by default for efficiency to facilitate Embodied AI applications. Blender can also be used to render HOLODECK scenes to improve realism.

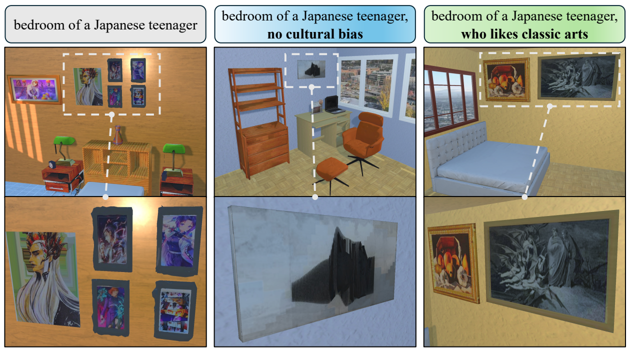

bedroom of a Japanese teenager
bedroom of a Japanese teenager, **no cultural bias**
bedroom of a Japanese teenager, **who likes classic arts**

Figure 19. We can address the cultural bias of GPT-4 by prompting.

Figure 24 presents a comparative analysis of layouts created by five methods. Figure 25 offers a visual comparison of residential scenes from iTHOR, PROCTHOR, and HOLODECK, highlighting the differences and capabilities of each system.

C. NOVELTYTHOR

NOVELTYTHOR comprises human-designed scenes crafted to challenge embodied agents in unique and diverse environments with a wide array of assets from Objaverse.

To integrate Objaverse assets into Unity, we developed tools that run a conversion pipeline on various operating systems, including macOS and Windows. This flexibility also enables the inclusion of assets other than those found in Objaverse. We designed a user-friendly interface for our artists and designers, facilitating asynchronous asset integration while optimizing storage efficiency.

The critical step of this process is the generation of Unity templates (prefabs) for the assets and their associated resources, leading to the creation of the scenes discussed in this paper. Figures 26 and 27 showcase top-down views of the 10 NOVELTYTHOR scenes, spanning five categories.

D. Cultural Bias

Cultural biases in HOLODECK generation can stem from biases in the LLM and the 3D asset retrieval component. For example, in Figure 19 (left), when the prompt contains culturally specific terms such as "Japanese", the generated scene may disproportionately feature prototypical objects like Manga posters. One mitigation strategy is to adjust the prompts, e.g., users can control the generation by simply adding a suffix like "no cultural bias" or making the prompt more detailed. This strategy is unlikely to fully remove bias, but these qualitative results suggest it can significantly help.

Floor plan Prompt: You are an experienced room designer. Please assist me in crafting a floor plan. Each room is a rectangle. You need to define the four coordinates and specify an appropriate design scheme, including each room's color, material, and texture. Assume the wall thickness is zero. Please ensure that all rooms are connected, not overlapped, and do not contain each other. The output should be in the following format: room name | floor material | wall material | vertices (coordinates). Note: the units for the coordinates are meters.
For example:
living room | maple hardwood, matte | light grey drywall, smooth | [(0, 0), (0, 8), (5, 8), (5, 0)]
kitchen | white hex tile, glossy | light grey drywall, smooth | [(5, 0), (5, 5), (8, 5), (8, 0)]

Here are some guidelines for you:
1. A room's size range (length or width) is 3m to 8m. The maximum area of a room is 48 m^2. Please provide a floor plan within this range and ensure the room is not too small or too large.
2. It is okay to have one room in the floor plan if you think it is reasonable.
3. The room name should be unique.

Now, I need a design for {input}.
Additional requirements: {additional_requirements}.

Your response should be direct and without additional text at the beginning or end.

Wall Height Prompt: I am now designing {input}. Please help me decide the wall height in meters. Answer with a number, for example, 3.0. Do not add additional text at the beginning or in the end.

Doorway Prompt: I need assistance in designing the connections between rooms. The connections could be of three types: doorframe (no door installed), doorway (with a door), or open (no wall separating rooms). The sizes available for doorframes and doorways are single (1m wide) and double (2m wide).

Ensure that the door style complements the design of the room. The output format should be: room 1 | room 2 | connection type | size | door style. For example:
exterior | living room | doorway | double | dark brown metal door
living room | kitchen | open | N/A | N/A
living room | bedroom | doorway | single | wooden door with white frames

The design under consideration is {input}, which includes these rooms: {rooms}.
The length, width, and height of each room in meters are: {room_sizes}
Certain pairs of rooms share a wall: {room_pairs}. There must be a door to the exterior.
Adhere to these additional requirements {additional_requirements}.

Provide your response succinctly, without additional text at the beginning or end.

Window Prompt: Guide me in designing the windows for each room. The window types are: fixed, hung, and slider.
The available sizes (width x height in cm) are:
fixed: (92, 120), (150, 92), (150, 120), (150, 180), (240, 120), (240, 180)
hung: (87, 160), (96, 91), (120, 160), (130, 67), (130, 87), (130, 130)
slider: (91, 92), (120, 61), (120, 91), (120, 120), (150, 92), (150, 120)

Your task is to determine the appropriate type, size, and quantity of windows for each room, bearing in mind the room's design, dimensions, and function.
Please format your suggestions as follows: room | wall direction | window type | size | quantity | window base height (cm from floor). For example: living room | west | fixed | (130, 130) | 1 | 50

I am now designing {input}. The wall height is {wall_height} cm.
The walls available for window installation (direction, width in cm) in each room are: {walls}
Please note: It is not mandatory to install windows on every available wall. Within the same room, all windows must be the same type and size. Also, adhere to these additional requirements: {additional_requirements}.

Provide a concise response, omitting any additional text at the beginning or end.

Figure 20. Prompt templates for Floor Module, Wall Module, Doorway Module, and Window Module.

Object Selection Prompt: You are an experienced room designer, please assist me in selecting large *floor*/*wall* objects and small objects on top of them to furnish the room. You need to select appropriate objects to satisfy the customer's requirements. You must provide a description and desired size for each object since I will use it to retrieve objects. If multiple identical items are to be placed in the room, please indicate the quantity and variance type (same or varied). Present your recommendations in JSON format:
{ object_name:{
"description": a short sentence describing the object,
"location": "floor" or "wall",
"size": the desired size of the object, in the format of a list of three numbers, [length, width, height] in centimeters,
"quantity": the number of objects (int),
"variance_type": "same" or "varied",
"objects_on_top": a list of small children objects (can be empty) which are placed *on top of* this object. For each child object, you only need to provide the object name, quantity and variance type. For example, {"object_name": "book", "quantity": 2, "variance_type": "varied"} } }

ONE-SHOT EXAMPLE

Currently, the design in progress is *INPUT*, and we are working on the *ROOM_TYPE* with the size of ROOM_SIZE. Please also consider the following additional requirements: REQUIREMENTS.
Here are some guidelines for you:
1. Provide reasonable type/style/quantity of objects for each room based on the room size to make the room not too crowded or empty.
2. Do not provide rug/mat, windows, doors, curtains, and ceiling objects which have been installed for each room.
3. I want at least 10 types of large objects and more types of small objects on top of the large objects to make the room look more vivid.

Please first use natural language to explain your high-level design strategy for *ROOM_TYPE*, and then follow the desired JSON format *strictly* (do not add any additional text at the beginning or end).

Layout Design Prompt: You are an experienced room designer. Please help me arrange objects in the room by assigning constraints to each object. Here are the constraints and their definitions:
1. global constraint:
1) edge: at the edge of the room, close to the wall, most of the objects are placed here.
2) middle: not close to the edge of the room.
2. distance constraint:
1) near, object: near to the other object, but with some distanbce, 50cm < distance < 150cm.
2) far, object: far away from the other object, distance >= 150cm.
3. position constraint:
1) in front of, object: in front of another object.
2) side of, object: on the side (left or right) of another object.
4. alignment constraint: 1) center aligned, object: align the center of the object with the center of another object.
5. Rotation constraint: 1) face to, object: face to the center of another object.

For each object, you must have one global constraint and you can select various numbers of constraints and any combinations of them and the output format must be: object | global constraint | constraint 1 | constraint 2 | ...
For example: sofa-0 | edge
coffee table-0 | middle | near, sofa-0 | in front of, sofa-0 | center aligned, sofa-0 | face to, sofa-0
tv stand-0 | edge | far, coffee table-0 | in front of, coffee table-0 | center aligned, coffee table-0 | face to, coffee table-0

Here are some guidelines for you:
1. I will use your guideline to arrange the objects *iteratively*, so please start with an anchor object which doesn't depend on the other objects (with only one global constraint).
2. Place the larger objects first.
3. The latter objects could only depend on the former objects.
4. The objects of the *same type* are usually *aligned*.
5. I prefer objects to be placed at the edge (the most important constraint) of the room if possible which makes the room look more spacious.
6. Chairs must be placed near to the table/desk and face to the table/desk.
Now I want you to design {room_type} and the room size is {room_size}.
Here are the objects that I want to place in the {room_type}: {objects}
Please first use natural language to explain your high-level design strategy, and then follow the desired format *strictly* (do not add any additional text at the beginning or end) to provide the constraints for each object.

Figure 21. Prompt templates for Object Selection Module and Layout Design Module.

3D Asset annotation Prompt: Please annotate this 3D asset with the following values (output valid JSON):
"annotations": {
"category": a category such as "chair", "table", "building", "person", "airplane", "car", "seashell", "fish", etc. Try to be more specific than "furniture",
"synset": the synset of the object that is most closely related. This could be "cat.n.01", "glass.n.03", "bank.n.02",
"width": approximate width in cm. For a human being, this could be "45",
"length": approximate length in cm. For a human being, this could be "25",
"height": approximate height in cm. For a human being, this could be "182",
"volume": approximate volume in cm^3. For a human being, this could be "62000",
"mass": approximate mass in kilogram. For a human being, this could be "72",
"frontView": which of the views represents the front of the object (value should be an integer with the first image being 0). Note that the front view of an object, including furniture, tends to be the view that exhibits the highest degree of symmetry,
"description": a description of the object (don't use the term "3D asset" here),
"materials": a Python list of the materials that the object appears to be made of (roughly in order of most used material to least used),
"onCeiling": whether this object can appear on the ceiling or not, return true or false with no explanations. This would be true for a ceiling fan but false for a chair,
"onWall": whether this object can appear on the wall or not, return true or false with no explanations. This would be true for a painting but false for a table,
"onFloor": whether this object can appear on the floor or not, return true or false with no explanations. This would be true for a piano but false for a curtain,
"onObject": whether this object can appear on another object or not, return true or false with no explanations. This would be true for a laptop but not for a sofa }

Please output the JSON now.

Figure 22. Prompt template for annotating 3D assets with GPT-4-V.

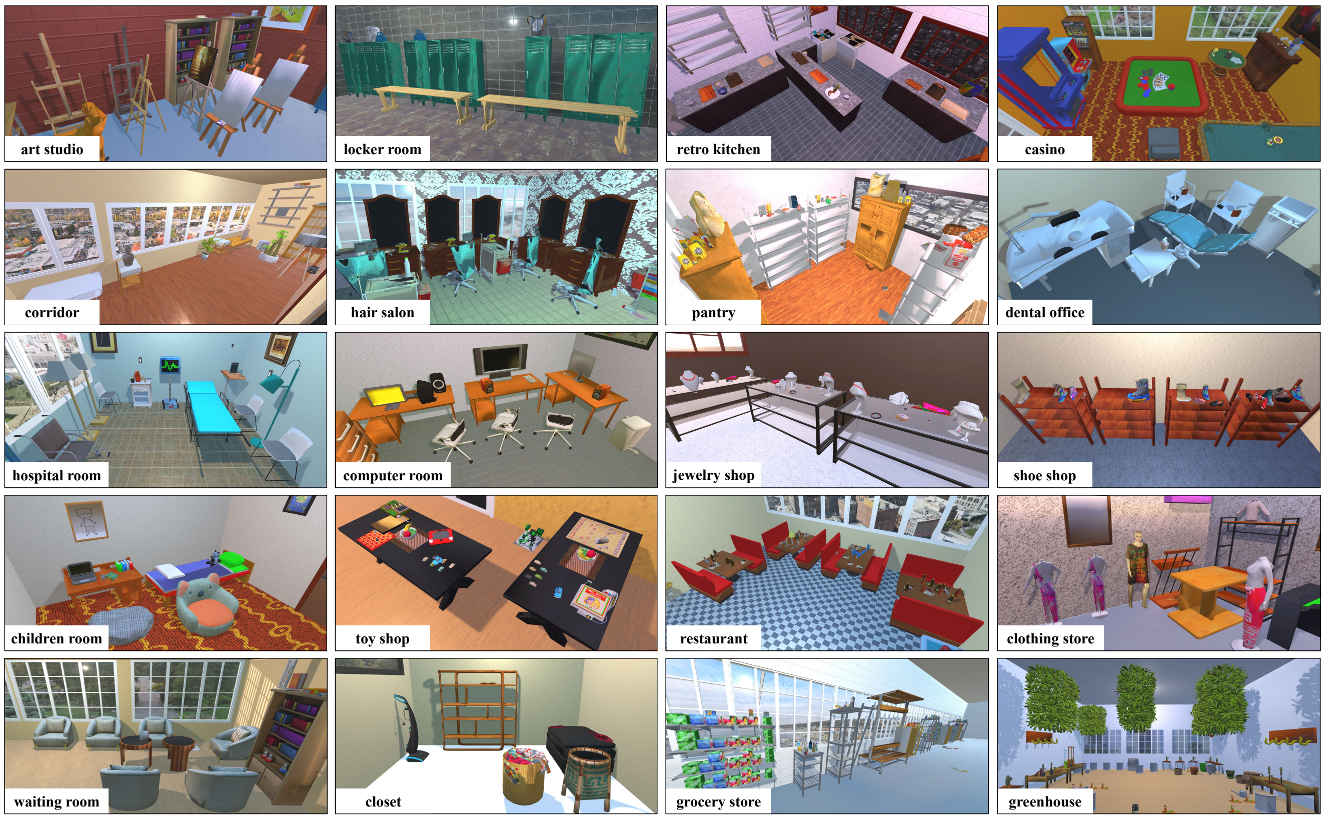


Figure 23. Additional examples of different scene types from MIT Scenes [41].

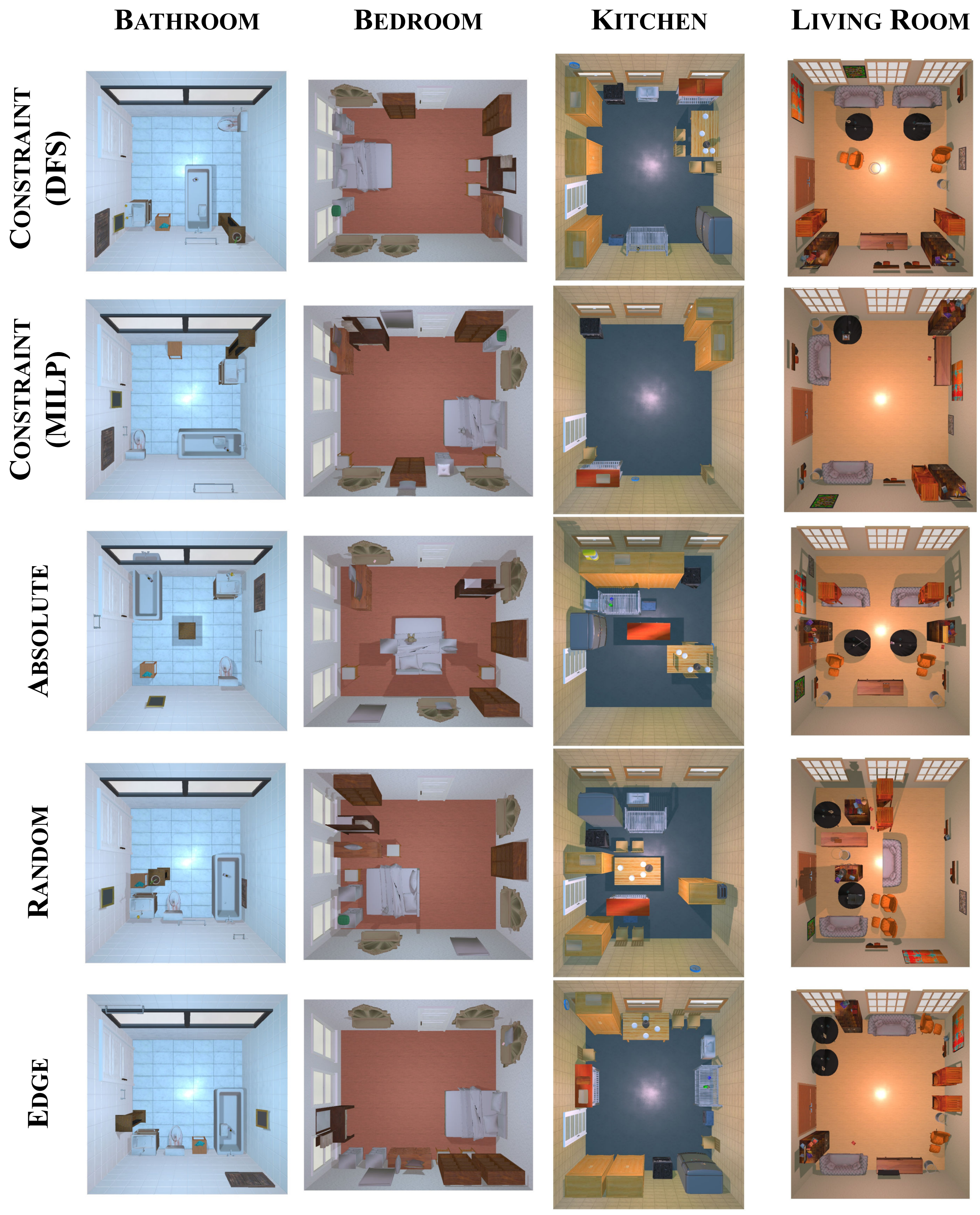


Figure 24. Qualitative comparison of the different layout methods.

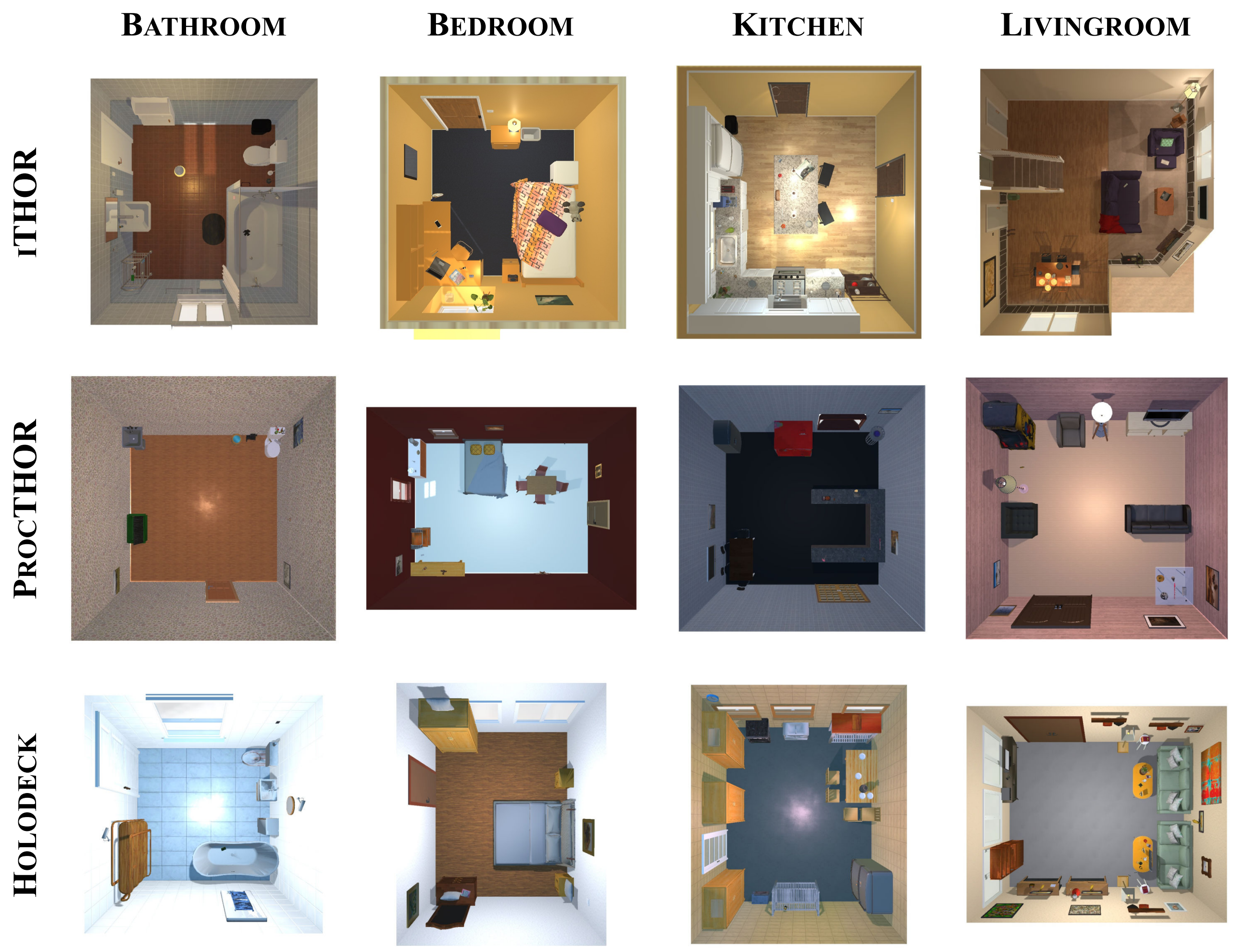


Figure 25. Qualitative examples on four types of residential scenes from iTHOR, PROCTHOR, and HOLODECK.

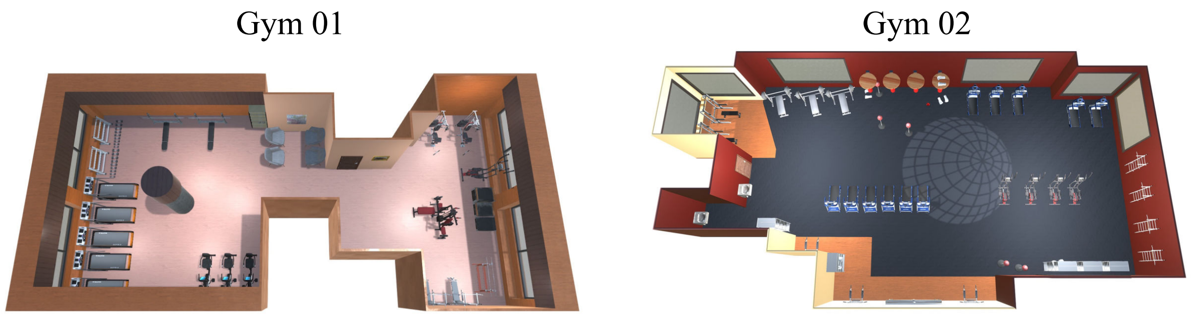


Figure 26. Top-down view of NOVELTYTHOR. Each scene type has two instances.

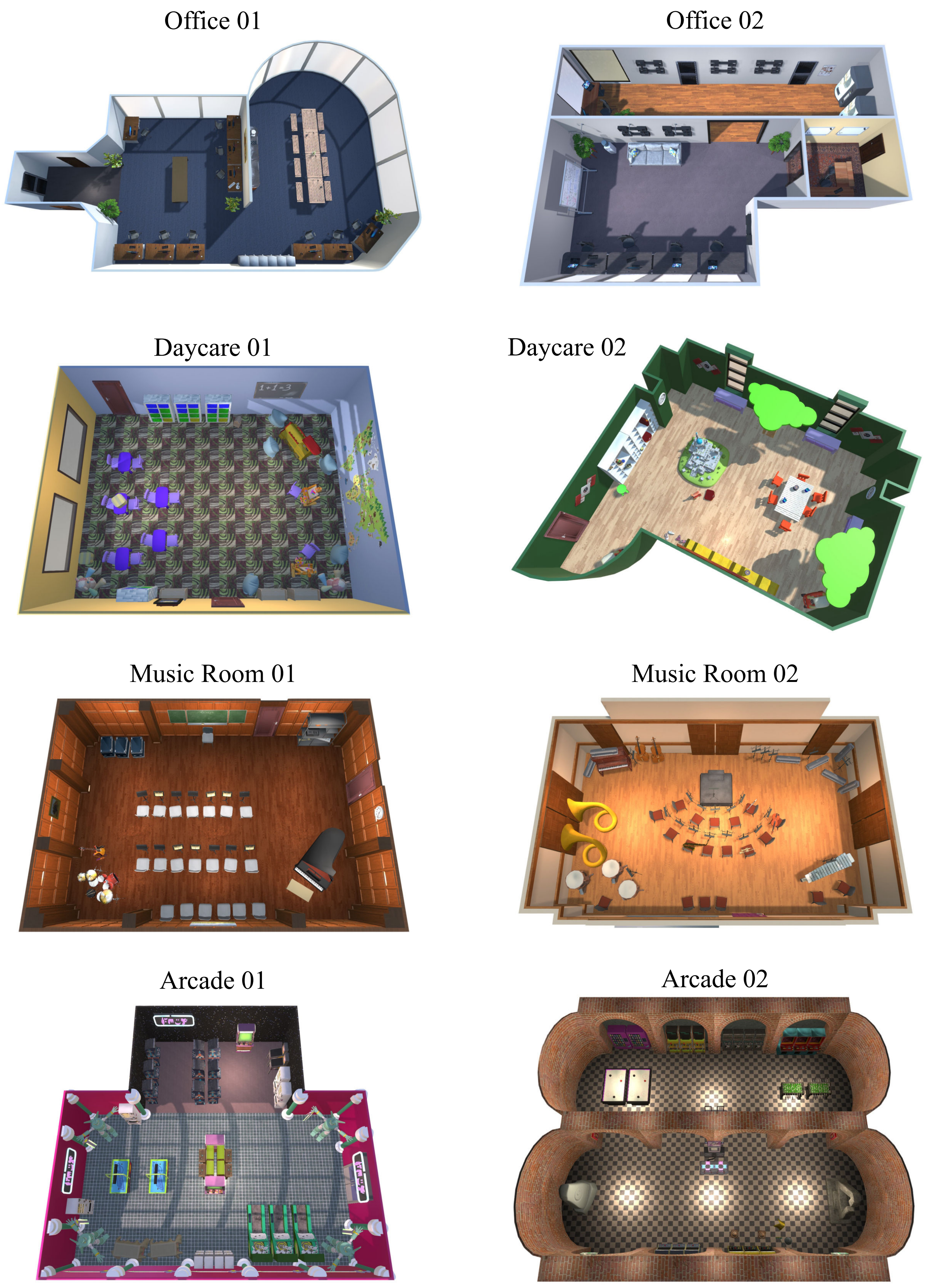


Figure 27. Top-down view of NOVELTYTHOR (continued).