# HOLODECK: Language Guided Generation of 3D Embodied AI Environments

Yue Yang*[1], Fan-Yun Sun*[2], Luca Weihs*[4], Eli Vanderbilt[4], Alvaro Herrasti[4],
Winson Han[4], Jiajun Wu[2], Nick Haber[2], Ranjay Krishna[3,4], Lingjie Liu[1],
Chris Callison-Burch[1], Mark Yatskar[1], Aniruddha Kembhavi[3,4], Christopher Clark[4]

[1]University of Pennsylvania, [2]Stanford University,
[3]University of Washington, [4]Allen Institute for Artificial Intelligence

yueyang1996.github.io/holodeck/

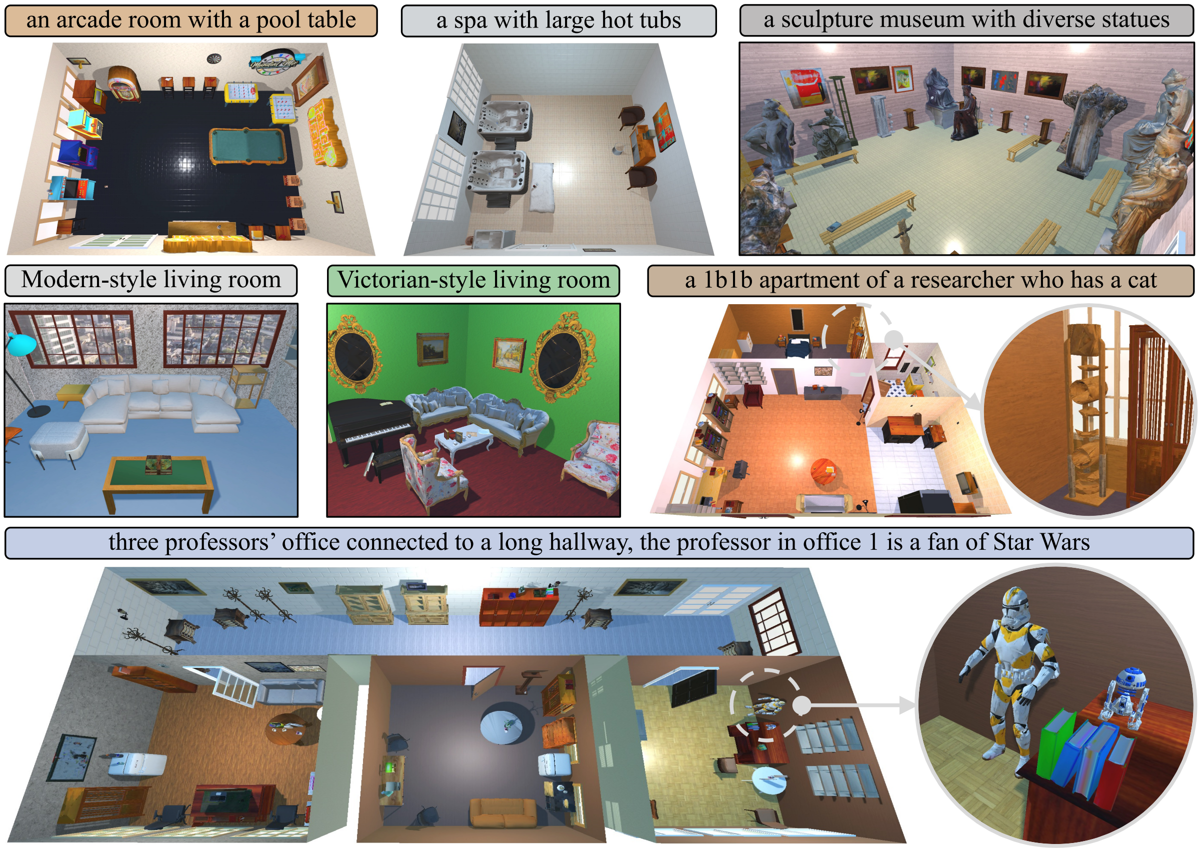

an arcade room with a pool table
a spa with large hot tubs
a sculpture museum with diverse statues
Modern-style living room
Victorian-style living room
a 1b1b apartment of a researcher who has a cat
three professors' office connected to a long hallway, the professor in office 1 is a fan of Star Wars

Figure 1. Example outputs of HOLODECK—a large language model powered system, which can generate diverse types of environments (arcade, spa, museum), customize for styles (Victorian-style), and understand fine-grained requirements ("has a cat", "fan of Star Wars").

## Abstract

*3D simulated environments play a critical role in Embodied AI, but their creation requires expertise and extensive manual effort, restricting their diversity and scope. To mitigate this limitation, we present* HOLODECK*, a system that generates 3D environments to match a user-supplied prompt fully automatedly.* HOLODECK *can generate diverse scenes, e.g., arcades, spas, and museums, adjust the designs for styles, and can capture the semantics of complex queries such as "apartment for a researcher with a cat" and "office of a professor who is a fan of Star Wars".* HOLODECK *leverages a large language model (GPT-4) for common sense knowledge about what the scene might look like and uses a large collection of 3D assets from Objaverse to populate the scene with diverse objects. To address the challenge of positioning objects correctly, we prompt GPT-4 to generate spatial relational constraints between objects and then optimize the layout to satisfy those constraints. Our large-scale*

*Equal technical contribution. Work done while at PRIOR@AI2.

*human evaluation shows that annotators prefer* Holodeck *over manually designed procedural baselines in residential scenes and that* Holodeck *can produce high-quality outputs for diverse scene types. We also demonstrate an exciting application of* Holodeck *in Embodied AI, training agents to navigate in novel scenes like music rooms and daycares without human-constructed data, which is a significant step forward in developing general-purpose embodied agents.*

## 1. Introduction

The predominant approach in training embodied agents involves learning in simulators [8, 23, 25, 38, 43, 54]. Generating realistic, diverse, and interactive 3D environments plays a crucial role in the success of this process.

Existing Embodied AI environments are typically crafted through manual design [6, 13, 25, 26], 3D scanning [8, 41, 43], or procedurally generated with hard-coded rules [7]. However, these methods require considerable human effort that involves designing a complex layout, using assets supported by an interactive simulator, and placing them into scenes while ensuring semantic consistency between the different scene elements. Therefore, prior work on producing 3D environments mainly focuses on limited environment types. To move beyond these limitations, recent works adapt 2D foundational models to generate 3D scenes from text [11, 18, 56]. However, these models often produce scenes with significant artifacts, such as mesh distortions, and lack the interactivity necessary for Embodied AI. Moreover, there are models tailored for specific tasks like floor plan generation [19, 45] or object arrangement [36, 52]. Although effective in their respective domains, they lack overall scene consistency and rely heavily on task-specific datasets.

In light of these challenges, we present **Holodeck**, a language-guided system built upon AI2-THOR [25], to automatically generate diverse, customized, and interactive 3D embodied environments from textual descriptions. Shown in Figure 2, given a description (e.g., *a 1b1b apartment of a researcher who has a cat*), Holodeck uses a Large Language Model (GPT-4 [35]) to design the floor plan, assign suitable materials, install the doorways and windows and arrange 3D assets coherently in the scene using constraint-based optimization. Holodeck chooses from over 50K diverse and high-quality 3D assets from Objaverse [9] to satisfy a myriad of environment descriptions.

Motivated by the emergent abilities of Large Language Models (LLMs) [51], Holodeck exploits the commonsense priors and spatial knowledge inherently present in LLMs. This is exemplified in Figure 1, where Holodeck creates diverse scene types such as *arcade*, *spa* and *museum*, interprets specific and abstract prompts by placing relevant objects appropriately into the scene, e.g., an "R2-D2"[1] on the desk for "a fan of Star Wars". Beyond object selection and layout design, Holodeck showcases its versatility in style customization, such as creating a scene in a "Victorian-style" by applying appropriate textures and designs to the scene and its objects. Moreover, Holodeck demonstrates its proficiency in spatial reasoning, like devising floor plans for "three professors' offices connected by a long hallway" and having regular arrangements of objects in the scenes. Overall, Holodeck offers a broad coverage approach to 3D environment generation, where textual prompts unlock new levels of control and flexibility in scene creation.

The effectiveness of Holodeck is assessed through its scene generation quality and applicability to Embodied AI. Through large-scale user studies involving 680 participants, we demonstrate that Holodeck significantly surpasses existing procedural baseline ProcTHOR [7] in generating residential scenes and achieves high-quality outputs for various scene types. For the Embodied AI experiments, we focus on Holodeck's application in aiding zero-shot object navigation in previously unseen scene types. We show that agents trained on scenes generated by Holodeck can navigate better in novel environments (e.g., *Daycare* and *Gym*) designed by experts.

To summarize, our contributions are three-fold: (1) We propose Holodeck, a language-guided system capable of generating diverse, customized, and interactive 3D environments based on textual descriptions; (2) The human evaluation validates Holodeck's capability of generating residential and diverse scenes with accurate asset selection and realistic layout design; (3) Our experiments demonstrate that Holodeck can aid Embodied AI agents in adapting to new scene types and objects during object navigation tasks.

## 2. Related Work

**Embodied AI Environments.** Previous work mainly relies on 3D artists to design the environments [6, 13, 25, 26, 38, 54], which is hard to scale up or construct scenes from 3D scans [41, 43, 46] to reduce human labor, but the scenes are less interactive. The procedural generation framework ProcTHOR [7] showcases its potential to generate large-scale interactive environments for training embodied agents. Phone2Proc [8] uses a phone scan to create training scenes that are semantically similar to the desired real-world scene. A concurrent work, RoboGen [50], proposes to train robots by generating diversified tasks and scenes. These works parallel our concept, Holodeck, which aims to train generalizable embodied agents and presents an avenue for further exploration in text-driven 3D interactive scene generation.

**Large Language Model for Scene Design.** Many works on scene design either learn spatial knowledge priors from existing 3D scene databases [4, 29, 47–49, 52, 57] or leverage user input and refine the 3D scene iteratively [3, 5]. However, having to learn from datasets of limited categories such as

[1] A fictional robot character in the Star Wars.

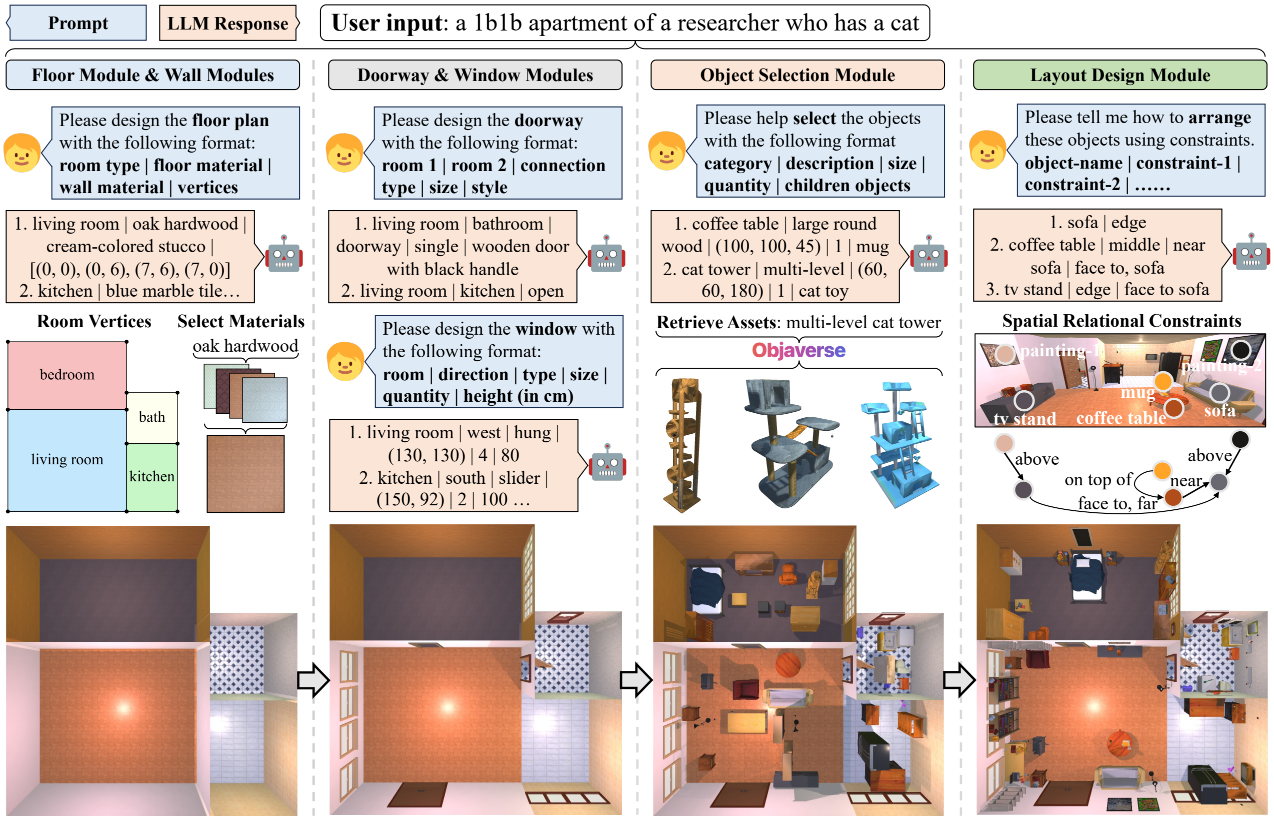

Prompt
LLM Response
User input: a 1b1b apartment of a researcher who has a cat
Floor Module & Wall Modules
Doorway & Window Modules
Object Selection Module
Layout Design Module
Please design the floor plan with the following format: room type | floor material | wall material | vertices
1. living room | oak hardwood | cream-colored stucco | [(0, 0), (0, 6), (7, 6), (7, 0)] 2. kitchen | blue marble tile…
Room Vertices
Select Materials
oak hardwood
bedroom
bath
living room
kitchen
Please design the doorway with the following format: room 1 | room 2 | connection type | size | style
1. living room | bathroom | doorway | single | wooden door with black handle 2. living room | kitchen | open
Please design the window with the following format: room | direction | type | size | quantity | height (in cm)
1. living room | west | hung | (130, 130) | 4 | 80 2. kitchen | south | slider | (150, 92) | 2 | 100 …
Please help select the objects with the following format category | description | size | quantity | children objects
1. coffee table | large round wood | (100, 100, 45) | 1 | mug 2. cat tower | multi-level | (60, 60, 180) | 1 | cat toy
Retrieve Assets: multi-level cat tower
Objaverse
Please tell me how to arrange these objects using constraints. object-name | constraint-1 | constraint-2 | ……
1. sofa | edge 2. coffee table | middle | near sofa | face to, sofa 3. tv stand | edge | face to sofa
Spatial Relational Constraints
painting-1
painting-2
mug
tv stand
coffee table
sofa
above
above
on top of
near
face to, far

Figure 2. Given a text input, HOLODECK generates the 3D environment through multiple rounds of conversation with an LLM.

3D-FRONT [12] restricts their applicability. Recently, Large Language Models (LLMs) were shown to be useful in generating 3D scene layouts [10, 28]. However, their methods of having LLMs directly output numerical values can yield layouts that defy physical plausibility (e.g., overlapping assets). In contrast, HOLODECK uses LLMs to sample spatial relational constraints and a solver to optimize the layout, ensuring physically plausible scene arrangements. Our human study shows a preference for HOLODECK-generated layouts over those generated end-to-end by LLMs. (see Sec 4.3).

**Text-driven 3D Generation.** Early endeavors in 3D generation focus on learning the distribution of 3D shapes and/or textures from category-specific datasets [16, 33, 53, 55, 58]. Subsequently, the advent of large vision-language models like CLIP [40] enables zero-shot generation of 3D textures and objects [14, 20, 27, 30, 31, 37]. These works excel at generating 3D objects but struggle to generate complex 3D scenes. More recently, emerging works generate 3D scenes by combining pre-trained text-to-image models with depth prediction algorithms to produce either textured meshes or NeRFs [11, 18, 56]. However, these approaches yield 3D representations that lack modular composability and interactive affordances, limiting their use in embodied AI. In contrast, HOLODECK utilizes a comprehensive 3D asset database to generate semantically precise, spatially efficient, and interactive 3D environments suitable for training embodied agents.

## 3. HOLODECK

HOLODECK is a promptable system based on AI2-THOR [7, 25], enriched with massive assets from Objaverse [9], which can produce diverse, customized, and interactive Embodied AI environments with the guidance of large language models.

As shown in Figure 2, HOLODECK employs a systematic approach to scene construction, utilizing a series of specialized modules: (1) the *Floor & Wall Module* develop floor plans, constructs wall structures and selects appropriate materials for the floors and walls; (2) the *Doorway & Window Module* integrates doorways and windows into the created environment; (3) the *Object Selection Module* retrieves appropriate 3D assets from Objaverse, and (4) the *Constraint-based Layout Design Module* arranges the assets within the scene by utilizing spatial relational constraints to ensure that the layout of objects is realistic.

In the following sections, we introduce our prompting approach that converts high-level user natural language specifications into a series of language model queries for constructing layouts. We then provide a detailed overview of each module shown in Figure 2 and how they contribute to the final scene. Finally, we illustrate how HOLODECK leverages Objaverse assets to ensure diversity in scene creation and efficiency for Embodied AI applications. Comprehensive details of HOLODECK can be found in the supplements.

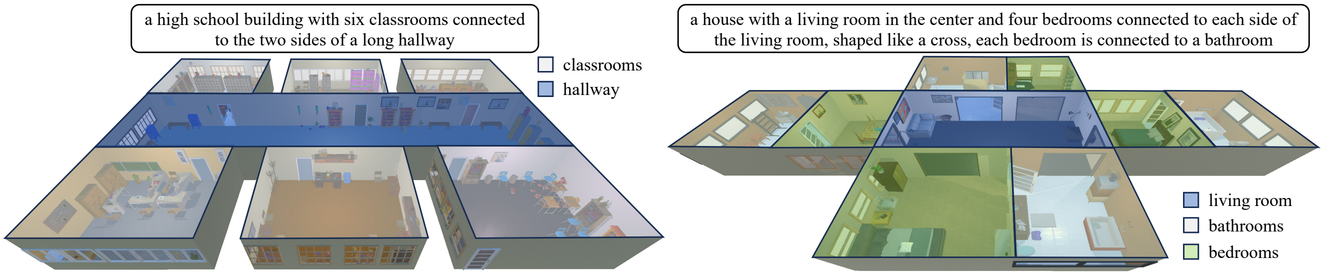

a high school building with six classrooms connected to the two sides of a long hallway
□ classrooms
■ hallway
a house with a living room in the center and four bedrooms connected to each side of the living room, shaped like a cross, each bedroom is connected to a bathroom
■ living room
□ bathrooms
🟩 bedrooms

Figure 3. **Floorplan Customizability.** Holodeck can interpret complicated input and craft reasonable floor plans correspondingly.

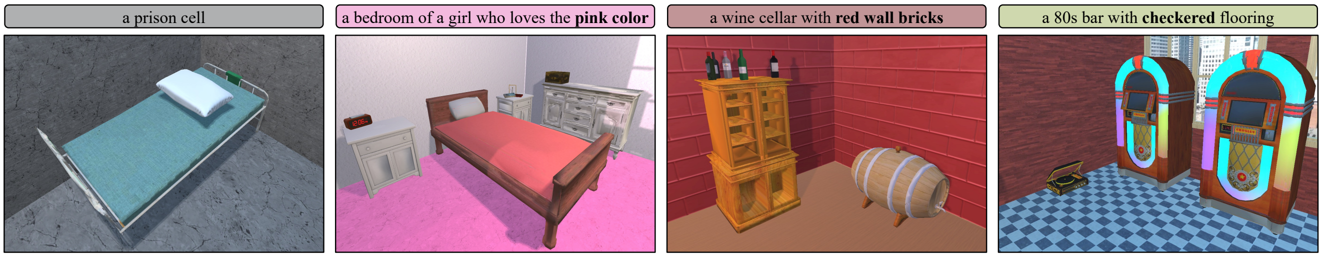

a prison cell
a bedroom of a girl who loves the **pink color**
a wine cellar with **red wall bricks**
a 80s bar with **checkered** flooring

Figure 4. **Material Customizability.** Holodeck can adapt the floor and wall materials to make the scenes more realistic.

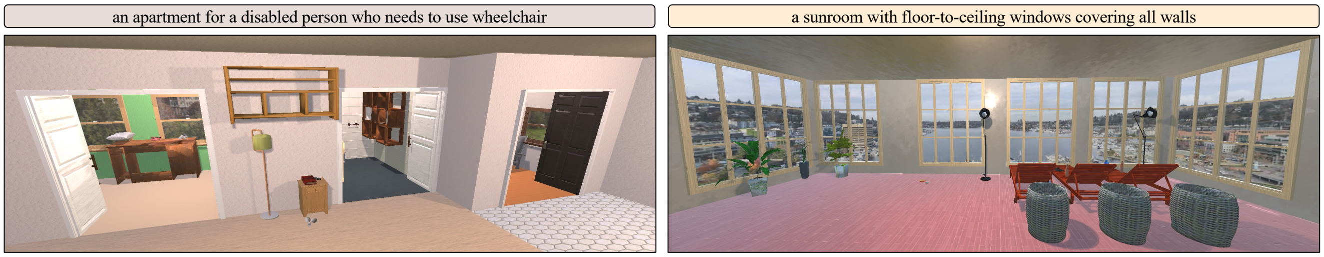

an apartment for a disabled person who needs to use wheelchair
a sunroom with floor-to-ceiling windows covering all walls

Figure 5. **Door & window Customizability.** Holodeck can adjust the size, quantity, position, etc., of doors & windows based on the input.

**Overall Prompt Design.** Each module in Figure 2 takes information from a language model and converts it to elements included in the final layout. An LLM prompt is designed for each module with three elements: (1) *Task Description*: outlines the context and goals of the task; (2) *Output Format*: specifies the expected structure and type of outputs and (3) *One-shot Example*: a concrete example to assist the LLM's comprehension of the task. The text within the blue dialog boxes of Figure 2 represents examples of simplified prompts[2]. LLM's high-level responses to these prompts are post-processed and then used as input arguments for the modules to yield low-level specifications of the scene.

The **Floor & Wall Module**, illustrated in the first panel of Figure 2, is responsible for creating floor plans, constructing wall structures, and selecting materials for floors and walls. Each room is represented as a rectangle, defined by four tuples that specify the coordinates of its corners. GPT-4 directly yields the coordinates for placing the rooms and suggests realistic dimensions and connectivity for these rooms. Figure 3 illustrates several examples of diverse layouts this module proposes where Holodeck generates prompt-appropriate, intricate, multi-room floor plans.

This module also chooses materials for the floors and walls, which is crucial for enhancing the realism of environments. Holodeck can match LLM proposals to one of 236 materials, each available in 148 colors, enabling semantic customization of scenes. As shown in Figure 4, Holodeck can generate scenes with suitable materials based on the type of scene, such as opting for concrete walls and floors in a *prison cell* scenario. Inputs with specific texture requirements are often reflected in the final design, for example, "pink color", "red wall bricks," and "checkered floor".

The **Doorway & Window Module**, illustrated in the second panel of Figure 2, is responsible for proposing room connections and windows. Each of these two properties is queried separately from the LLM. The LLM can propose doorways and windows that match 40 door styles and 21 window types, each of which can be modified by several properties, including size, height, quantity, etc. For instance, Figure 5 shows Holodeck's tailored designs on doors and windows, such as wider doors for "wheelchair accessibility" and multiple floor-to-ceiling windows in a "sunroom" setting.

[2] The complete prompts (available in the supplementary materials) include additional guidance for LLMs to avoid common errors we observe. For example, by adding a sentence, "the minimal area per room is 9 m$^2$", Holodeck can avoid generating overly small rooms.

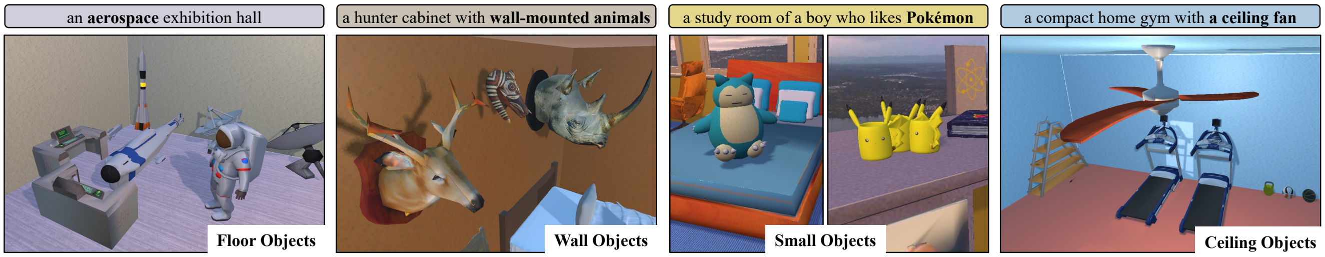

an aerospace exhibition hall
a hunter cabinet with wall-mounted animals
a study room of a boy who likes Pokémon
a compact home gym with a ceiling fan
Floor Objects
Wall Objects
Small Objects
Ceiling Objects

Figure 6. **Objects Customizability.** HOLODECK can select and place appropriate floor/wall/small/ceiling objects conditioned on the input.

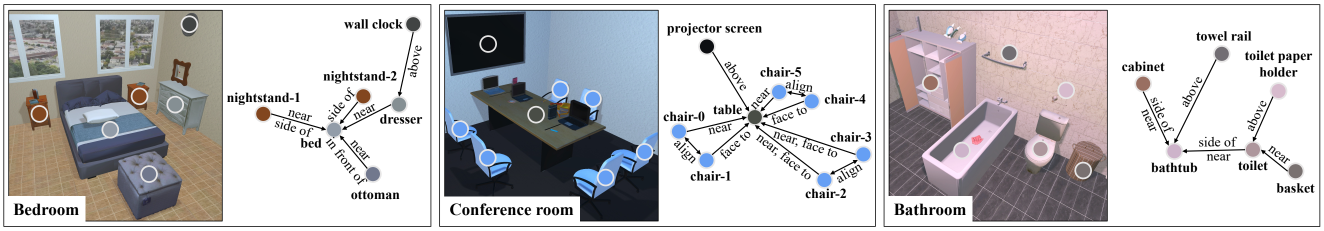


Figure 7. Examples of **Spatial Relational Constraints** generated by LLM and their solutions found by our constraint satisfaction algorithm.

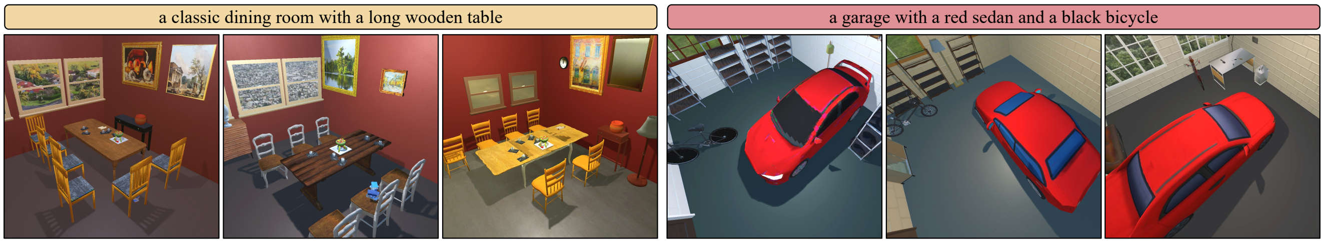

a classic dining room with a long wooden table
a garage with a red sedan and a black bicycle

Figure 8. **Output Diversity.** HOLODECK can generate **multiple variants** for the same input with different assets and layouts.

The **Object Selection Module**, illustrated in the third panel of Figure 2, allows the LLM to propose objects that should be included in the layout. Leveraging the extensive Objaverse asset collection, HOLODECK can fetch and place diverse objects in the scene. Queries are constructed with LLM-proposed descriptions and dimensions, like "multi-level cat tower, 60 × 60 × 180 (cm)" to retrieve the optimal asset from Objaverse. The retrieval function[3] considers visual and textual similarity and dimensions to ensure the assets match the design. Figure 6 shows the capability of HOLODECK to customize diverse objects on the floor, walls, on top of other items, and even on the ceiling.

The **Constraint-based Layout Design Module**, illustrated in the fourth panel of Figure 2, generates the positioning and orientation of objects. Previous work [10] shows LLM can directly provide the absolute value of the object's bounding box. However, when attempting to place a diverse lot of assets within environments, this method frequently leads to out-of-boundary errors and object collisions. To address this, instead of letting LLM directly operate on numerical values, we propose a novel constraint-based approach that employs LLM to generate spatial relations between the objects, e.g., "coffee table, in front of, sofa", and optimize the layout based on the constraints. Given the probabilistic nature of LLMs, HOLODECK can yield multiple valid layouts given the same prompt as shown in Figure 8.

*Spatial Relational Constraints.* We predefined ten types of constraints, organized into five categories: (1) Global: *edge*, *middle*; (2) Distance: *near*, *far*; (3) Position: *in front of*, *side of*, *above*, *on top of*; (4) Alignment: *center aligned* and (5) Rotation: *face to*. LLM selects a subset of constraints for each object, forming a scene graph for the room (examples shown in Figure 7). Those constraints are treated softly, allowing for certain violations when finding a layout to satisfy all constraints is not feasible. Besides those soft constraints, we enforce hard constraints to prevent object collisions and ensure that all objects are within the room's boundaries.

*Constraint Satisfaction.* We first reformulate the spatial relational constraints defined above into mathematical conditions (e.g., two objects are center-aligned if they share the same $x$ or $y$ coordinate). To find layouts that satisfy constraints sampled by LLMs, we adopt an optimization algorithm to place objects autoregressively. The algorithm first uses LLM to identify an anchor object and then explores placements for the anchor object. Subsequently, it employs Depth-First-Search (DFS)[4] to find valid placements for the remaining

[3] We use CLIP [40] to measure the visual similarity, Sentence-BERT [42] for the textual similarity, and 3D bounding box sizes for the dimension.

[4] Given the linear nature of constraints, a Mixed Integer Linear Programming (MILP) solver can also be employed. While we assume the DFS

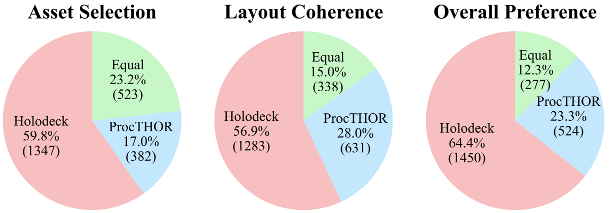


Figure 9. Comparative human evaluation of HOLODECK and PROCTHOR across three criteria. The pie charts show the distribution of annotator preferences, showing both the percentage and the actual number of annotations favoring each system.

objects. A placement is only valid if all the hard constraints are satisfied. For example, in Figure 7, *bed* is selected as the anchor object in the *bedroom*, and the *nightstands* are placed subsequently. The algorithm is executed for a fixed time (30 seconds) to get multiple candidate layouts and return the one that satisfies the most total constraints. We verify the effectiveness of our constraint-based layout in Sec 4.3.

**Leveraging Objaverse Assets**, HOLODECK is able to support the creation of diverse and customized scenes. We curate a subset of assets suitable for indoor design from Objaverse 1.0. These assets are further annotated by GPT-4-Vison [34] automatically with additional details, including textual descriptions, scale, canonical views, etc.[5] Together with the assets from PROCTHOR, our library encompasses 51,464 annotated assets. To import Objaverse assets into AI2-THOR for embodied AI applications, we optimize the assets by reducing mesh counts to minimize the loading time in AI2-THOR, generating visibility points and colliders. More details on importing Objaverse assets into AI2-THOR are available in the supplementary materials.

In the following sections, we will evaluate the quality and utility of the scenes generated by HOLODECK.

## 4. Human Evaluation

We conduct comprehensive human evaluations to assess the quality of HOLODECK scenes, with a total of 680 graduate students participating in three user studies: (1) a comparative analysis on **residential scenes** with PROCTHOR as the baseline; (2) an examination of HOLODECK's ability in generating **diverse scenes**, and (3) an ablation study to validate the effectiveness of our **layout design** method. Through these user studies, we demonstrate that HOLODECK can create residential scenes of better quality than previous work while being able to extend to a wider diversity of scene types.

### 4.1. Comparative Analysis on Residential Scenes

This study collects human preference scores to compare HOLODECK with PROCTHOR [7], the sole prior work capable of generating complete, interactable scenes. Our comparison focuses on residential scenes, as PROCTHOR is limited to four types: *bathroom*, *bedroom*, *kitchen*, and *living room*.

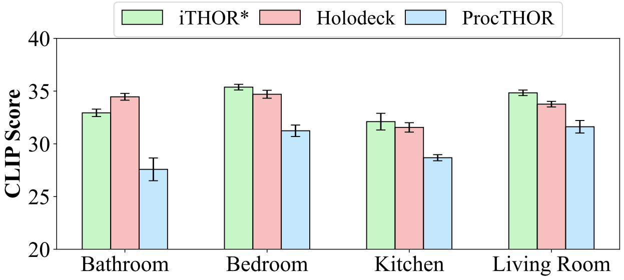


Figure 10. CLIP Score comparison over four residential scene types. * denotes iTHOR scenes are designed by human experts.

**Setup.** We prepared 120 scenes for human evaluation, comprising 30 scenes per scene type, for both HOLODECK and the PROCTHOR baseline. The PROCTHOR baseline has access to the same set of Objaverse assets as HOLODECK. For HOLODECK, we take the scene type, e.g., "bedroom", as the prompt to generate the scenes. We pair scenes of the same scene type from the two systems, resulting in 120 paired scenes for human evaluation. For each paired scene, we display two shuffled top-down view images of the scenes from the two systems. We ask the annotator to choose which scene is better or equally good based on three questions: (1) **Asset Selection**: which selection of 3D assets is more accurate/faithful to the scene type? (2) **Layout Coherence**: which arrangement of 3D assets adheres better to realism and common sense (considering the position and orientation)? and (3) **Overall Preference**: which of the two scenes would you prefer given the scene type?

**Humans prefer HOLODECK over PROCTHOR.** Figure 9 presents a clear preference for HOLODECK in the comparative human evaluation against PROCTHOR, with a majority of annotators favoring HOLODECK for Asset Selection (59.8%), Layout Coherence (56.9%), and showing a significant preference in Overall Preference (64.4%).

In addition to human judgments, we employ CLIP Score[6] [17] to quantify the visual coherence between the top-down view of the scene and its corresponding scene type embedded in a prompt template "*a top-down view of [scene type]*". Besides, we add human-designed scenes from iTHOR [25] as the upper bound for reference. Figure 10 shows the CLIP scores of HOLODECK exceed PROCTHOR with great margins and closely approach the performance of iTHOR, demonstrating HOLODECK's ability to generate visually coherent scenes faithful to the designated scene types. The CLIP Score experiment agrees with our human evaluation.

solver in our experiments, we analyze the MILP solver in the supplements.

[5] GPT-4-Vision can take in multiple images, we prompt it with multi-view screenshots of 3D assets to get the annotations.

[6] Here, we use OpenCLIP [22] with ViT-L/14 trained on LAION-2B [44]. We use cosine similarity times 100 as the CLIP Score.

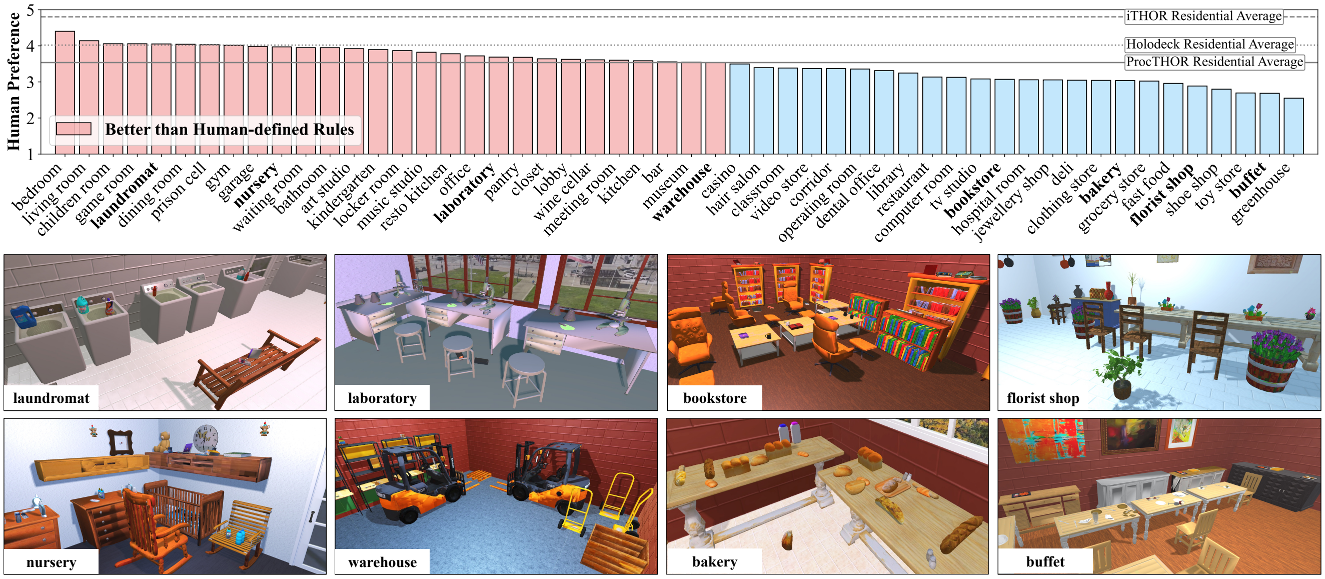


Figure 11. Human evaluation on 52 scene types from MIT Scenes [39] with qualitative examples. The three horizontal lines represent the average score of iTHOR, HOLODECK, and PROCTHOR on four types of residential scenes (*bedroom*, *living room*, *bathroom* and *kitchen*.)

## 4.2. HOLODECK on Diverse Scenes

To evaluate HOLODECK's capability beyond residential scenes, we have humans rate its performance on 52 scene types[7] from MIT Scenes Dataset [39], covering five categories: Stores (*deli*, *bakery*), Home (*bedroom*, *dining room*), Public Spaces (*museum*, *locker room*), Leisure (*gym*, *casino*) and Working Space (*office*, *meeting room*).

**Setup.** We prompt HOLODECK to produce five outputs for each type using only the scene name as the input, accumulating 260 examples across the 52 scene types. Annotators are presented with a top-down view image and a 360-degree video for each scene and asked to rate them from 1 to 5 (with higher scores indicating better quality), considering asset selection, layout coherence, and overall match with the scene type. To provide context for these scores, we include residential scenes from PROCTHOR and iTHOR in this study, with 20 scenes from each system.

**HOLODECK can generate satisfactory outputs for most scene types.** Figure 11 demonstrates the human preference scores for diverse scenes with qualitative examples. Compared to PROCTHOR's performance in residential scenes, HOLODECK achieves higher human preference scores over half of (28 out of 52) the diverse scenes. Given that PROCTHOR relies on human-defined rules and residential scenes are relatively easy to build with common objects and simple layout, HOLODECK's breadth of competence highlights its robustness and flexibility in generating various indoor environments. However, we notice that HOLODECK struggles with scenes requiring more complex layouts such as *restaurant* or unique assets unavailable in Objaverse, e.g., "a dental x-ray machine" for the scene *dental office*. Future work can improve the system by incorporating more assets and introducing more sophisticated layout algorithms.

| Method | Bathroom | Bedroom | Kitchen | Living Room | Average |
|---|---|---|---|---|---|
| ABSOLUTE | 0.369 | 0.343 | 0.407 | 0.336 | 0.364 |
| RANDOM | 0.422 | 0.339 | 0.367 | 0.348 | 0.369 |
| EDGE | 0.596 | 0.657 | **0.655** | 0.672 | 0.645 |
| CONSTRAINT | **0.696** | **0.745** | 0.654 | **0.728** | **0.706** |

Table 1. Mean Reciprocal Rank (↑) of different layouts ranked by human. CONSTRAINT: using spatial relational constraints; ABSOLUTE: LLM-defined absolute positions; RANDOM: randomly place the objects and EDGE: put objects at the edge of the room.

## 4.3. Ablation Study on Layout Design

This user study aims to validate the effectiveness of HOLODECK's constraint-based layout design method.

**Baselines.** We consider four layout design methods: (1) CONSTRAINT: the layout design method of HOLODECK; (2) ABSOLUTE: directly obtaining the absolute coordinates and orientation of each object from LLM akin to LayoutGPT [10]; (3) RANDOM: randomly place all objects in the room without collision; (4) EDGE: placed objects along the walls.

**Setup.** We modify the residential scenes of HOLODECK used in 4.1 by altering the layouts using the previously mentioned methods while keeping the objects in the scene identical. We present humans with four shuffled top-down images from each layout strategy and ask them to rank the four layouts considering out-of-boundary, object collision, reachable space, and layout realism.

**Constraint-based layout is more reliable.** Table 1 reports the Mean Reciprocal Rank of different layout design meth-

[7] Limited by the PROCTHOR framework, we filter those scenes types that require special structures such as *swimming pool*, *subway*, etc.

| | Office | | Daycare | | Music Room | | Gym | | Arcade | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Success | SPL | Success | SPL | Success | SPL | Success | SPL | Success | SPL | Success | SPL |
| Random | 3.90 | 0.039 | 4.05 | 0.041 | 5.20 | 0.052 | 2.84 | 0.029 | 2.54 | 0.025 | 3.71 | 0.037 |
| ProcTHOR [7] | 8.77 | 0.031 | 2.87 | 0.011 | 6.17 | 0.027 | 0.68 | 0.002 | 2.06 | 0.005 | 4.11 | 0.015 |
| +Objaverse (ours) | 18.42 | 0.068 | 8.99 | 0.061 | 25.69 | 0.157 | **18.79** | 0.101 | **13.21** | **0.076** | 17.02 | 0.093 |
| +Holodeck (ours) | **25.05** | **0.127** | **15.61** | **0.127** | **31.08** | **0.202** | 18.40 | **0.110** | 11.84 | 0.069 | **20.40** | **0.127** |

Table 2. Zero-shot ObjectNav on NoveltyTHOR. ProcTHOR is the model pretrained on ProcTHOR-10K [7]. +Objaverse and +Holodeck stand for models finetuned on the corresponding scenes. We report Success (%) and Success weighted by Path Length (SPL).

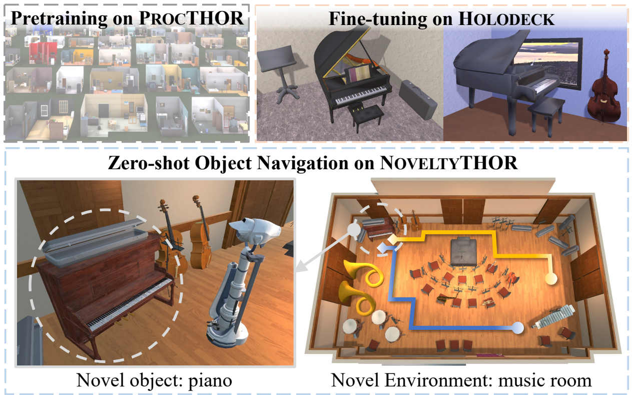

Pretraining on ProcTHOR
Fine-tuning on Holodeck
Zero-shot Object Navigation on NoveltyTHOR
Novel object: piano
Novel Environment: music room

Figure 12. Zero-shot object navigation in novel scenes. Given a novel scene type, e.g., *Music Room*, Holodeck can synthesize new scenes for fine-tuning to improve the performance of pretrained agents in expert-designed environments.

ods. Holodeck's constraint-based approach outperforms the other methods significantly on *bathroom*, *bedroom* and *living room*. Constraint and Edge perform similarly on *kitchen*, where it is common to align most objects against walls. The Absolute method performs no better than Random due to its tendency to create scenes with collision and boundary errors (see examples in the supplement), typically rated poorly by humans. These results endorse spatial relational constraints as a viable strategy for generating scenes that adhere to commonsense logic.

## 5. Object Navigation in Novel Environments

As illustrated in Figure 12, one application of Holodeck is synthesizing training environments to better match a novel testing distribution. To study this application, we consider ObjectNav [1], a common task in which a robot must navigate toward a specific object category. As existing benchmarks [6, 7, 41] for ObjectNav consider only household environments and support a very limited collection of object types (16 object types in total combining the above benchmarks), we introduce NoveltyTHOR, an artist-designed benchmark to evaluate embodied agents in diverse environments. Subsequently, we use the ObjectNav model pretrained on ProcTHOR-10K [25] and finetune it on 100 scenes generated by Holodeck. These scenes are created by prompting Holodeck with the novel scene type as input. The model is then evaluated on NoveltyTHOR.

**NoveltyTHOR.** We have two professional digital artists manually create 10 novel testing environments with two examples for each of the five categories: *Office*, *Daycare*, *Music Room*, *Gym*, and *Arcade*. Each scene contains novel object types not included in the existing ObjectNav tasks, e.g., "piano" in *Music Room*, "treadmill" in *Gym*, etc. Across NoveltyTHOR, there are 92 unique object types.

**Baselines.** For all methods except the one of random action, we use the same pre-trained ObjectNav model from ProcTHOR-10K [25], which has been trained for ≈400M steps to navigate to 16 object categories. To adapt the agent to novel scenes without human-construct training data, we consider two methods: (1) +Holodeck: we prompt[8] Holodeck to automatically create 100 scenes for each scene type; (2) +Objaverse: a strong baseline by enhancing ProcTHOR with Holodeck's scene-type-specific object selection, specifically, those scenes are populated with similar Objaverse assets chosen by Holodeck.

**Model.** Our ObjectNav models use the CLIP-based architectures of [24], which contains a CNN visual encoder and a GRU to capture temporal information. We train each model with 100 scenes for 50M steps, which takes approximately one day on 8 Quadro RTX 8000 GPUs. We select the checkpoint of each model based on the best validation performance on its own validation scenes.

**Results.** Table 2 shows zero-shot performance on NoveltyTHOR. Holodeck achieves the best performance on average and surpasses baselines with considerable margins on *Office*, *Daycare*, and *Music Room*. On *Gym* and *Arcade*, +Holodeck and +Objaverse perform similarly. Given that the main difference between +Holodeck and +Objaverse scenes is in the object placements, the observed difference suggests that Holodeck is more adept at creating layouts that resemble those designed by humans. For example, We can observe in Figure 12 that the music room in NoveltyTHOR contains a piano, violin cases, and cellos that are in close proximity to each other. The music room generated by Holodeck also shows a similar arrangement of these objects, highlighting the "commonsense" understanding of our method. ProcTHOR struggles in NoveltyTHOR, often indistinguishably from random, because of poor object coverage during training.

[8] Here, we prompt with the scene name and its paraphrases to get more diverse outputs, e.g., we use "game room", "amusement center" for *Arcade*.

## 6. Conclusion and Limitation

We propose HOLODECK, a system guided by large language models to generate diverse and interactive Embodied AI environments with text descriptions. We assess the quality of HOLODECK with large-scale human evaluation and validate its utility in Embodied AI through object navigation in novel scenes. We plan to add more 3D assets to HOLODECK and explore its broader applications in Embodied AI in the future.

## References

[1] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects. In *arXiv:2006.13171*, 2020. 8

[2] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971. 13

[3] Angel Chang, Manolis Savva, and Christopher D Manning. Interactive learning of spatial knowledge for text to 3d scene generation. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 14–21, 2014. 2

[4] Angel X Chang, Mihail Eric, Manolis Savva, and Christopher D Manning. Sceneseer: 3d scene design with natural language. *arXiv preprint arXiv:1703.00050*, 2017. 2

[5] Yu Cheng, Yan Shi, Zhiyong Sun, Dezhi Feng, and Lixin Dong. An interactive scene generation using natural language. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6957–6963. IEEE, 2019. 2

[6] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. Robothor: An open simulation-to-real embodied ai platform. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3164–3174, 2020. 2, 8

[7] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In *NeurIPS*, 2022. Outstanding Paper Award. 2, 3, 6, 8, 12

[8] Matt Deitke, Rose Hendrix, Ali Farhadi, Kiana Ehsani, and Aniruddha Kembhavi. Phone2proc: Bringing robust robots into our chaotic world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9665–9675, 2023. 2

[9] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023. 2, 3

[10] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *arXiv preprint arXiv:2305.15393*, 2023. 3, 5, 7

[11] Rafail Fridman, Amit Abecasis, Yoni Kasten, and Tali Dekel. Scenescape: Text-driven consistent scene generation. *arXiv preprint arXiv:2302.01133*, 2023. 2, 3

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

[13] Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020. 2

[14] Jiatao Gu, Alex Trevithick, Kai-En Lin, Joshua M Susskind, Christian Theobalt, Lingjie Liu, and Ravi Ramamoorthi. Nerfdiff: Single-image view synthesis with nerf-guided distillation from 3d-aware diffusion. In *International Conference on Machine Learning*, pages 11808–11826. PMLR, 2023. 3

[15] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. 13

[16] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9984–9993, 2019. 3

[17] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021. 6

[18] Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. Text2room: Extracting textured 3d meshes from 2d text-to-image models. *arXiv preprint arXiv:2303.11989*, 2023. 2, 3

[19] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)*, 39(4):118–1, 2020. 2

[20] Ian Huang, Vrishab Krishna, Omoruyi Atekha, and Leonidas Guibas. Aladdin: Zero-shot hallucination of stylized 3d assets from abstract scene descriptions. *arXiv preprint arXiv:2306.06212*, 2023. 3

[21] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. 12

[22] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, 2021. If you use this software, please cite it as below. 6, 12

[23] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, S. Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5:6670–6677, 2019. 2

[24] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings

for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 8

[25] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 2, 3, 6, 8

[26] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023. 2

[27] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023. 3

[28] Yiqi Lin, Hao Wu, Ruichen Wang, Haonan Lu, Xiaodong Lin, Hui Xiong, and Lin Wang. Towards language-guided interactive 3d generation: Llms as layout interpreter with generative feedback. *arXiv preprint arXiv:2305.15808*, 2023. 3

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

[30] Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. Latent-nerf for shape-guided generation of 3d shapes and textures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12663–12673, 2023. 3

[31] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 3

[32] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. 14

[33] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7588–7597, 2019. 3

[34] OpenAI. GPT-4V(ision) System Card, 2023. 6

[35] R OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023. 2

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

[37] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 3

[38] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018. 2

[39] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE conference on computer vision and pattern recognition*, pages 413–420. IEEE, 2009. 7, 14, 17

[40] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 3, 5, 12

[41] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. 2, 8

[42] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. 5, 12

[43] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019. 2

[44] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. 6, 12

[45] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5466–5475, 2023. 2

[46] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34:251–266, 2021. 2

[47] Fuwen Tan, Song Feng, and Vicente Ordonez. Text2scene: Generating compositional scenes from textual descriptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6710–6719, 2019. 2

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

[49] Xinpeng Wang, Chandan Yeshwanth, and Matthias Nießner. Sceneformer: Indoor scene generation with transformers. In *2021 International Conference on 3D Vision (3DV)*, pages 106–115. IEEE, 2021. 2

[50] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023. 2

[51] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022. 2

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

[53] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29, 2016. 3

[54] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9068–9079, 2018. 2

[55] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019. 3

[56] Jingbo Zhang, Xiaoyu Li, Ziyu Wan, Can Wang, and Jing Liao. Text2nerf: Text-driven 3d scene generation with neural radiance fields. *arXiv preprint arXiv:2305.11588*, 2023. 2, 3

[57] Yiqun Zhao, Zibo Zhao, Jing Li, Sixun Dong, and Shenghua Gao. Roomdesigner: Encoding anchor-latents for style-consistent and shape-compatible indoor scene generation. *arXiv preprint arXiv:2310.10027*, 2023. 2

[58] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021. 3

# 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] [40], 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:

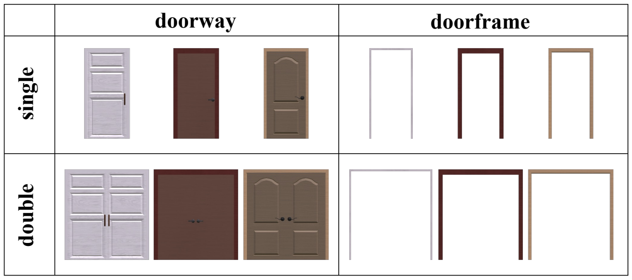


Figure 13. Examples of different doors in Holodeck.

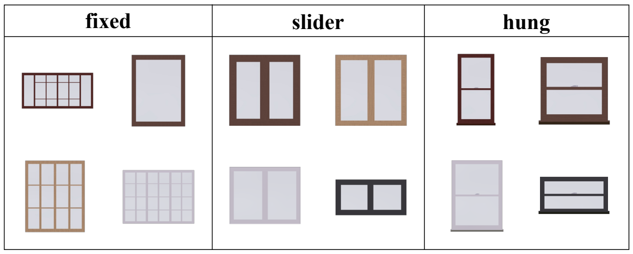


Figure 14. Examples of different windows in Holodeck.

- **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) [42] 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-

[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 [44], for all CLIP-related components in this paper.

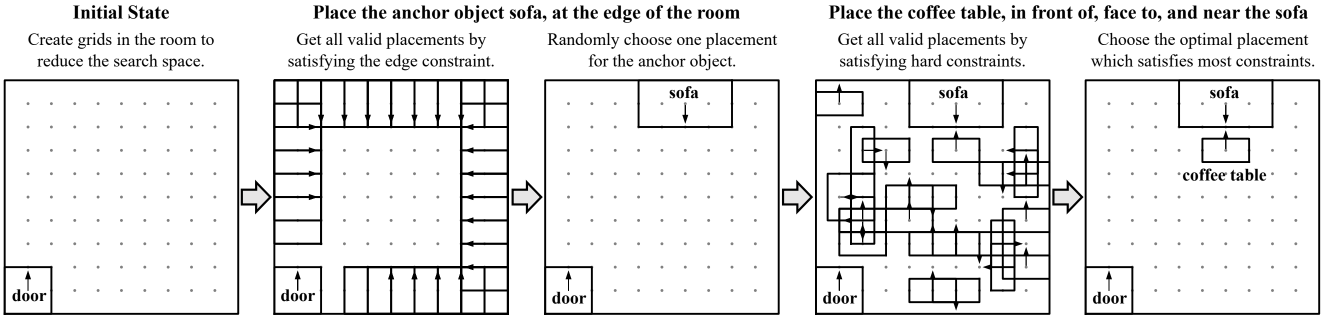

Initial State
Create grids in the room to reduce the search space.
Place the anchor object sofa, at the edge of the room
Get all valid placements by satisfying the edge constraint.
Randomly choose one placement for the anchor object.
Place the coffee table, in front of, face to, and near the sofa
Get 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.

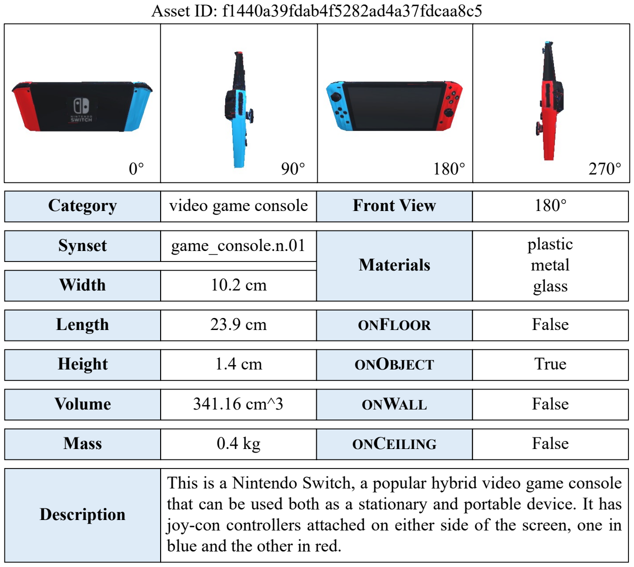
In our MILP formulation, each object's position is determined by four variables: $(x, y, \text{rotate}_{90}, \text{rotate}_{180})$. The variables $\text{rotate}_{90}$ and $\text{rotate}_{180}$ are boolean, indicating rotations of 90 and 180 degrees, respectively. For example, if $\text{rotate}_{90}$ and $\text{rotate}_{180}$ 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 21.

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

- **Above (Floor Object)**: This denotes the floor object di-

[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
0°
90°
180°
270°

| Category | video game console | Front View | 180° |
|---|---|---|---|
| Synset | game_console.n.01 | Materials | plastic<br>metal<br>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.

rectly 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 [32] 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.

### 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.

## B. Prompt

The complete prompt templates of Holodeck's modules are provided in Figure 17 and 18. The prompt for annotating 3D assets using GPT-4-V is shown in Figure 19.

## C. Qualitative Examples

In Figure 20, we showcase an additional 20 scenes generated by Holodeck. These 20 scene types are chosen from the MIT dataset [39], distinct from examples in the main paper. Figure 21 presents a comparative analysis of layouts created by five methods. Figure 22 offers a visual comparison of residential scenes from iTHOR, ProcTHOR, and Holodeck, highlighting the differences and capabilities of each system.

## D. 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 beyond those found in Objaverse. A user-friendly interface was designed 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 23 and 24 showcase top-down views of the 10 NoveltyTHOR scenes, spanning five categories.

[12] AI2-THOR RandomSpawn Documentation

**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 17. 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 18. 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 19. Prompt template for annotating 3D assets with GPT-4-V.

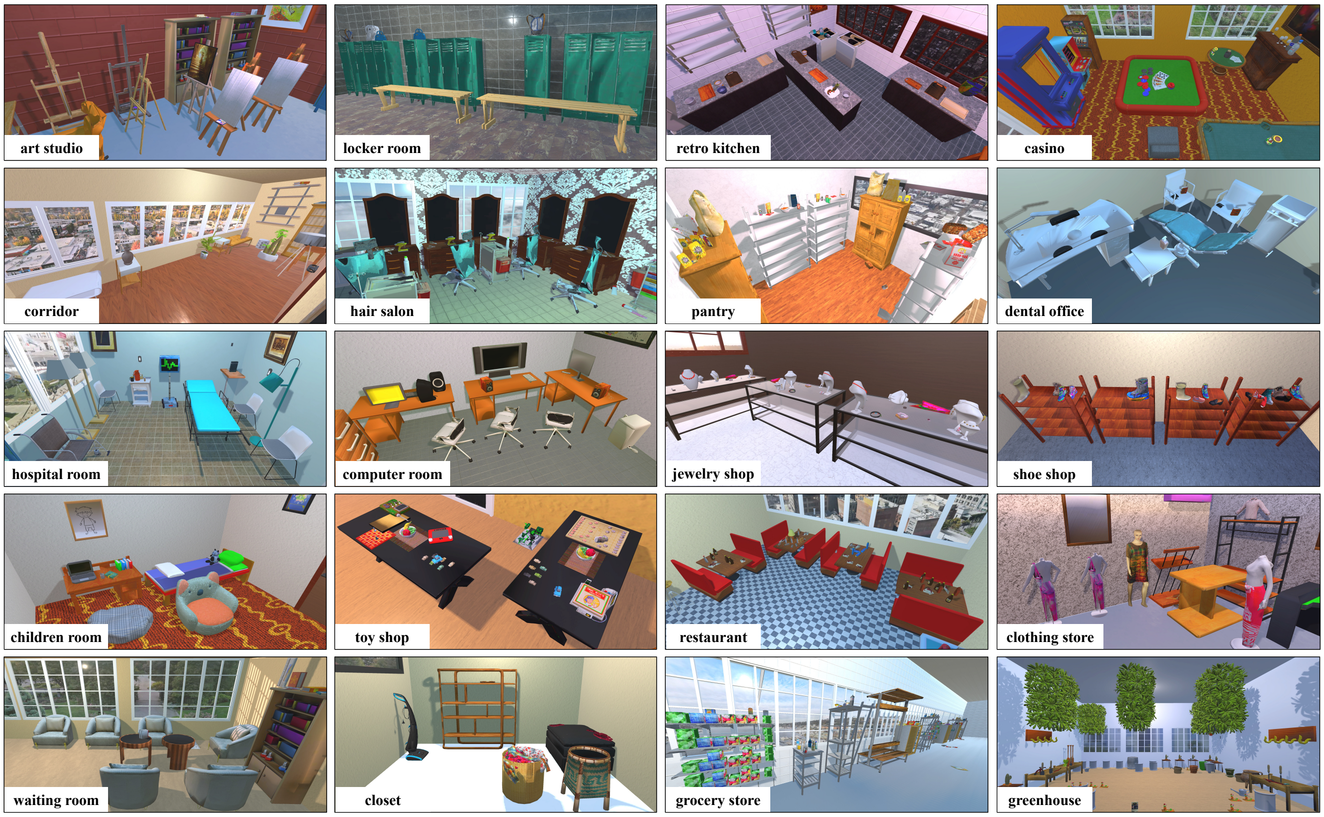


Figure 20. Additional examples of different scene types from MIT Scenes [39].

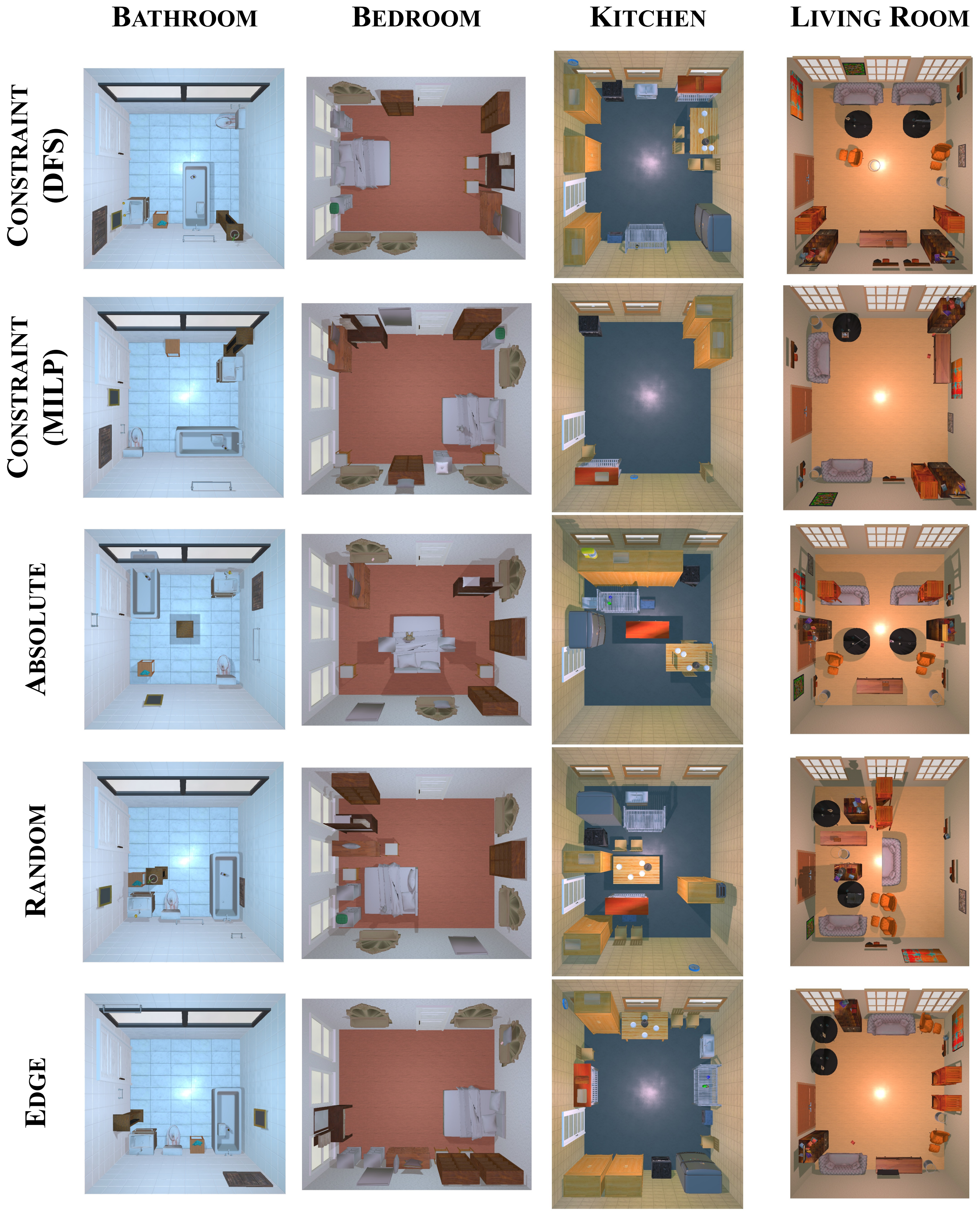


Figure 21. Qualitative comparison of the different layout methods.

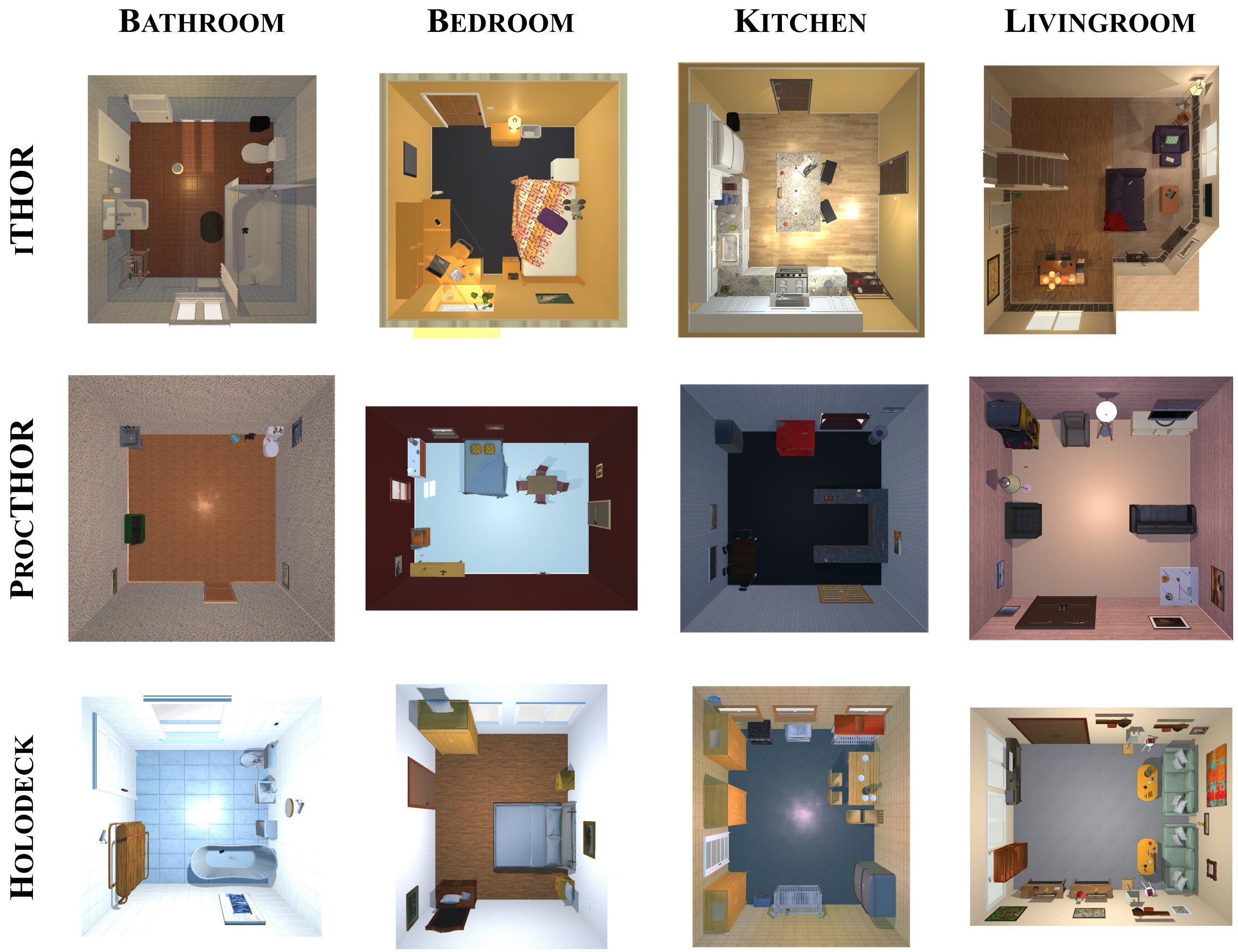


Figure 22. Qualitative examples on four types of residential scenes from iTHOR, ProcTHOR, and Holodeck.

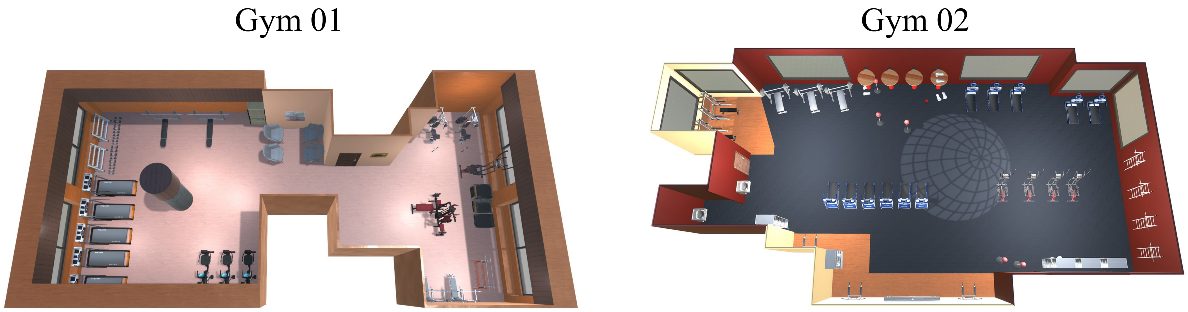
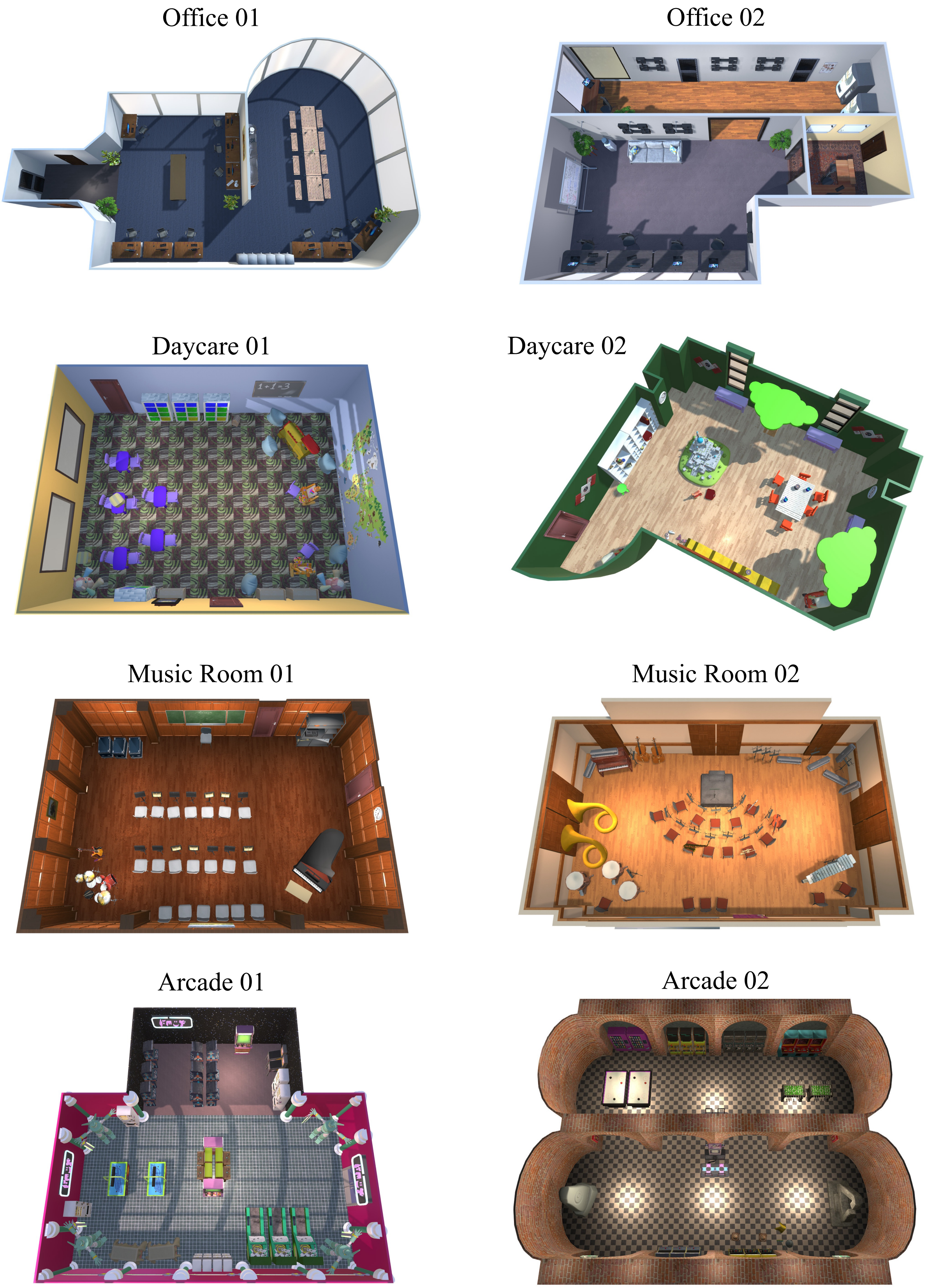


Figure 23. Top-down view of NoveltyTHOR. Each scene type has two instances.



Office 01

Office 02

Daycare 01

Daycare 02

Music Room 01

Music Room 02

Arcade 01

Arcade 02

Figure 24. Top-down view of NOVELTYTHOR (continued).