LAYOUTVLM: Differentiable Optimization of 3D Layout via Vision-Language Models

Supplementary Material

A. Details of LAYOUTVLM

This section elaborates on the details of our method, including the prompts we used.

A.1. Grouping

To address the challenge of handling large numbers of 3D assets, we cluster related assets into groups using the following prompt:

```
You are an experienced 3D layout designer. You are
     teaching a junior designer the concept of
     semantic asset group. Understanding and
     recognizing semantic asset groups will help
     the designers to design a room layout more
     efficiently by decomposing the room layout
     into smaller, more manageable parts.

**Definition:**
A semantic asset group is a collection of assets
     that are logically related to each other. The
     grouping of the assets are based on functional
     , semantic, geometric, and functional
     relationships between assets.
Usually assets that are close to each other in
     space can be grouped together. For example, a
     bed, a nightstand, and a lamp on top of the
     nightstand can be grouped together.
However, it is also possible to group assets that
     are not physically close to each other but are
      semantically related. For example, a sofa, a
     tv console in front of the sofa, and a tv on
     top of the tv console can be grouped together
     even though the tv and the tv console is a few
      meters away from the sofa. They can be
     grouped together because they are semantically
      related -- the tv is in front of the sofa.

**Task:**
Now, given a 3D scene, you will use it as an
     example to teach the junior designer how to
     group assets into semantic asset groups.

**Step-by-Step Guide:**
1. You will first be provided a list of assets.
     Based on the assets, you should describe the
     general layout of the scene, the types of
     assets present, and any notable features.
2. You will identify the semantic relationships
     between the assets. You should consider the
     functional, semantic, and geometric
     relationships between the assets.
3. You will then describe how you would group the
     assets into semantic asset groups. You should
     explain the rationale behind each group and
     how the assets within each group are related
     to each other.
4. You will then order the semantic asset groups
     based on the sequence in which they should be
     placed in the scene. You should consider the
     significance of each group and the logical
     flow of the scene layout. For example, larger
     or more prominent assets may be placed first
     to establish the scenes focal points.
5. Finally, you will format the grouping
```

```
     information into a clear and organized
     structure that can be easily understood by
     other designers or stakeholders.

**Example:**
Suppose you are examining a bedroom scene. In the
     bedroom, there are the following assets:
bed | ...
nightstand | ...
lamp | ...
bed_bench | ...
dresser-0 | ...
dresser-1 | ...
photo_frame-0 | ...
dressing_table-0 | ...
chair-0 | ...

1. After examining the scene, you will describe the
      scene a bedroom with a bed and a seating area
      for dressing.
2. You will list the assets and their relationships
     :
- the bed is the central piece
- the nightstand is next to the bed for placing
     items. The bedside table should be close to
     the bed for easy access.
- the lamp is on the nightstand for lighting. The
     lamp should be close to the bed for reading.
- the end of bed bench is at the foot of the bed.
     The bench is at the end of the bed for seating
      or placing items.
- the dresser is on the other side of the room. The
      dresser is on the opposite side of the bed
     for storage.
- the photo frame is on the dresser. The photo is
     directly opposite the bed for viewing.
- the dressing table is in an open area of the room
     .
- the chair is in front of the dressing table for
     seating.
3. You will group the assets into semantic asset
     groups:
- Group 1: Bed, Nightstand, Lamp. The rational is
     that the bed is the central piece, the
     nightstand is next to the bed, and the lamp is
      on the nightstand. They are related to each
     other because they are used for sleeping and
     reading.
- Group 2: End of bed bench. The bench is at the
     foot of the bed for seating or placing items.
- Group 3: Dresser, Photo frame. The dresser is on
     the opposite side of the bed for storage, and
     the photo frame is directly opposite the bed
     for viewing.
- Group 4: Dressing table, Chair. The dressing
     table is in an open area of the room, and the
     chair is in front of the dressing table for
     seating.
4. You will order the semantic asset groups based
     on the sequence in which they should be placed
      in the scene:
- Group 1: Bed, Nightstand, Lamp. They should be
     placed first to establish the sleeping area.
     They are the focal point of the room.
- Group 2: End of bed bench. It should be placed
     next to the bed to complement the sleeping
     area.
- Group 3: Dresser, Photo frame. They should be
     placed on the opposite side of the room to
```

````
     balance the layout.
- Group 4: Dressing table, Chair. They should be
     placed in an open area of the room to create a
      dressing area.
5. You will format the grouping information into a
     clear and organized structure:
```json
{
    "list": [
        {"id": 1,
         "name": "sleeping area",
         "assets": ["bed", "nightstand", "lamp"],
         "rational": "they are used for sleeping
              and reading.",
         "key_relations_between_assets": ["the bed
              is the central piece", "the
              nightstand is next to the bed", "the
              lamp is on the nightstand"],
         "key_relations_with_other_groups": []
        },
        {"id": 2,
         "name": "seating area",
         "assets": ["bed_bench"],
         "rational": "this end of bed bench is at
              the foot of the bed for seating or
              placing items.",
         "key_relations_between_assets": [],
         "key_relations_with_other_groups": ["the
              bench complements the sleeping area."
              ]
        },
        {"id": 3,
         "name": "storage area",
         "assets": ["dresser-0", "dresser-1", "
              photo_frame-0"],
         "rational": "the dresser is on the
              opposite side of the bed for storage,
               and the photo frame is directly
              opposite the bed for viewing.",
         "key_relations_between_assets": ["the
              photo frame is on top of the dresser"
              ],
         "key_relations_with_other_groups": ["the
              dresser is for storage, and the photo
               frame is for viewing. To make the
              photo frame visible from the bed, the
               dresser should be placed on the
              opposite side of the bed."]
        },
        {"id": 4,
         "name": "dressing area",
         "assets": ["dressing_table-0", "chair-0"],
         "rational": "the dressing table is in an
              open area of the room, and the chair
              is in front of the dressing table for
               seating.",
         "key_relations_between_assets": ["the
              chair is in front of the dressing
              table"],
         "key_relations_with_other_groups": ["the
              dressing area complements the
              sleeping area and the storage area by
               providing another function in the
              room."]
        }
    ]
}

```

Now, please proceed by grouping and organizing the
     following list of assets according to the
     layout instruction:
NOTE: it is very important to include all the
     assets!!! And please do not change the name of
      the assets.
````

```
Task: TASK_DESCRIPTION
Instruction: LAYOUT_CRITERIA
In the room, there are the following assets:
ASSET_LISTS
```

A.2. Differentiable Spatial constraint

We define differentiable objectives for the spatial constraints used in our method. We introduce the necessary notations and provide the mathematical formulations below. The pose of an asset m_i is represented as $p_i = (x_i, y_i, z_i, \theta_i)$, the orientation of the asset θ_i is represented with an orientation vector $\mathbf{v}_i = (\cos\theta_i, \sin\theta_i)$, and b_i denotes the 3D bounding box size of the asset. Below are the mathematical objectives for various spatial relations.

Distance Objective

$$\mathcal{L}_{\text{distance}}(p_i, p_j, d_{\text{min}}, d_{\text{max}}) = \text{clamp}\big(\min(\|p_i - p_j\| - d_{\text{min}}, d_{\text{max}} - \|pos_i - pos_j\|), 0, 1\big) \tag{3}$$

where $\|pos_i - p_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ is the Euclidean distance between i and j in the x-y plane. The function clamp(x, a, b) constrains x to $[a, b]$, defined as clamp$(x, a, b) = \min(\max(x, a), b)$.

On-Top-Of Objective

$$\mathcal{L}_{\text{on\_top\_of}}(p_i, p_j, b_i, b_j) = -\mathcal{L}_{\text{DIoU}}(p_i, p_j, b_i, b_j). \tag{4}$$

where IoU(a, b) denotes the Intersection-over-Union of the bounding boxes of a and b. Instead of using a loss function for the z-axis, the On-Top-Of objective directly sets z_i the z-coordinate of the object i to be on top of the object j.

Point-Towards Objective

$$\mathcal{L}_{\text{point\_towards}}(p_i, p_j, \phi) = \begin{cases} 0, & \text{if } \mathbf{v}_i \cdot \mathbf{d}_{ij} > 0, \\ 1 - \frac{\mathbf{v}_i \cdot \mathbf{d}_{ij}}{\|\mathbf{v}_i\|\|\mathbf{d}_{ij}\|}, & \text{otherwise,} \end{cases} \tag{5}$$

where $\mathbf{d}_{ij}$ is the direction vector from i to j rotated by ϕ degree around the z-axis.

Align-With Objective

$$\mathcal{L}_{\text{align\_with}}(p_i, p_j, \phi) = 1 - \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\|\|\mathbf{v}_j\|}, \tag{6}$$

where $\mathbf{v}_i$ and $\mathbf{v}_j$ are the orientation vectors of i and j, respectively.

Against-Wall Objective The "Against-Wall" objective consists of two components: (a) the sum of distances from the object's corners to the wall, and (b) a term that encourages the object to point away from the wall. Let $c_i^{(k)}$ be the k-th corner of the i-th object, which is calculated based on the object's x-y position (x_i, y_i), rotation, and bounding box size b_i. The loss function is:

$$\mathcal{L}_{\text{against\_wall}}(p_i, w_j, b_i) = \sum_{k=1}^{4} \text{clamp}\big(\|c_i^{(k)} - w_j\|, 0, 1\big) + \left(1 - \frac{\mathbf{v}_i \cdot \mathbf{n}_{w_j}}{\|\mathbf{v}_i\|\|\mathbf{n}_{w_j}\|}\right), \qquad (7)$$

where $\|c_i^{(k)} - w_j\|$ denotes the Euclidean distance between the k-th corner of to the wall segment on the x-y plane and $\mathbf{n}_w$ denotes the normal vector of wall w (i.e., perpendicular to the wall).

Optimization Details We use Adam optimizer and Exponential LR scheduler with a decay factor 0.96. Each optimization runs for 400 steps with projection back to the boundary every 100 iterations. Optimizing a scene with 40 assets takes 1–5 minutes on a single GPU, 5-10 GPT-4o calls (i.e., one per group), varying based on the VLM-defined optimization problem.

Below is the prompt we feed VLM to generate spatial constraints.

```
You are an experienced layout designer that place 3
     D assets into a scene. Specify the asset
     placement using a Python-based DSL.

**3D Convention:**
- Right-handed coordinate system.
- The X-Y plane is the floor; the Z axis points up.
      The origin is at a corner, defining the
     global frame.
- Asset front faces point along the positive X axis
     . The Z axis points up. The local origin is
     centered in X-Y and at the bottom in Z. A [90]
      rotation means that the object will face the
     positive Y axis. The bounding box aligns with
     the assets local frame.

**DSL:**
```python
from pydantic import BaseModel, Field
from typing import List, Optional


class Wall(BaseModel):
    corner1: List[float] = Field(description="XY
         coordinates of first corner")
    corner2: List[float] = Field(description="XY
         coordinates of second corner")

class AssetInstance(BaseModel):
    position: Optional[List[float]] = Field(
         description="XYZ position", default=[0, 0,
          0]])
    rotation: List[float] = Field(description="
         counterclockwise rotation in degrees
         around Z-axis.", default=[0])

class Assets(BaseModel):
    description: str = Field(description="Asset
         description")
    placements: List[AssetInstance] = Field(
         description="Instances of the 3D asset
         that will share the same shape and
         dimension.")
    size: Optional[List[float]] = Field(description
         ="BBox size. Z-axis up. Assets front faces
          point along the positive X axis.")


class ConstraintSolver:
    def __init__(self):
        self.constraints = []

    def on_top_of(self, asset1: AssetInstance,
         asset2: AssetInstance):
             pass

    def against_wall(self, asset1: AssetInstance,
         wall: Wall):

        pass

    def distance_constraint(self, asset1:
         AssetInstance, asset2: AssetInstance,
         min_distance, max_distance, weight=1):
              pass

    def align_with(self, asset1: AssetInstance,
         asset2: AssetInstance, angle=0.):
              pass

    def point_towards(self, asset1: AssetInstance,
         asset2: AssetInstance, angle=0.):
            pass

solver = ConstraintSolver()
```

Constraints:
- Z-axis: on_top_of
- Planar: distance_constraint
- Orientation: align_with, point_towards
- Planar & Orientation: against_wall
- For constraints with multiple arguments, the
     order determines the constraint direction. For
      example, to update both assets placements
     with align_with, specify the constraint twice,
      swapping the arguments. For example, solver.
     point_towards(chair[0], sofa[0]) makes the
     chair point to the sofa, while solver.
     point_towards(sofa[0], chair[0]) adjusts the
     sofa to point to the chair.

Task:
You will receive:
1. High-level design goals.
2. A list of existing scene assets (if any).
3. A list of new assets with their dimensions and
     orientations.
4. A top-down view of the current scene, with a
     marked global frame, 1-meter grid, labeled
     assets, and front-facing orientation arrows.
     Walls are also labeled with orientation arrows
     .
5. A side view of the current scene, with the
     global frame and 1-meter grid.
6. A top-down view of each new asset in an empty
     scene, facing the positive X-axis, labeled
     with its name and front-facing arrow.

Your task is to write a program that:
1. Specifies precise position and rotation for the
     new asset placements.
```

```
2. Constraints for the asset placements. These
     constraints will ensure that the layout
     semantics are maintained when the layout is
     being adjusted to be physically feasible.

**Instructions:**
Follow these instructions carefully:
- Specify the constraints for all the assets to be
     placed, specifically for each asset instance
     in the placements list of the Assets class.
- Specify at least one planar constraint and one
     orientation constraint for each asset.
- Do not specify constraints for existing assets or
      walls.
- Do not re-initialize existing assets or walls.
- Do not hallucinate assets.
- Enclose your answer in the '''python ''' code
     block. PLEASE DO NOT REPEAT THE GIVEN PROGRAM.
- Use code comments to explain your reasoning.
- Do not overwrite asset variable names (e.g.,
     avoid for chair in sofa.placements: ...).
- It is important to find a empty space to place
     the new sets of assets given the
```

A.3. Self-Consistent Decoding

We propose self-consistent decoding to address the challenge of maintaining layout coherence in VLM-generated spatial plans. Our main hypothesis is that preserving self-consistent spatial relations—those that align with the estimated numerical poses of objects—is essential for ensuring semantic and physical plausibility during optimization. During implementation, we simplify the decoding process by enforcing that each asset maintains at most one orientational constraint, either to “point towards" or “align with" another asset. Additionally, the spatial relation "on top of" is excluded from the self-consistency decoding, as we empirically observe that “on top of " relations are almost accurately and reasonably predicted by our model; thus, enforcing self-consistency is unnecessary.

A.4. Annotating Unlabeled 3D Assets

We annotate the 3D assets used in a similar way as in Holodeck [5], using GPT-4o to determine the front face of the object and to determine the textual description of the asset. More specifically, GPT-4o 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.
- **Variable Name**: a string denoting the python variable name that will be used to refer to this object in our scene layout representation.
- **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.5. Finetuning VLMs with Scene Datasets

This scene representation can be automatically extracted from scene layout datasets without requiring manual annotations. Specifically, given a set of posed objects in a 3D scene, we apply the preprocessing procedure outlined in Section 3 to obtain both textual descriptions and oriented bounding boxes for each object. After canonicalizing the objects, we compute cost values for our defined spatial relations based on the ground-truth positions and orientations of the objects, using heuristic thresholds to determine whether each spatial relation is satisfied. The resulting scene representation includes both raw object poses and the satisfied spatial relations, which we then use to fine-tune VLMs to generate these scene representations from input objects and scene renderings. In our implementation, we use the 3D-Front dataset to extract training data for around 9000 rooms. Our approach is capable of identifying layout patterns in 3D scenes, such as a variable number of chairs around a table, nightstands positioned beside a bed, or an entertainment area comprising a TV, coffee table, and sofa. We investigate fine-tuning two VLMs for the layout generation task: the closed-source *GPT-4o* [29] and the open-source *LLaVA-NeXT-Interleave* [35].

B. Details of our Experiments

B.1. Generating Test Cases

We developed a pipeline for generating valid open-vocabulary 3D layout generation cases to benchmark our method against existing methods.

First, we feed the following prompt to GPT-4o to generate a layout instruction given the room type:

```
Given a task description, return a string
     description of layout criteria for an interior
      design focused on the provided task.
Include considerations for aesthetics,
     functionality, and spatial organization. Each
     layout criteria string should start with
the phrase "The layout criteria should follow the
     task description and be...".

For example, if the task description is a spatious
     study room, the layout criteria should be:
"The layout criteria should follow the task
     description and be spatious, tidy, and minimal
     "

task description: TASK_DESCRIPTION

Return only the layout criteria and nothing else.
     Ensure that the criteria is
no longer than 1-2 sentences. It is extremely
     important.
```

Condition on the generated room layout instruction, we then use the following prompt to retrieve a bunch of plausible assets:

```
Given a client's description of a room or tabletop
     and the floor vertices of the room, determine
     what objects and how many of
them should be placed in this room or table.

Objects should follow the requirements below.

Requirement 1: Be specifc  about how you describe
     the objects, while using as simple english as
     possible. Each object should try to be around
     two words, including
a relevant adjective if possible. Normally this
     adjective should be the room type, such as "
     kitchen scale" or "bathroom sink". However, it
      can also be a color or
a material, such as "wooden chair" or "red table"
     if necessary. Ensure that descriptions are
     simple - an elementary student should
     understand what each object is.

For example, if a client description asks for "
     weightlifting racks", simplify the description
      to "weightlifting equipment".
For example, if a client description asks for "a
     1980's jukebox", simplify the description to "
     vintage jukebox".
For example, if a client description includes "
     aerobic machines",simplify the description to
     "treadmill" and "exercise bike".
For example, if a client is describing a kitchen
     and is asking for a "scale" for food, ensure
     the object includes an adjective to describe
     the object, such as "kitchen scale".
For example, if a client is describing a bathroom
     and is asking for a "sink", ensure the object
     includes an adjective to describe the object,
     such as "bathroom sink".


Requirement 2: Only choose objects that are
     singular in nature.
For example, instead of choosing a "speaker system"
     , just choose "speaker".
For example, Instead of choosing "tables" and "
     chairs", just choose "table" and "chair".


Requirement 3: Ensure that the objects are relevant
      to the room or tabletop.
A client's description can either describe a room
     or tabletop arrangement. If it is describing a
      tabletop arrangement,
do not include objects like "table" or "chair" in
     the response. Only include objects that would
     be placed on the table.

If it is describing a room arrangement, do not
     describe include things like "windows" or "
     doors" in the response.
Only include objects that would be placed in the
     room. Other than paintings, posters, light
     fixtures, or shelfs,
do not include objects that would be placed on the
     wall.


Requirement 4: Ensure that rooms have a place to
     sit and a place to put things down, like a
     counter, table, or nightstand.
This also means that objects like art easels, work
     benches, or desks should have corresponding
     chairs or stools.
For example, if a client is describing a bar,
     ensure that the response includes a "bar table
     " or "counter" and "bar stools" as well.
For example, if a client describes a classroom,
     ensure that all desks have corresponding
     chairs.

Requirement 5: Try and include as many objects as
     possible that are relevant to the room or
     tabletop. Aim for at least 10
objects in each response, but ideally include more.


After ensuring these requirements, return a
     dictionary objects, where the key is the
     object name and the value is an array tuple of
      two values.
The first value of the key array is the number of
     times that object should occur in the room and
      the second value is how many
types of that object should exist.

For example, for a given description of a garden,
     you would want many plants, but do not want
     all of them to be the same type.
Thus, the value of the key array would be [9, 3]
     for the object "plant". This means that there
     should be 9 plants in the garden
and there should be 3 different types of ferns in
     the garden.

For example, for a given description of "A study
     room 5m x 5m"
Return the Dictionary: {"desk": [1, 1], "chair":
     [1, 1], "lamp": [1, 1], "bookcase": [2, 1], "
     laptop_computer": [1, 1], "computer monitor":
     [1, 1], "printer": [1, 1], "sofa": [1, 1], "
     flowerpot": [1, 1], "painting": [1, 1]}

For example, for a given description of "A tabletop
      arrangement with a bowl placed on a plate 1m
     x 1m"
Return the Dictionary: {"plate": [1, 1], "bowl":
     [1, 1], "fork": [1, 1], "knife": [1, 1], "
     spoon": [1, 1], "napkin": [1, 1], "salt shaker
     ": [1, 1], "pepper shaker": [1, 1], "wine
     glass": [1, 1], "water glass": [1, 1]}

For example, for a given description of "a vibrant
     game room filled with vintage arcade games and
      a jukebox, 6m x 6m"
Return the Dictionary: {"jukebox": [1, 1], "arcade
     machine": [3, 1], "pool table": [1, 1], "darts
      board": [1, 1], "bar stool": [4, 1], "bar
     table": [1, 1], "neon sign": [1, 1], "popcorn
     machine": [1, 1], "vending machine": [1, 1], "
     air hockey table": [1, 1]}

For example, for a given description of "a lush
     inside garden filled with a variety of plants
     and a small birdbath, 5m x 3m"
Return the Dictionary: {"fern": [8, 3], "birdbath":
      [1, 1], "flowerpot": [3, 1], "watering can":
     [1, 1], "garden gnome": [1, 1], "garden bench"
     : [1, 1], "garden shovel": [1, 1], "garden
     rake": [1, 1], "garden hose": [1, 1]}

task description: TASK_DESCRIPTION
layout criteria: LAYOUT_CRITERIA
room size in meters: ROOM_SIZE

Remember, you should only include objects that are
     most important to be placed in the room or on
     the table.
The dictionary should not include the room
     dimensions.

Return only the dictionary of objects and nothing
     else. It is extremely important.
```

Subsequently, we embed the generated asset descriptions using CLIP [36] and use the embeddings to retrieve 3D assets from Objaverse. The following prompt is employed to verify whether the retrieved object belongs to the given room:

```
You are an interior designer. A client is
     suggesting possible objects that he thinks
     belongs in a described
room. You are tasked with determining if the client
      is correct or not, stating whether the
     proposed object
belongs in the described room.

Given a client's description of a room or tabletop,
      the description of an object, and images of
     the object,
determine if the described object should be placed
     in the room that is described. To help, you
     are also
given a description of what object the client was
     initially looking for. Ensure that the style
     and color
of the object matches the type of the room. If an
     object is not in the style of what the room
     type
would typically have, it should not be placed in
     the room.

Return "True" if the object should be kept in the
     room and "False" if the object should not be.

For example, if the room description is a "A
     tabletop arrangement with a bowl placed on a
     plate 1m x 1m" and the object appears to be "a
      shovel":
Return: False

For example, if the room description is a "A
     spatious study room with a desk and chair" and
      the object appears to be "an 18th century
     book":
Return: True

For example, for a given description of "a vibrant
     game room filled with vintage arcade games and
      a jukebox, 6m x 6m" and the object appears to
      be "a 1980s pinball machine":
Return: True

For example, for a given description of an "art
     room with chairs", and the object appears to
     be a "a pink beach chair":
Return: False

task description: TASK_DESCRIPTION
layout criteria: LAYOUT_CRITERIA
object description: OBJECT_DESCRIPTION
object client requested: OBJECT_LOOKING_FOR

Remember, you should only return "True" if the
     object should be placed in the room / tabletop
      and "False" if the object should not be.
Do not include any other words in your response. It
      is extremely important.
```

At last, we conduct many verifications to remove assets that humans deem unsuitable given the room type and layout instruction (e.g., a 3D asset of an entire city should not appear in an indoor scene).

B.2. Evaluation

Evaluating the quality of generated 3D layouts requires metrics that measure both physical plausibility and semantic coherence. In this section, we introduce the evaluation prompts we feed to VLM to assess the performance of layout generation systems.

We measure the positional and rotational *Semantic coherency* score with the following prompts.

```
You are an interior designer.
Given generated renderings of a room, your job is
     to evaluate how well an
automated 3D layout generator does.

The instruction given to the 3D layout generator
     was:

Evaluate the 3D layout generator as follows:
Assess the the relative position (do not consider
     orientation) between assets: determine if
     related objects are placed near each other in
     a way that makes sense for their use.
- Scoring Criteria for Position:
    100-81: Excellently Positioned - Related
         objects are positioned near each other
         perfectly, facilitating their combined use
         .
    80-61: Well Positioned - Most related objects
         are logically placed near each other, with
          few exceptions.
    60-41: Adequately Positioned - Some related
         objects are not optimally placed,
         impacting their use together.
    40-21: Poorly Positioned - Many related objects
          are placed far apart, hindering their
         joint use.
    20-1: Very Poorly Positioned - Related objects
         are placed without consideration for their
          relationship, severely affecting
         functionality.

Please assess the image based on how coherently its
      layout aligns with the given target criteria.
Please provide justification and explanation for
     the score you give, in detail.
Always end your answer with "### my final rating is
     : [replace this with a number between 1-100]".
      This is extremely important.
```

```
You are an interior designer.
Given generated renderings of a room, your job is
     to evaluate how well an
automated 3D layout generator does.

The instruction given to the 3D layout generator
     was:


Evaluate the 3D layout generator as follows:
Assess the coherency of asset orientation: Evaluate
      if related objects are oriented relative to
     each other in a way that makes sense for their
      use.
- Scoring Criteria for Orientation:
    100-81: Excellently Oriented - The orientation
         of objects perfectly complements their use
          and relationship with each other.
    80-61: Properly Oriented - Most objects are
         oriented sensibly relative to each other,
         with minor misalignments.
    60-41: Adequately Oriented - Several objects
         have orientations that do not fully
         support their use or relation.
    40-21: Poorly Oriented - Many objects are
         oriented in ways that detract from their
         functionality or relation.
    20-1: Very Poorly Oriented - Objects are
```

```
        oriented without any apparent logic,
        severely undermining their intended use
        and relationship.

Please assess the image based on how coherently its
     layout aligns with the given target criteria.
Please provide justification and explanation for
    the score you give, in detail.
Always end your answer with "### my final rating is
    : [replace this with a number between 1-100]".
     This is extremely important.
```

We measure the *Physically-grounded Semantic Alignment Score (PSA)* with *Collision-Free Score (CF)*, *In-Boundary Score (IB)*, and the overall prompt alignment score following prompt.

```
You are an interior designer.
Given generated renderings of a room, your job is
    to evaluate how well an
automated 3D layout generator does.

The instruction given to the 3D layout generator
    was:


Evaluate the 3D layout generator as follows:
Layout criteria match: On a scale of 1 to 100, how
    well does the layout (i.e. just the layout not
     the assets) capture the essence of the
    specified layout criteria?
- Scoring Criteria:
    100: Excellent: The layout of the scene (i.e.
         relative position and pose of objects in
         the scene) align very well with the
         criteria.
    80: Good: The layout of the scene mostly align
         with the criteria (only ~10% assets do not
         ).
    60: Ok: The layout of the scene somewhat align
         well with the criteria (only ~30% assets
         do not).
    40: Poor: The layout of the scene (over ~50% of
          the assets) do not align with the
         criteria.
    20: Very Poor: The layout of the whole scene
         does not capture the target criteria at
         all.

Please assess the image based on how coherently its
     layout aligns with the given target criteria.
Please provide justification and explanation for
    the score you give, in detail.
Always end your answer with "### my final rating is
    : [replace this with a number between 1-100]".
     This is extremely important.
```

B.3. Human Eval to validate our metrics

We follow our baseline I-Design, evaluating layouts using physical plausibility metrics and GPT-4o-rated scores. To further evaluate the alignment between GPT-4o ratings and human ratings, we conducted a user study. Following prior work [34], we recruited five graduate students to rank the methods based on position, orientation, and overall performance. They were given the same instructions used as prompts for the GPT-4o evaluator. We collected 495 ratings for each method and metric pair. We converted GPT-4o scores to rankings and computed Kendall's Tau [37] in Table 6, showing **strong user-user agreement and user-GPT agreement**. Table 5 reports the average rankings based on user and GPT-4o ratings, demonstrating **strong correlation with the results in our proposed metrics**.

	User			GPT-4o		
	Position	Rotation	PSA	Position	Rotation	PSA
LayoutGPT	1.91	1.86	2.77	2.00	1.82	2.83
Holodeck	3.44	3.50	3.10	3.20	3.43	3.10
I-Design	2.86	2.91	2.64	2.89	2.86	2.62
LAYOUTVLM	1.79	1.73	**1.50**	1.91	1.89	**1.45**

Table 5. Average ranks based on user ratings and GPT4-o scores.

	Position	Rotation	PSA
Within Users	0.51	0.57	0.50
Users With GP4-o	0.49	0.61	0.46

Table 6. User-User and User-GPT4o Alignment/Agreement

C. More Qualitative Comparison

We provide a qualitative example of LAYOUTVLM with and without self-consistency loss in Figure 6

(a) LAYOUTVLM (b) w/o self-consistency decoding

Figure 6. Comparison when w/ and w/o self-consistency decoding.

In Figure 7, we present more qualitative examples of layouts generated by LAYOUTVLM and baseline methods. LAYOUTVLM consistently outperforms baseline methods across all room types in terms of both physcial plausibility and semantic coherency. In the Living Room example, LAYOUTVLM excels in identifying semantic asset group by clustering sofa and table together. By leveraging spatial reasoning from VLMs, we are able to stack assets toegther, as shown in the Deli example.

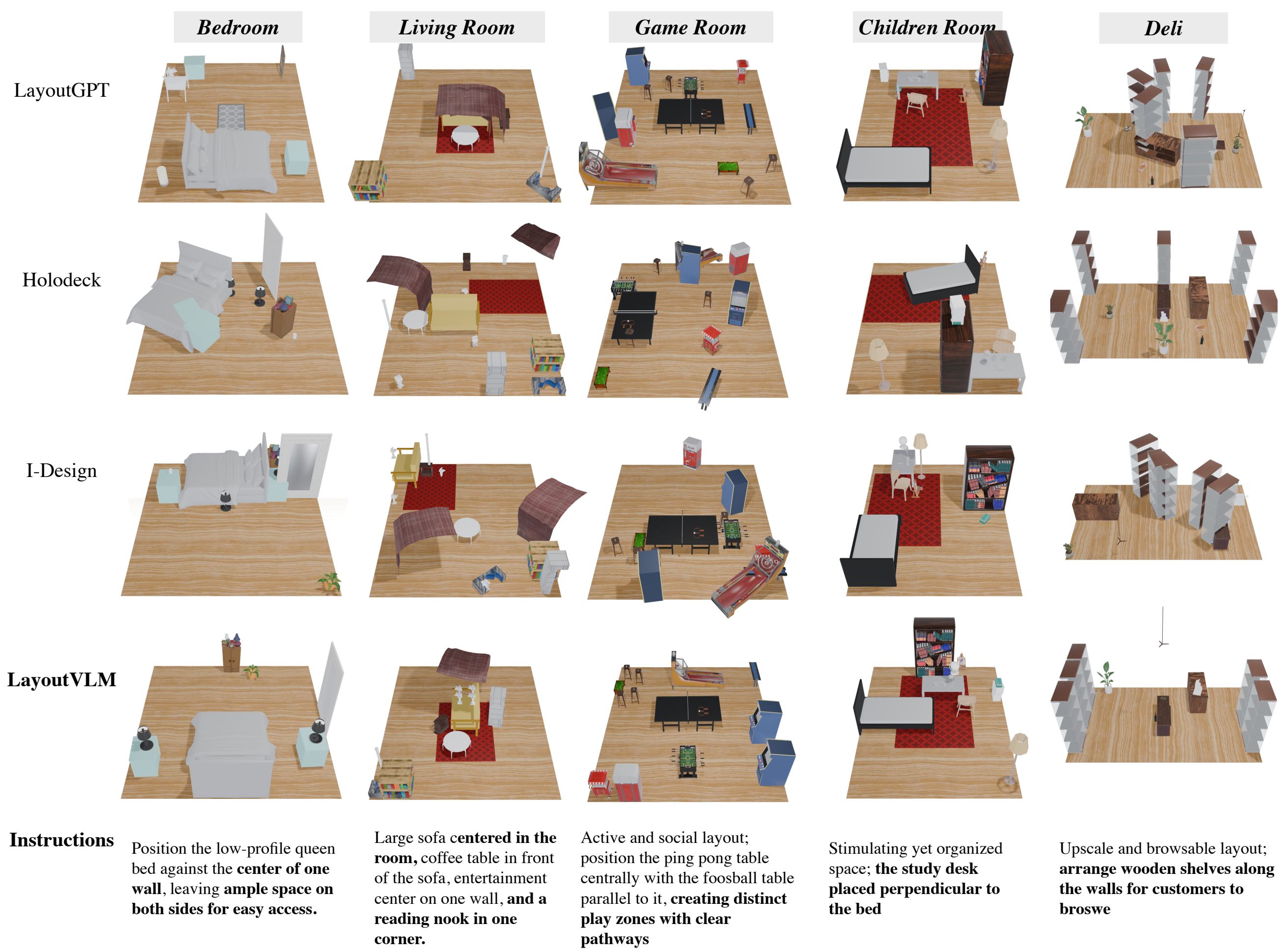

Figure 7. More qualitative comparison with baseline methods in generating layouts based on detailed language instructions.