FirePlace: Geometric Refinements of LLM Common Sense Reasoning for 3D Object Placement

Supplementary Material

In Section 6, we discuss the limitations and potential improvements to FirePlace, and comment on societal impact in Section 7.

We elaborate on the designs of metrics (specifically the energy and plausibility scores introduced in the main paper) in further detail in Section 8.

We discuss the prompts and algorithms used in constraint outline (Section 9), anchor object extraction (Section 10), interaction surface extraction (Section 11) and continuous parameter estimation (Section 12). We also share the implementations of our constraint functions in Section 13.

In Section 14, we explain how we adapted Holodeck and LayoutGPT for the object placement task, and discuss the prompts used. Further details on the evaluation dataset is provided in Section 15.

In Section 16, we show qualitative consequences of ablations done in the main paper, and also do additional experiments on the performance effects of scaling *down* inference compute used in Batched Visual Selection, further demonstrating the benefit of scaling inference compute for the visual selection task.

In Section 17, we show the performance of FirePlace on *image* inputs that depict a single example of an object placement, where FirePlace is tasked to generate similar placements. Finally, in Section 18, we show superior performance of FirePlace over baselines (for both text and image inputs), based on comparisons done by an MLLM.

6. System Limitations

Since our method uses MLLMs in every step of the placement generation process (with the exception of Step 4 in Figure 2), latency is a limitation of our approach. Our system currently takes on the range of 30 seconds to 2 minutes per object placement, depending on the number of prior objects within the scene (with more anchor object candidates, Batched Visual Selection must select among more options, creating more calls to the MLLM) and the number of surfaces that get extracted. Additionally, much of the computation time was used for rendering object placements. For our experiments, we only used CPUs for FirePlace, so it is likely possible to be sped up using GPU rendering.

Figure 6 shows some qualitative examples of common failure cases for FirePlace. They often come from intersections between the placed objects and preexisting objects within the environment (which can be addressed using intersection constraints), or failure to generate comprehensive sets of constraints (leading to under-constrained placements), or failure to prune the set of generated placements, often when the item being placed is small compared to the rest of the scene. In other cases, FirePlace may choose the incorrect object for the anchor object or the incorrect surfaces of objects, a limitation inherited from existing MLLMs. As MLLMs improve, we expect these issues to be mitigated.

Figure 6. **Common failure modes.** On the left, the placement of the object overlaps with preexisting objects, due to the constraint library not including a constraint to minimize intersections. In the middle, the placement of the chair was not constrained beyond contact to the ground, but additional constraints should have been generated (such as parallelism between the backs of the masked chair and the adjacent chair). On the right, the plausibility pruning step failed to remove implausible placements in the event of under-constrained placements (the bottom of the books are in contact with the table, but is overhanging), leading to a placement result that features the book floating over the edge of the table.

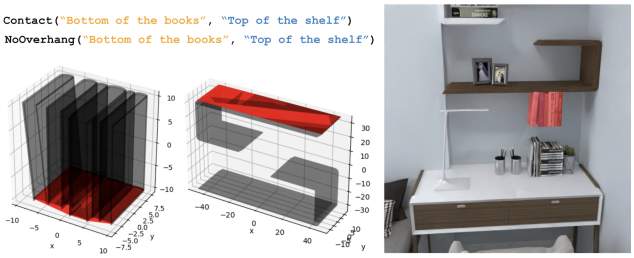

Figure 7. **When surface extraction is done in canonical space, but the object is rotated in world space.** A failure case where a stack of books is placed *under a shelf* because contact constraints were enforced for a upward-pointing (in the canonical space) surface of the shelf that *has been rotated by the artist* in its final position in the world frame.

Our surface extraction method extracts surfaces of anchor objects in canonical space. This assumes that surfaces of anchor objects that are pointing, say, upwards in the canonical space are *also* pointing upwards in the world

space once the anchor object is transformed. In rare cases within our dataset, this is not a valid assumption, the resultant optimized placements reflect this. An example can be seen in Figure 7.

7. Societal Impact

We do not foresee any substantial negative impacts of our work, beyond the inheritance of potential bias that may already be inside the MLLMs that currently exist. We anticipate that ongoing and future efforts in reducing MLLM bias will mitigate this.

8. Design of Metrics

Evaluating object placement is tricky. On the one hand, we have a groundtruth placement created by human artists with which we can compare generated placements, but such groundtruth accounts for only *one* of *many* possible object placements (*e.g.* consider placing a cup on a table). To mitigate this, energy and plausibility scores guard against the "precision" and "recall" of the constraint functions that are generated.

Consider the simple example of placing a cup on top of a table. Our system creates constraint functions that should be *minimally valued* at the groundtruth positions. This is what the *energy score* communicates – the *proportion* of constraint functions constructed that are minimal (< 0.01) when evaluated on the groundtruth placement generated by the human artist. If energy score is 1, this means that all the constraint energy functions generated for placing the cup (e.g. parallel constraints between bottom of cup and the top of the table) are "correct" according to the one groundtruth example. The energy score can be low when the placement is *over*-constrained, or when the constraint functions have low "precision".

On the other hand, what if the constraints are *under*-constrained? The plausibility score seeks to measure this. Assume that the parallel constraint mentioned above was the *only* one generated. As you may imagine, many solutions satisfying this constraint would render the object floating in parallel to the table, but not necessarily in contact or within the tabletop's perimeter. This would score poorly on plausibility by the MLLM, since many final renderings will display placements that are not physically realistic or disagreeing with the input text prompt (see Figure 8 for examples). In this case, the plausibility score will indicate that the placement is *under*-constrained, or that the constraint functions have low "recall". The prompts (and rubric) used to evaluate the plausibility score are given in Figure 21.

Prior works like [49] use MLLMs for evaluation, and have shown through human evaluations that MLLM evaluations are aligned with human preferences. In this paper, we observe the same for the plausibility score. As shown in Figure 8, plausibility scores capture the extent to which placements are physically feasible and semantically plausible. Additionally, as noted in the main paper, when using plausibility scores to decide the better placement between two samples, they agree with humans 89.92% of the time, when using Gemini 1.5 Pro. See Section 4.3 for more.

9. Constraint Outline Generation

We use the prompt shown in Figure 23 to query the MLLM to generate constraint outlines. As part of the prompt, we also provide the rendering of the input 3D scene ([Source Layout Rendering]), the input text prompt ([Placement text prompt]) and a doc string describing the constraint functions within our constraint function library ([Constraint library doc string]). The contents of this doc string is shown in Figure 22.

10. Extraction of Anchor Objects

We use Batched Visual Selection to select for object instances that match language descriptions of the anchor objects from the constraint outline. Beyond the procedure outlined in Algorithm 2, we use the prompts shown in Figure 24 to select among every batch. The result of doing so is shown in Figure 9. Their image segmentation masks are derived from the USD rendering process of the input 3D scene.

11. Extraction of Surfaces

To execute the surface extraction steps in Algorithm 1, we will first describe how **DirExtr()** works, then elaborate on the geometric processing underlying **SurfExtr()**.

In **DirExtr()** the MLLM is prompted to generate surface normals that match the language descriptions of the surfaces that best match the constraint outline descriptions of surfaces that participate in the constraint. For instance, if the transformable object should be sitting on the seat of the chair, the surface that should be extracted from the chair should be pointing upwards (*i.e.* the seat). This is done through the prompt shown in Figure 25. Note that in our experiments, we provide the MLLM with the 6 major directions (left, right, front, back, up, down) that surfaces can point, but our method can also work with more directions by updating the prompt accordingly. After this is done for the anchor and transformable objects, geometric processing algorithms are then called to extract the surfaces that point in these directions.

To this end, we first filter the faces of the object mesh to a subset that have face normals within some threshold level of cosine similarity with the unit vectors corresponding to the extraction direction generated in the previous step. For the faces that have similar face normals to the extraction direction, we project the center of these faces along the desired

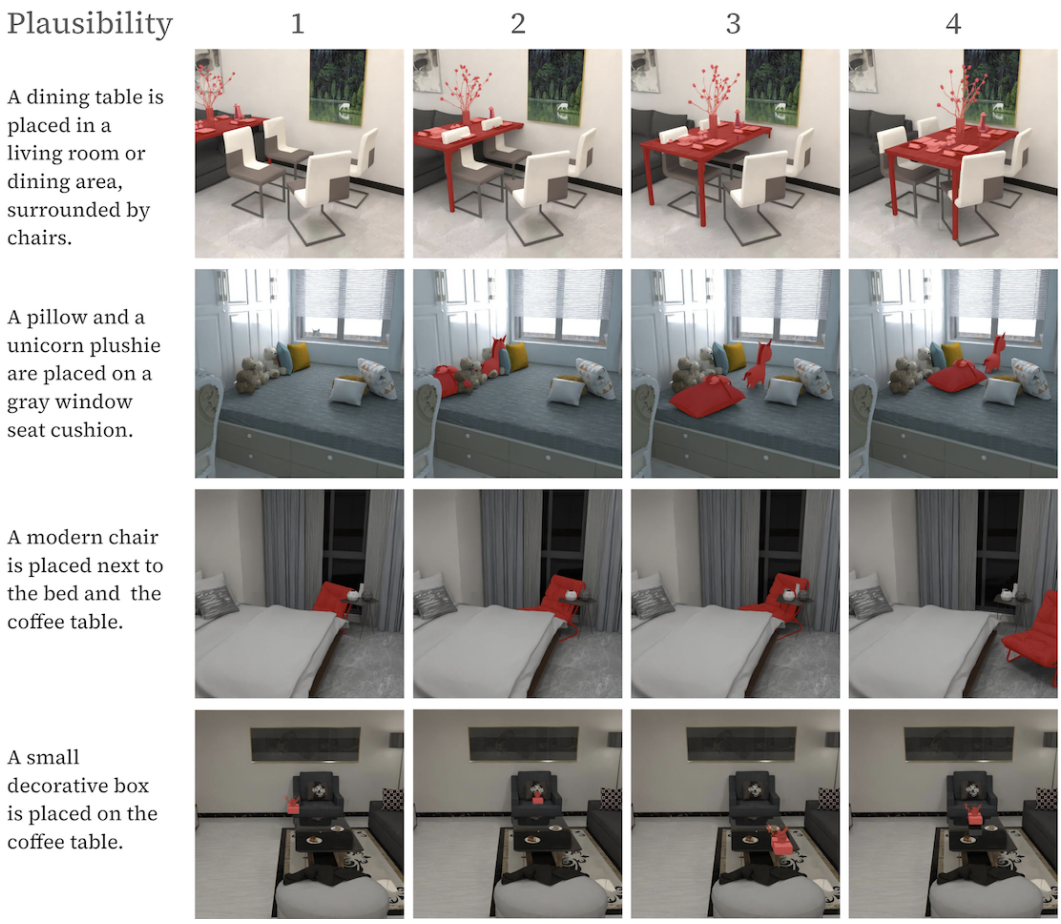

Figure 8. Examples of plausibility scores for different placements. The text prompts are shown on the left, and objects placed are shown on the right. A plausibility score of 4 is the maximum, and a 1 is the minimum. Refer to Figure 21 for the definitions of these scores.

surface normal, then cluster them using DBSCAN. This allows us to find different sets of faces that lie along a similar "level" along the desired surface normal. Faces in each set are then projected to the same level before a convex hull is fitted onto that group, which extracts a flat convex hull tightly circumscribing each set. Each of these planar convex hulls is an interaction surface candidate, and has a surface normal equal to the extraction direction.

This process leads to many candidate interaction surfaces and, depending on the level of geometric complexity of assets, may be prohibitively cumbersome/expensive/difficult to filter down to one using Batched Visual Selection. As such, we merge interaction surfaces that are close to each other within a certain distance threshold by keeping the interaction surfaces that have a larger area. Additionally, we recognize that many constraints can be adequately expressed using bounding box constraints, so we append the bounding box surface aligned with the extraction direction to the set of candidate interaction surfaces.

The process of choosing the correct interaction surfaces for the anchor and transformable objects is very similar to that of choosing the anchor object (See Section 10). We render the surfaces overlaid on top of the original object mesh using `matplotlib`, taking care to show no greater than 3 candidates for every batch. The selection is done according to the prompt shown in Figure 26.

Figure 15, Figure 16, Figure 17, Figure 18, Figure 19, Figure 20 show examples of the constraints constructed by FirePlace for different placement tasks using the surfaces extracted by this approach.

12. Parameter estimation

Once interaction surfaces of the anchor and transformable objects are extracted, we prompt the MLLM with renderings of the scene and `matplotlib` renderings of the interaction surfaces, according to the prompt shown in Figure 27. A documentation of the meaning of the continuous parameters of each constraint function is also provided. In our setting, only two of the constraint functions have continuous parameters (`CloseTo` and `FarFrom`); namely, max-

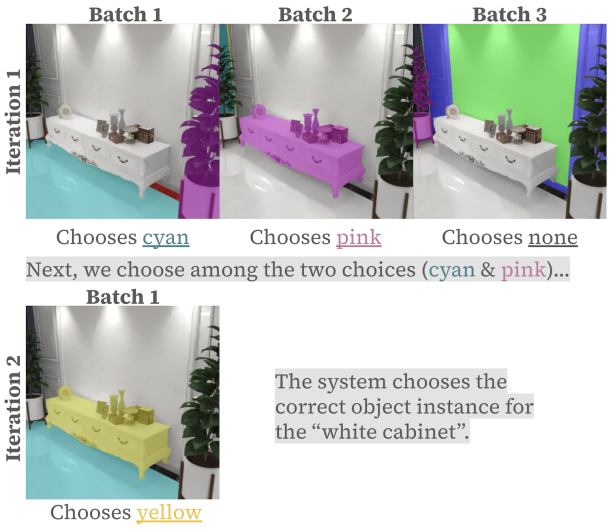

Figure 9. The batched visual selection process for a scene with only a few items. Here, the MLLM is tasked to find the anchor object corresponding to "the white cabinet" from a constraint outline generated. Each batch shows 3 options rendered in different colors (for batch size = 3), and the MLLM chooses object instances that best match the description by indicating the color of the mask in each round. The chosen instances across each batch are merged and the process is repeated until only one object instance is chosen. This is done using the prompt shown in Figure 24.

imum/minimum distances between the two interaction surfaces.

13. Constraint functions

The constraint functions outlined in Section 3.4 are implemented as binary functions that evaluate to 0 when a geometric relationship is satisfied between two interaction surfaces, `p1` and `p2`. Below, we describe the implementations of each:

1. `Parallel(p1, p2)` returns

$$\min(|1 - n_1^T n_2|, |-1 - n_1^T n_2|)$$

where n_1 is the surface normal of `p1` and n_2 that of `p2`. This function is minimal when either the surface normals are aligned, or pointing in parallel but opposite directions.

2. `CloseTo(p1, p2, dist)` returns

$$k \max(d(\texttt{p1}, \texttt{p2}) - \texttt{dist}, 0)$$

for some scaling constant k (which we set to 0.1) and distance function d between two interaction surfaces. This function is minimal when $d(\texttt{p1}, \texttt{p2})$ is *smaller* than `dist`.

3. `InFrontPlane(p1, p2)` first finds the pairwise vector differences between the vertices of `p1` and `p2`, then evaluates the dot product between each vector difference and the surface normal of `p1`, n_1. The return value of this function is

$$\min(0, \max(\{-\frac{v_{ij}}{|v_{ij}|}^T n_1\}_{ij}))$$

for all vector differences v_{ij} between the ith vertex in `p1` and the jth vertex in `p2`.

4. `Contact(p1, p2)` is

`InFrontPlane(p1,p2)+InFrontPlane(p2,p1)`

which is minimal when `p1` and `p2` are either in contact with each other or coplanar.

5. `NoOverhang(p1, p2)` is more involved. Let p_1 be the set of vertices of `p1` (with N vertices and 3 coordinates, $p_1 \in \mathbb{R}^{N\times 3}$) and p_2 be that of `p2`. Let n_2 be the normal vector of `p2`. We then first project p_1 onto `p2`,

$$p_1|_{\texttt{p2}} = p_1 - ((p_1 - o) \cdot n_2)n_2$$

for some arbitrary vertex o from p_2. Then, we sample 1000 points from the region bound by $p_1|_{\texttt{p2}}$, which we call $q \in \mathbb{R}^{1000}$. We can then calculate whether q is contained within the bounds of p_2. The final output value of this function is defined as

$$1 - \frac{1}{1000}\sum_{i=1}^{1000} \mathbb{I}_{inside}(q_i)$$

where $\mathbb{I}_{inside}$ is an indicator function indicating whether q_i lies on the inside of `p2`. The function is minimal when the projection of `p1` onto `p2` is entirely contained within `p2` (*i.e.* $\forall i, \mathbb{I}_{inside}(q_i) = 1$).

14. Prompts and Constraint Functions for Holodeck and LayoutGPT

For the Holodeck and LayoutGPT baselines in our experiments, we use the prompts shown in Figure 28 and Figure 29 for Holodeck, and Figure 30 for LayoutGPT. Note that we use the prompts found in the implementations released on github [1], modified only to provide additional information about the objects that already exist within the scene. To do this, we use Gemini to caption object renderings of assets found within the scene, and provide these captions to both methods via the prompts.

For LayoutGPT, we provide the caption of each object alongside their bounding box information (length, width, height, left, top, depth, orientation) as part of the prompt. These can be derived from their local-to-global transformation matrices, as well as the length, width, height of their bounding boxes in canonical space.

[1] Official Holodeck prompts and official LayoutGPT prompts

For Holodeck, these captions (*e.g.* "a brown curtain") are provided alongside object ID's (*e.g.* `object-34`), indicated by [Descriptions and labels of preexisting objects in the scene] in Figure 29. Holodeck can then reference the object ID's in the construction of the bounding box constraints. For the bounding box constraints used by Holodeck, we implement bounding box constraints as binary constraint functions, similar to those in Section 3.4, with the big difference being that they operate on *bounding boxes* instead of *interaction surfaces*. The Holodeck baseline has access to a bounding box constraint library composed of the following constraints: (1) `FaceTo` (that the front face of the object bounding box faces the center of another bound box), (2) `near` (that objects are closer than 150 cm from each other and further than 50 cm away) (3) `far` (object are further than 150 cm away from each other) (4) `infront`, (5) `sideof`, (6) `leftof`, (7) `rightof`, (8) `behind`, (9) `ontop`, (10) `centeraligned_front`, (11) `centeraligned_side`. These are all bounding box constraints originally used in the Holodeck implementation. For comparisons between Holodeck and FirePlace, we use the same constraint solver (with the same parameters) to solve constraints created by both methods.

As mentioned in the paper, experiments on FirePlace, LayoutGPT, Holodeck all use the same MLLM (Gemini 1.5Pro) for a fair comparison.

15. Evaluation Dataset

All scenes used for evaluation of our method and the baselines are in Universal Scene Descriptor (USD) format, which contains meshes of all objects, architectural elements and photorealistic materials. When choosing the transformable objects for each placement task, our goal was to choose objects for which the correctness of its final position depends on the successful identification of "anchor" objects and the relevant constraints. As such, the transformable objects selected for our evaluation placement tasks are often furniture pieces (chairs, tables, refrigerators ...etc) and decorative items (books, picture-frames, wall art...etc). Figure 10 shows the distribution of the number of possible "anchor" objects in each of the placement tasks composed of architectural elements and furniture/household objects. This motivates the need for Batched Visual Selection to make the visual selection task easier by breaking down the decision process into multiple stages.

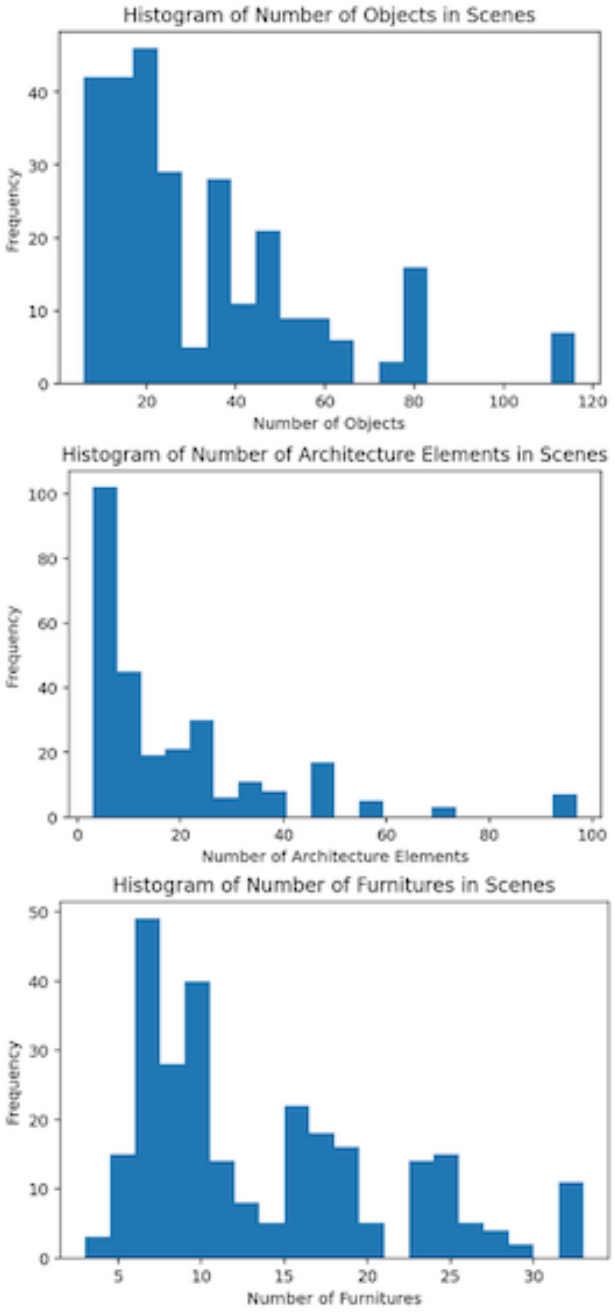

Figure 10. Distributions of the number of objects (furniture and architectural elements) within the placement tasks used for evaluation.

16. Ablations

16.1. Qualitative Examples of Ablations

The ablations tested in Table 3 have qualitative consequences on the placements that get generated. Figure 11 and Figure 12 show this for two placement tasks. Note that in both cases, the transformable object must be placed into a shelf-like object, and that it's crucial to have access to the low-level geometry, which bounding box representation do not provide ("– Geometry"). In both cases, purely using the MLLM to choose among randomly generated placements leads to suboptimal placements, oftentimes creating final placements that feature the object floating in air (*e.g.* "–

Figure 11. An example of the effects of ablations in Table 3 on the placements. In this example, FirePlace is tasked to place the coat into the closet.

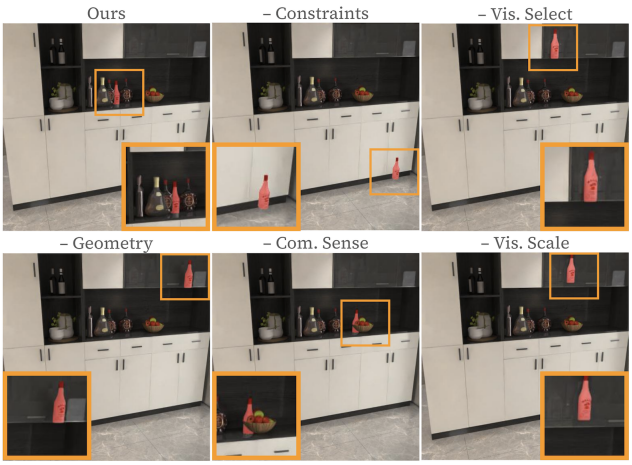

Figure 12. An example of the effects of ablations in Table 3 on the placements. In this example, FirePlace is tasked to place the bottle on the cabinet.

Constraints" in Figure 12 shows the bottle floating slightly above the ground). We can also see that plausibility pruning tends to get rid of implausible overlaps that may happen when placing the object according to raw geometric constraints – in both Figure 11 and Figure 12, removing plausibility pruning ("– Com. Sense") leads to final placements that overlap with assets already in the scene. Finally, removing the ability to visually select anchor objects (as opposed to selecting anchor objects based on text annotations) and removing the ability to scale inference compute for Batched Visual Selection both lead to incorrect placements, due to the wrong anchor object/surfaces being selected for the constraints.

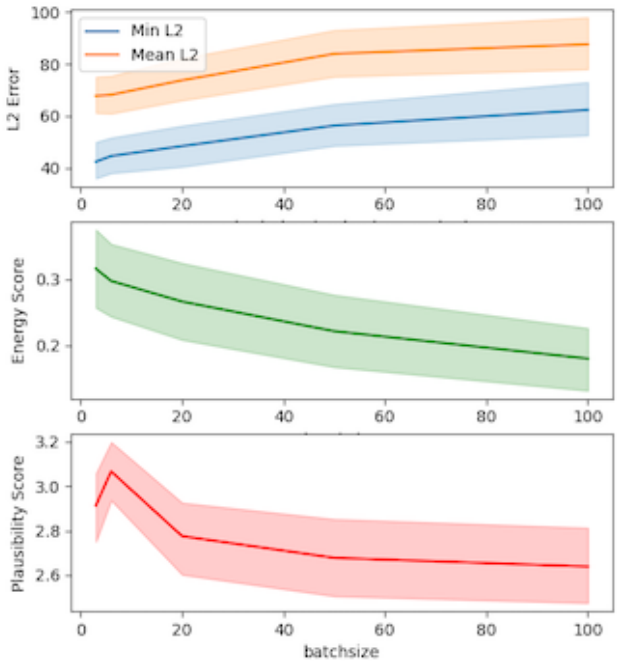

Figure 13. Increasing the batch size within Batched Visual Selection leads to lower performance on placement tasks.

Metric	LayoutGPT	Holodeck	Ours
Min L2 error cm (↓)	126.19	91.25	**43.69**
Mean l2 error cm (↓)	166.11	136.51	**68.84**
Visibility score (↑)	0.69	0.59	**0.88**
Plausibility score (↑)	2.31	1.99	**2.92**
Energy score (↑)	–	0.17	**0.38**

Table 4. Comparison with Baseline for *image inputs*

16.2. Inference Compute Scaling for Batched Visual Selection

Figure 13 displays the trends on the performance metrics as we increase the batch size (lower the level of inference compute) used by Batched Visual Selection. This means that for the selection process of anchor objects and interaction surfaces, an MLLM must choose among larger sets of options at a time. We can observe a downward trend in plausibility and energy scores (due to incorrectly selected object instances and and interaction surfaces), and also an upward trend in both mean and minimum L2 errors, suggesting that the resultant placements become further away from the groundtruth as MLLMs are prompted to choose among more and more visual options at a time. Our default settings uses a batch size of 3, and for Figure 13, we increase the batch size to 6, 20, 50 and 100.

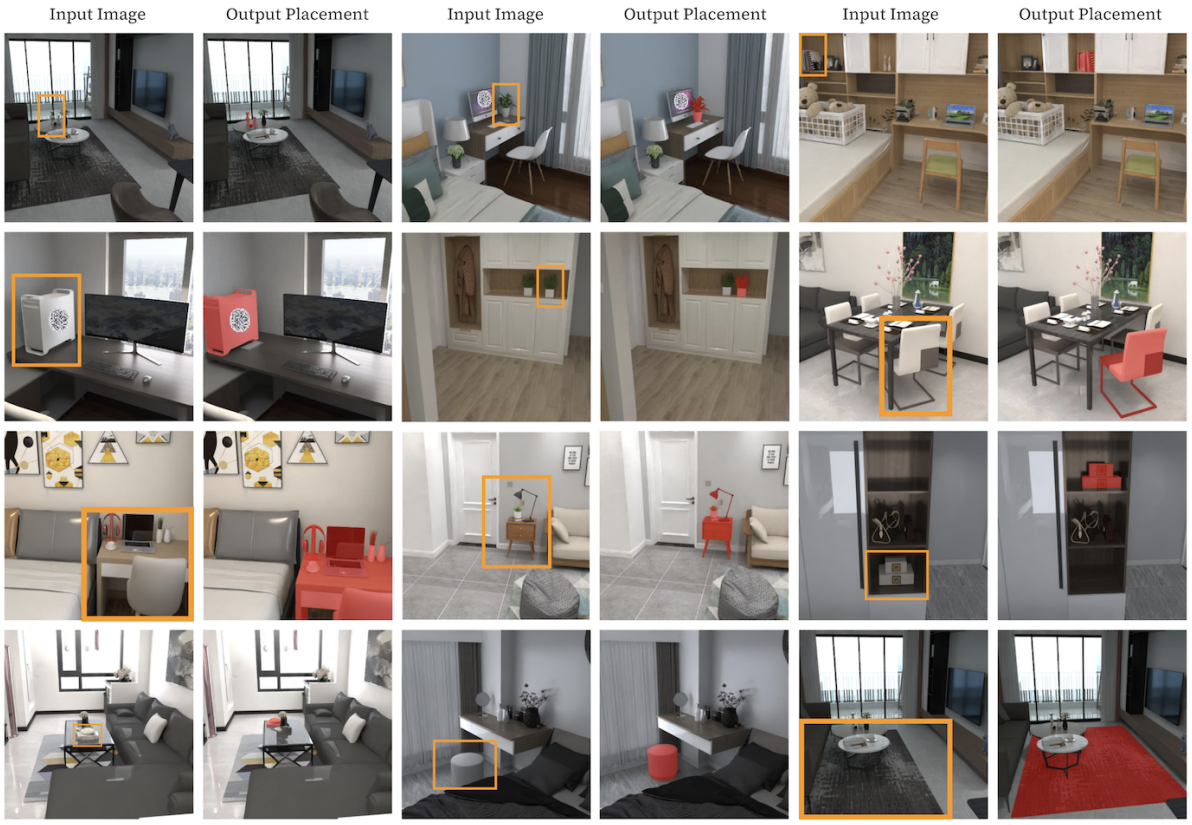

Figure 14. Qualitative results of object placement when FirePlace is given *image* inputs of placement examples. Note how generated placements follow the semantics of object placements shown in the input image to varying degrees, but can vary in their final positions.

17. Performance on Image Inputs

Since FirePlace uses an MLLM for constraint outline generation, an additional input modality that we can demonstrate – besides language annotations – is an image *example*. Given an image showing one possible placement of the object, FirePlace can generate variations of placements that are *semantically similar* in the sense that output placements capture the underlying constraints and placement considerations of the image example. Table 4 shows that our method also outperforms the baselines across the metrics on this task. Figure 14 shows object placements generated from our system when given different placement examples. For this experiment, the prompts of our method, Holodeck, and LayoutGPT are changed accordingly to insert the image example instead of a text prompt, as done for the experiments in the main paper.

18. Using MLLMs to Compare FirePlace to Baselines

In addition to the human evaluations that indicate FirePlace's superior performance, we can also use MLLMs to do pairwise comparisons, by giving it shuffled pairs of renderings (image A and B) generated by our method and the two baselines. The objective (text input or image input) is also provided to the MLLM. The prompt used is shown below:

```
Between Image A and Image B, which is a
    better match to the objective, in
   terms of its placement of the object
    masked in red?
Describe the scene and describe the
   object masked in red (what is it?
   Where should the object be according
    to the objective?), then respond
   with 'first' or 'second' in json:

```json
{
  "final_answer": "A"/"B"
}
```
```

The results generated with Gemini 1.5Pro for text inputs is shown in Table 6 and the result for image inputs (See Section 17) is shown in Table 5, showing FirePlace's superior

	vs. LayoutGPT	vs. Holodeck
LayoutGPT wins	-	0.56
Holodeck wins	0.44	-
Ours wins	**0.72**	**0.72**

Table 5. Win-rate comparison between our method and LayoutGPT and Holodeck according to Gemini 1.5Pro as judge, for placement tasks with image input.

	vs. LayoutGPT	vs. Holodeck
LayoutGPT wins	-	0.54
Holodeck wins	0.46	-
Ours wins	**0.70**	**0.72**

Table 6. Win-rate comparison between our method and LayoutGPT and Holodeck according to Gemini 1.5Pro as judge, for placement tasks with text input.

performance over both baselines in both task settings.

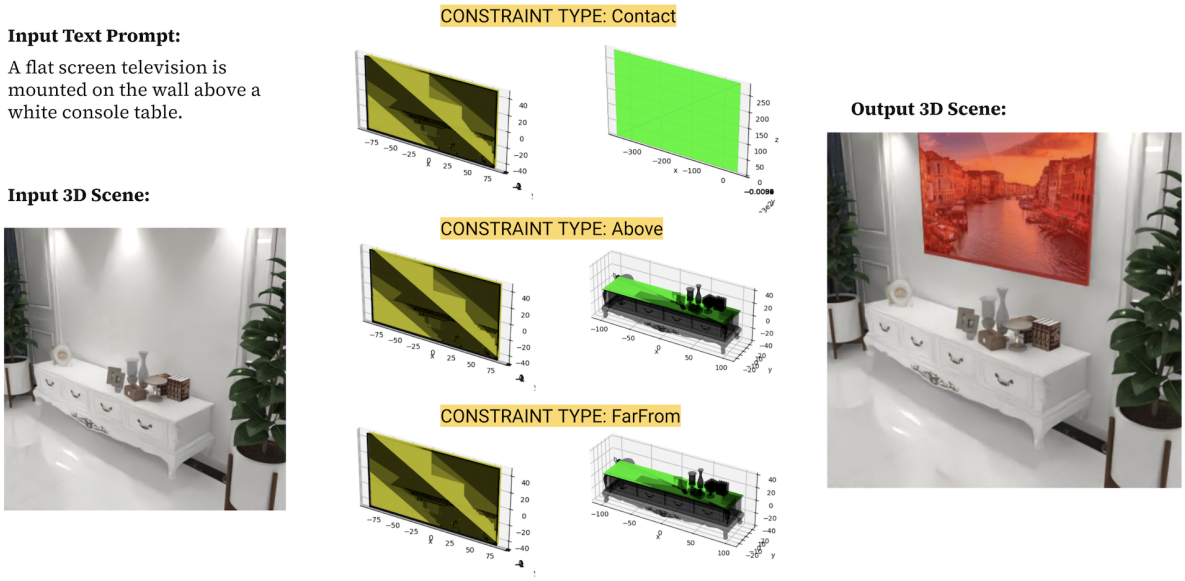

Figure 15. The constraints and interaction surfaces generated for the task of mounting a TV. For clarification, the contact constraint is enforced between the back of the TV and the wall (visualized as a plane). Note that there are multiple wall meshes within this scene (for instance, see Figure 9 – blue in Batch 3 and cyan in Batch 2 are both alternatives), and that Batched Visual Selection chooses the correct one

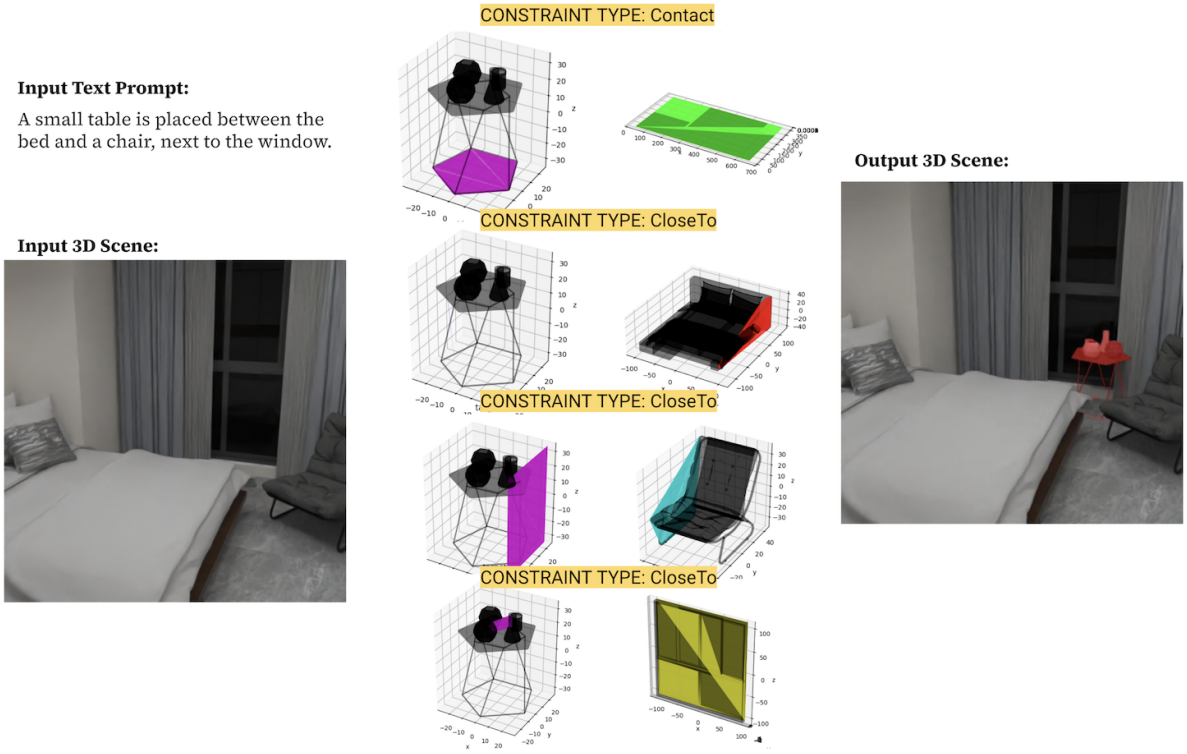

Figure 16. To place the small table into the scene, the constraints generated first identifies a Contact constraint between the bottom of the table and the floor, then uses various CloseTo constraints to capture its rough position in the room - the final position generated is close to all 3 surfaces chosen.

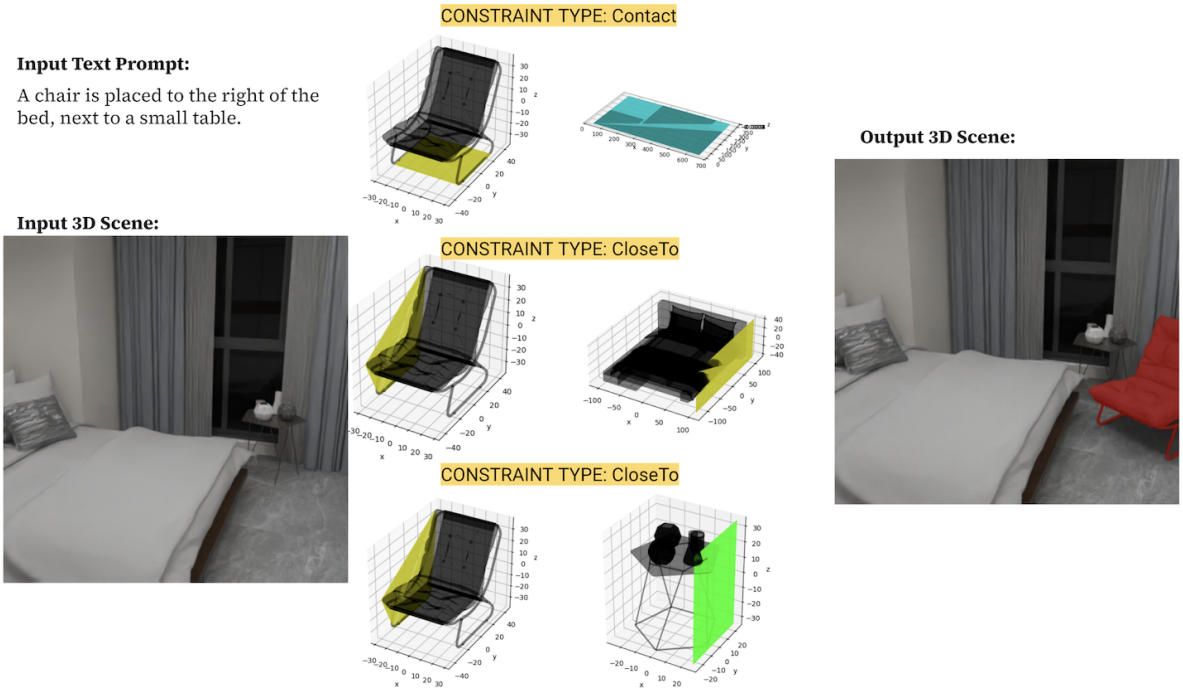

Figure 17. Similar to Figure 17, FirePlace enforces a contact constraint with the floor, then uses CloseTo constraints to restrict plausible placements.

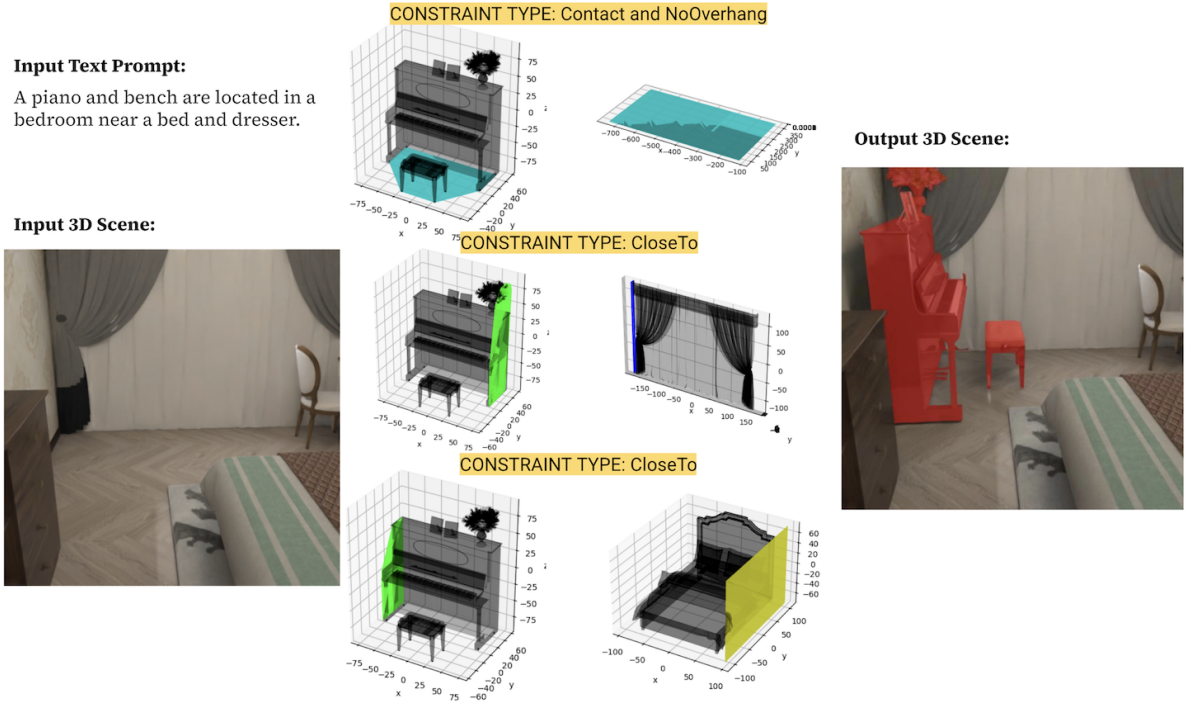

Figure 18. Placing a piano into the room. FirePlace successfully discerns that the left and the right of the piano must be close to different things, and that the right side should be closer to the left of the curtains, and the left should be closer to the bed.

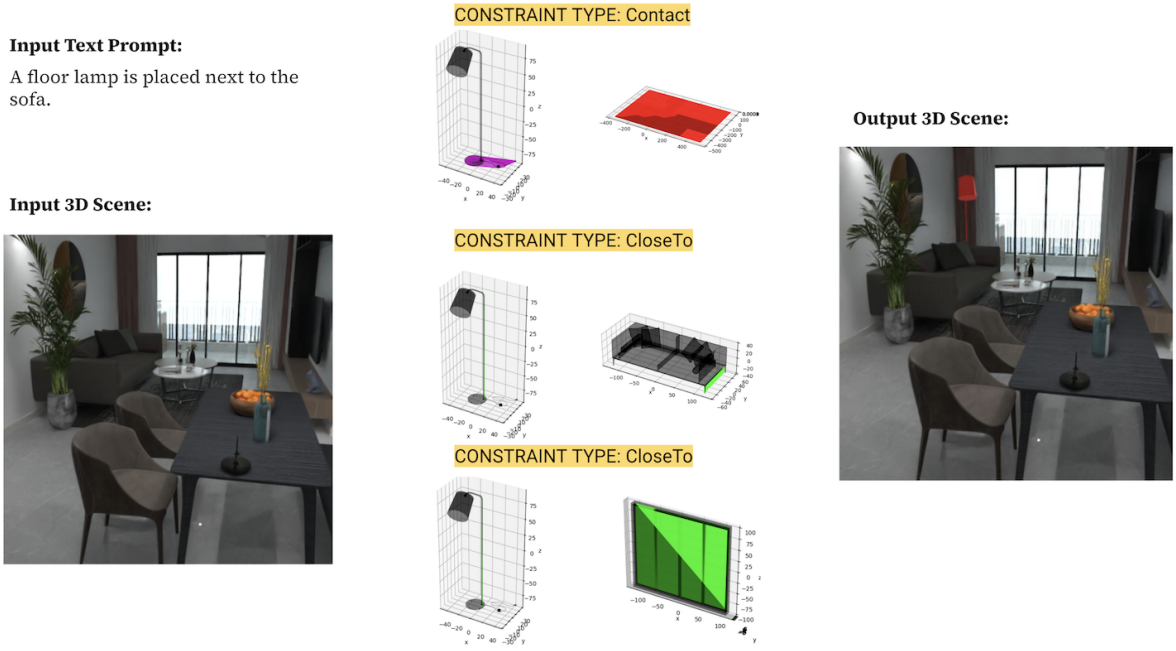

Figure 19. Placing the lamp in the living room is done by locating near-by objects and enforcing CloseTo constraints.

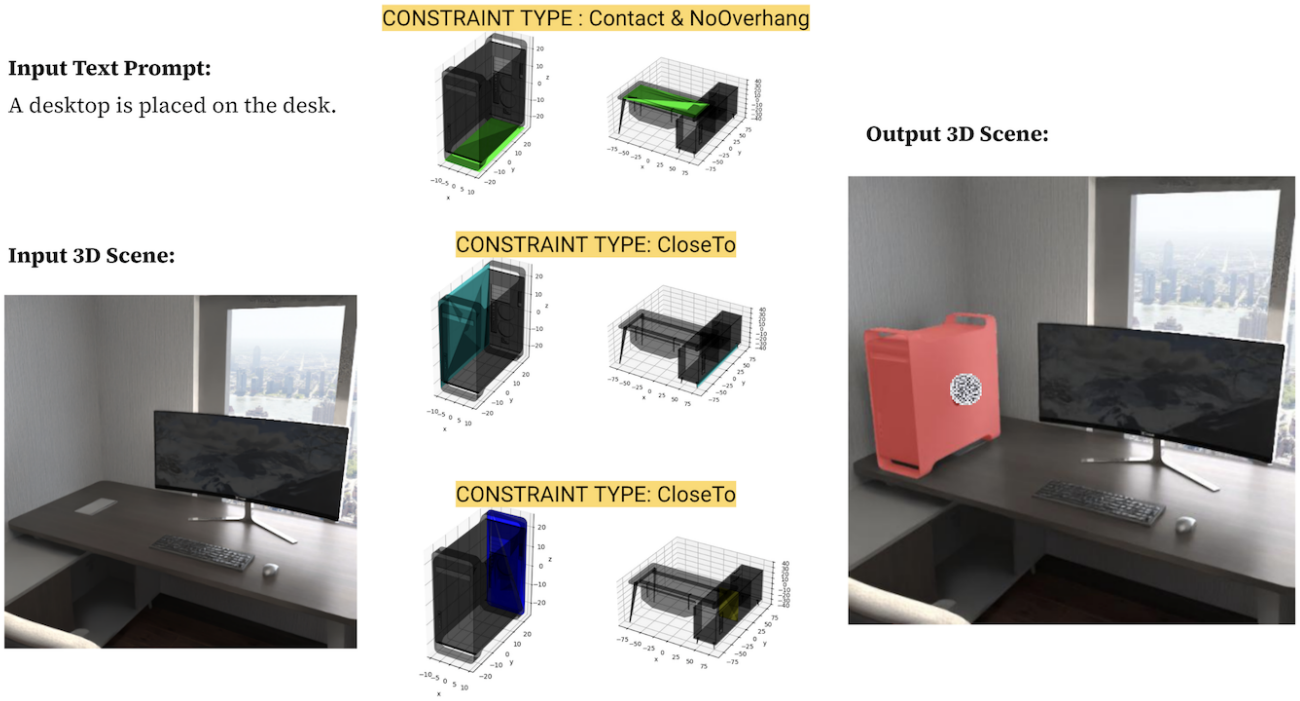

Figure 20. The desk is L-shaped, meaning that in order to insert place the desktop, it must correctly extract the top of the desk, and not just use the top face of the desk's bounding-box. We see here that it enforces Contact and NoOverhang constraints to the table, guides the placement according to other CloseTo constraints.

````
The desired placement of the object masked in red can be described by:
[Placement text prompt]

The predicted placement of the object is shown in the predicted image below:
[Masked rendering of predicted final placement,
with placed object in semi-transparent red mask.]

Please grade the predicted placement of the predicted placement of the object
   masked in red on a scale of 1-4, where:
1 = either the target object is not observed at all within the scene, or the
   placement is physically implausible (e.g. the object is floating in the air
   or intersecting with other objects)
2 = The object is observed within the scene, but its placement differs
   substantially from the placement observed in the reference image.
3 = The object is observed within the scene, and its placement is sensible (e.g
   . the target object is NOT floating in air or intersecting with other
   objects), and the placement of the target object is different from the
   reference image, but not in substantial ways.
4 = a totally sensible placement of the target object, and captures both valid
   physics (e.g. the target object is NOT floating in air or intersecting with
   other objects) as well as considerations for function, accessibility, and
   aesthetics.

Please reason about what is in
Describe the scene and describe the object masked in red (what is it? Where
   should the object be according to the objective?), then respond with the
   final score in json:

```json
{
  "final_answer": 1/2/3/4
}
```
````

Figure 21. Prompt for extracting plausibility scores.

```
* Contact (surface1, surface2)
  Enforces that surface1 and surface2 are in contact with each other.
* FarFrom (surface1, surface2, const)
  This enforcs that surface1 and surface2 are at a AT LEAST some distance away
      from eachother specified by const, a float in CENTIMETERS in the life-
     size 3D scene.
* CloseTo (surface1, surface2, const)
  This enforcs that surface1 and surface2 are AT MOST some distance away from
     eachother specified by const, a float in CENTIMETERS in the life-size 3D
     scene.
* Parallel (surface1, surface2)
  This enforces that surface1 and surface2 are parallel.
* Above (surface1, surface2)
  This enforces that surface1 is ABOVE surface2 (note the order.) This does
     NOT ensure that the birdseye view of object1 and object2 overlap.
* NoOverhang (surface1, surface2)
  This enforces that surface1's vertical projection is entirely contained in
     surface2's projection. This is good when you'd like one surface to be
     entirely contained within another, for instance for physical stability.
  Also, if you ever want to ensure contact with floor, make sure to use
     NoOverhang as well.
```

Figure 22. Prompt for constraint documentation

```
[Source layout rendering]

The following text describes the target object that we would like to place in
    the image above: [Placement text prompt]
In each of the reference images below, a target object is masked in red. Even
    though this may be different from the target object just mentioned and even
    though the scene may be different, the placements of the object shares
    underlying patterns and structure across the scenes.

You are given a library of the following functions:
above: [Constraint library doc string]

For the {item_to_focus_on}, return a json that has a list. Each element of the
    list specifies:
1) in the field called 'constraints': the constraint function (e.g. Parallel)
2) in the field called 'surface1' a string describing surface1 -- if it's a
    surface on the target object (or collection of objects), refer to it as the
    'target object'. (e.g. bottom of the target object)
3) in the field called 'surface2' a string describing surface2 -- if it's a
    surface on the target object (or collection of objects), refer to it as the
    'target object'. (e.g. top of the table in the corner of the room)

Return a json marked by ```json at the very beginning, with a list of ALL of
    the constraints that is relevant to the {item_to_focus_on}.
NOTE: since none of the objects are floating in the scene, ALWAYS start off the
     list with a CONTACT constraint. What is the target object in contact with?
    Which surface of the target object is in contact with something else?

NOTE: specify distance relations between objects using the constraints `CloseTo
    ` and `FarFrom`, by spcifying which surfaces on the target object and the
    surounding objects are relevant to these distance relations.
For instance, if the target object is a potted plant at the corner of the room,
     you would use the CloseTo distance relation with respect to one side (left
    or right) of the plant, the surface of the correct wall, and another CloseTo
     distance relation between the back side of the plant and the other wall
    forming the corner.
Alternatively, if the target object is something mounted on the wall above
    something else, it would make sense to specify FarFrom constraint between
    the bottom surface of the target object and the surface of the object
    underneath.
Note that in both cases, you must specify a threshold distance in Centimeters.
    Use the assumption that the scene is real-life sized in 3D.

You are allowed to return an empty list if nothing is relevant to the
    description of the desired physical relation above.
```

Figure 23. Prompt for constraint outline generation

````
The following text describes the target object to be placed into the scene:
[Placement text prompt]

The target object is related to another 'anchor' object within the scene by the
    physical relation: [Constraint outline]
The surface of that 'anchor' object in the physical relation can be described
   by: [constraint outline]
We refer to this object as the anchor object.

In the following image, I want you to find the anchor object among the
   segmentation masks I show in different colors.
What is the color of segmentation mask of the the anchor object in the
   following image?[ concat color names /none]

[Insert masked rendering of anchor objects]

I want you to reason about it, then output a json, like:
```json
{'final_answer': " concat color names /none"
   }
```
respond with `none` if none of the segmentation masks in the image match the
   anchor masked object.
````

Figure 24. Prompt for extracting anchor objects

```
The following image shows our "anchor" object in a semi-transparent red mask.

[insert masked rendering of anchor object]

Here's a 3D plot of the [ANCHOR/TARGET] object in its canonical space.
In this plot:
The direction 'upwards' is described by the positive z direction. (down is
   negative)
the direction 'right' is described by the positive x direction. (left is
   negative)
the direction 'forwards' id described by the negative y direction. (backwards
   is positive)

[Insert 3D plot of ANCHOR/TARGET object ]

And the following text describes the target object: [Placement text prompt]

You've decided the following must be true:
[Constraint outline]

Think about the [ANCHOR/TARGET] object. Which object does that correspond to
   in the above?
Think about the surface where the interaction between the two objects is
   happening. Which way is it pointing on the [ANCHOR/TARGET] object?

I'd like you now to tell me how you would extract the relevant surfaces from
   our [ANCHOR/TARGET] object relevant for the placements described above. The
    definition of a relevant surface is one that is involved in physical
   constraints. If something is supposed to be to the RIGHT of this object,
   then it makes sense that a surface pointing to the right is relevant, since
   that surface can be usd to judge whether the object is truly to the right of
    the object.
If the anchor object were a table, and the target object is to be put onto the
   table then the relevant surface of interaction for the ANCHOR OBJECT is
   pointing UPWARDS. The surface of interaction for the TARGET OBJECT is
   pointing downwards.
The surface you extract does NOT NEED TO BE in physical contact with the other
   object. For instance, if a distance is to be maintained between two objects,
    think which surface is most relevant in that distance calculation.

I want you to reason about it, then output a json to specify the direction of
   the surface in the [ANCHOR/TARGET] object relevant to the interaction, like
   :


'''json
{'final_answer': up/down/left/right/front/back}
'''
You cannot return the word 'none'.
```

Figure 25. Prompt for extracting object surface directions.

```
The following image shows our "anchor" object in a semi-transparent red mask.

[Insert masked rendering of ANCHOR object]

Here's a 3D plot of the [ANCHOR/TARGET] object in its canonical space.
In this plot:
The direction 'upwards' is described by the positive z direction. (down is
   negative)
the direction 'right' is described by the positive x direction. (left is
   negative)
the direction 'forwards' id described by the negative y direction. (backwards
   is positive)

[Insert 3D plot of ANCHOR/TARGET object]

And the following text describes the target object: [Placement text prompt]

You've decided the following must be true:
[Constraint outline]

Think about the [ANCHOR/TARGET] object. Which object does that correspond to
   in the above?
Think about the surface where the interaction between the two objects is
   happening. Which surface ([LIST OF COLORS]) is relevant to the interaction?

I want you to reason about it, then output a json to specify the the surface in
    the [ANCHOR/TARGET] object relevant to the interaction, like:

'''json
{'final_answer': "[LIST OF COLORS]/none"
   }
'''
```

Figure 26. Prompt for choosing object surfaces based on the color of the surface in the visualization.

```
The following image shows our "anchor" object in a semi-transparent red mask.

[Insert masked rendering of ANCHOR object]

Here's a 3D plot of the [ANCHOR/TARGET] object in its canonical space.
In this plot:
The direction 'upwards' is described by the positive z direction. (down is
    negative)
the direction 'right' is described by the positive x direction. (left is
    negative)
the direction 'forwards' id described by the negative y direction. (backwards
    is positive)

[Insert 3D plot of ANCHOR/TARGET object]

And the following text describes the target object: [Placement text prompt]

You've decided the following must be true:
[constraint outline]

In order to enforce this constraint, there's a few parameters you must specify.
Use your visual judgment to determine the best values for the following
    parameters:
[Arg documentation string for each constraint function]

For the target object, you chose the surface visualized below:

[Colorized visualization of the target object and the interaction surface chosen]

For the anchor object, you chose the surface visualized below:

[Colorized visualization of the anchor object and the interaction surface chosen]

What approximation of the parameters make the most sense, given the surfaces
    that you've chosen, and the constraint you've chosen to enforce?
Return a json with each value specified in the json. Begin the json with ```
    json.
```

Figure 27. Prompt for estimating continuous parameters

```
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. distance constraint:
   1) near, object: near to the other object, but with some distance, 50cm <
      distance < 150cm.
   2) far, object: far away from the other object, distance >= 150cm.
2. position constraint:
   1) in front of, object: in front of another object.
   2) around, object: around another object, usually used for chairs.
   3) side of, object: on the side (left or right) of another object.
   4) left of, object: to the left of another object.
   5) right of, object: to the right of another object.
   6) behind of, object: behind another object.
   7) in front of, object: in front of another object.
   8) ontop of, object: on top of another object.
3. direction constraint:
   1) face to, object: facing another object.
4. alignment constraint:
   1) center aligned top, object: align the center of the object with the
      center of the TOP of another object.
   2) center aligned front, object: align the center of the object with the
      center of the FRONT of another object.
   3) center aligned side, object: align the center of the object with the
      center of the SIDE of another object.
```

Figure 28. Prompt for Holodeck (Part 1) – For part 2, see Figure 29.

```
For each object, you can select various numbers of constraints and any
   combinations of them and the output format must be:
object | constraint 1 | constraint 2 | ...
For example:
coffee table-0 | near, sofa-0 | in front of, sofa-0 | center aligned front,
   sofa-0 | face to, sofa-0
tv stand-0 | far, coffee table-0 | in front of, coffee table-0 | center aligned
    front, coffee table-0 | face to, coffee table-0
desk-0 | far, tv stand-0
chair-0 | in front of, desk-0 | near, desk-0 | center aligned front, desk-0 |
   face to, desk-0
floot lamp-0 | near, chair-0 | side of, chair-0

Here are some guidelines for you:
1. The objects of the *same type* are usually *aligned*.
2. When handling chairs, you should use the around position constraint. Chairs
   must be placed near to the table/desk and face to the table/desk.

In the above examples, "coffe table-0", "sofa-0", "tv stand-0", "desk-0", "
   chair-0", "floot lamp-0" are all object IDS
in the scene. In reality, the object IDs look more like "object-0", "object-1",
    ...
i.e. object-1 | in front of, object-2 | near, object-2 | face to, object-3

Here is a list of preexisting objects in the scene, in the format [object_id]:
   [object_description]
[Descriptions and labels of preexisting objects in the scene]

Here is the object that I want to place in the room (object id of it is object-
   target):
[Description of transformable object]
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 the object.
```

Figure 29. Prompt for Holodeck (Part 2) – For Part 1, see Figure 28.

```
Instruction: synthesize the 3D layout of an indoor scene. The generated 3D
   layout should follow the CSS style, where each line starts with the
   furniture category and is followed by the 3D size, orientation and absolute
   position. Formally, each line should follow the template:
FURNITURE {length: ?cm; width: ?cm; height: ?cm; left: ?cm; top: ?cm; depth: ?
   cm; orientation: ? degrees;}
All values are in cm but the orientation angle is in degrees.

Here are the info of other objects within the scene:
[List of objects inside the scene, colon separated from
their width, height, left, top, depth and orientation information.]

Generate a line for an object described by the following:
[Description of transformable object]
```

Figure 30. Prompt for LayoutGPT