–Supplementary Material–
ScanEdit: Hierarchically-Guided Functional 3D Scan Editing

Mohamed El Amine Boudjoghra[1] Ivan Laptev[2] Angela Dai[1]

[1]Technical University of Munich
[2]Mohamed Bin Zayed University of Artificial Intelligence

1. Completing missing geometry

Figure 1. Missing geometry completion results

To complete missing geometry after editing the scene, we use a heuristic plane-based method. Our goal is to keep the region empty after removing objects, unlike 2D inpainting approaches like Infusion [1], which tend to hallucinate new content in the edited areas. Instead of inpainting, we assume that removed objects rest on a support surface (e.g., table or floor). We project the object's points onto this surface and use Delaunay triangulation estimate mesh faces for the projected points. This helps preserve continuity of the support plane and adjacent walls, avoiding visual holes in the reconstructed scene.

Let $\mathcal{P}_{\text{obj}}$ be the set of 3D points belonging to the object, and π_{sup} the support plane. We project each point $\mathbf{p} \in \mathcal{P}_{\text{obj}}$ onto π_{sup} by computing:

$$\mathbf{p}_{\text{proj}} = \mathbf{p} - (\mathbf{n}^\top(\mathbf{p} - \mathbf{c}))\mathbf{n}$$

where $\mathbf{n}$ is the normal of π_{sup}, and $\mathbf{c}$ is a point on the plane. The set of projected points is then triangulated using Delaunay triangulation to reconstruct the surface geometry.

For color prediction, we adopt a mirror reflection strategy based on scene geometry rather than the plane. Specifically, for each point $\mathbf{q}$ on the completed geometry, we first find its nearest neighbor $\mathbf{p}_{\text{nn}} \in \mathcal{P}$ in the original scene:

$$\mathbf{p}_{\text{nn}} = \arg\min_{\mathbf{p}\in\mathcal{P}} \|\mathbf{q} - \mathbf{p}\|_2$$

We then reflect $\mathbf{q}$ across $\mathbf{p}_{\text{nn}}$ to obtain a mirrored location $\mathbf{q}'$:

$$\mathbf{q}' = 2\mathbf{p}_{\text{nn}} - \mathbf{q}$$

Finally, we assign to $\mathbf{q}$ the color of the nearest point to $\mathbf{q}'$ in the original point cloud:

$$\text{Color}(\mathbf{q}) = \text{Color}\left(\arg\min_{\mathbf{p}\in\mathcal{P}} \|\mathbf{p} - \mathbf{q}'\|_2\right)$$

where $\mathcal{P}$ is the set of all original scene points with known color values.

This approach allows us to generate plausible surface completions and assign color in a geometry-aware way, avoiding artificial content and ensuring visual consistency with the original scene.

In order to filter out noise and get a smooth color for the completed region, we apply a median filter.

2. Results with machine generated class agnostic masks

We use predicted masks from Mask3D [2], trained on the ScanNet++ training set. Figure 2 shows results on two scenes: one from the ScanNet++ validation set and the other from the Replica dataset. We start by predicting instance masks using Mask3D and apply non-maximum suppression (NMS) to remove overlapping ones. Then, we run connected components on the remaining parts of the 3D scene to extract other instances that Mask3D fail to detect. For walls and floors, we use plane segmentation—identifying planar regions in the point cloud. The floor is selected as the plane with the highest upward-facing normal (along the z-axis) with lowest height, and the ceiling as the one with the lowest z orientation and highest height, while walls are selected as the planes perpendicular to the floor. To annotate the masks, we render 2D masks onto the undistorted DSLR images and use LLAMA Vision to predict node attributes like class name, color, and so on.

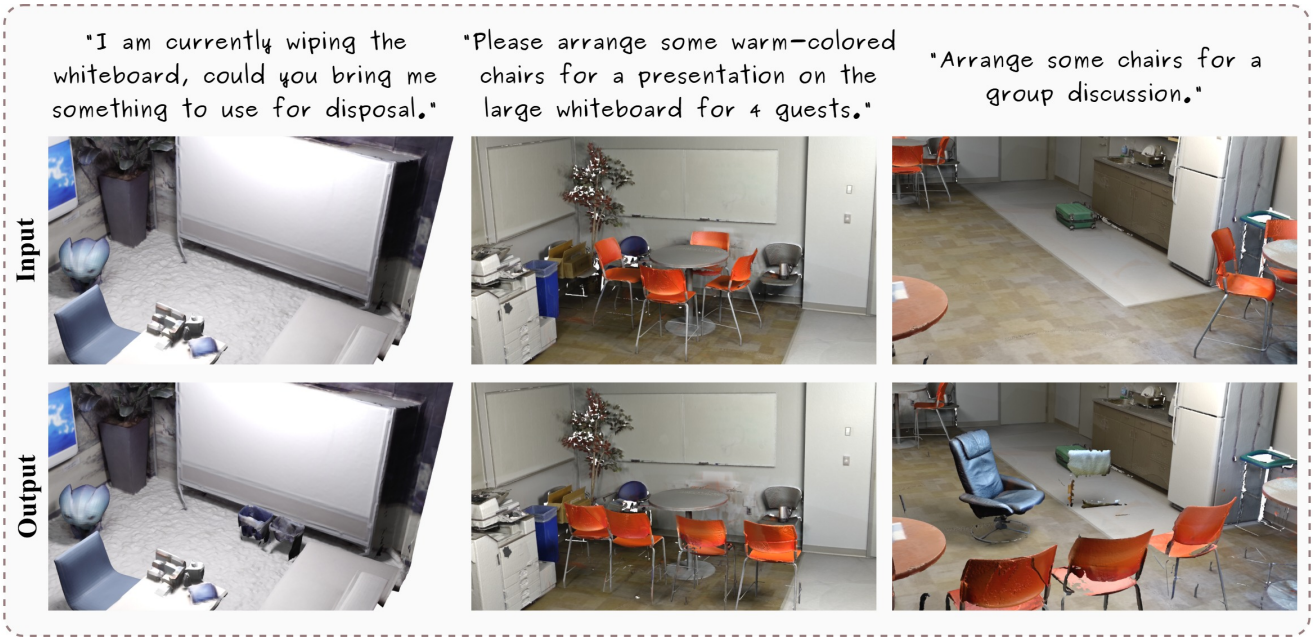

Figure 2. Editing results of our method with instance segmentation masks generated by Mask3D [2], and trained on ScanNet++ [4] training set. We show the results on two scenes, one from Replica [3] and ScanNet++ [4] validation set.

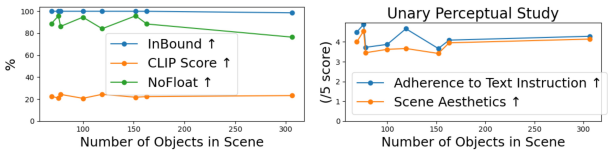

Figure 3. **Scene complexity study.** This figure shows how our method performs across 8 scenes with varying numbers of objects, ranging from 69 to 306. Our approach consistently maintains high performance on both geometric and semantic metrics, even as scene complexity increases.

3. Effect of maximum number of target locations on the performance

Table 1 shows the effect of limiting the maximum number of target locations considered by the planner when transforming an object. The results indicate that the best performance is achieved when the LLM is allowed to dynamically determine the number of target locations based on the structure of the target graph—such as relevant support objects or available support surfaces.

Table 1. Ablation over # of samples; 50% subset evaluation

Samples ↓	NoFloat (%)↑	InBound (%)↑	ColVol (m^3)↓	PIoU ↓
1	77.94	94.55	0.1681	0.34
3	**85.91**	96.27	0.1611	0.33
5	79.88	93.14	0.1766	0.36
LLM controlled (Ours)	83.56	**99.48**	**0.1528**	**0.32**

4. Additional Perceptual Study Details

In our perceptual study, we conducted both a binary and a unary perceptual evaluation to assess the quality of edited 3D scenes. The unary perceptual study required participants to score each generated scene on two key criteria: **Adherence to Instruction** and **Layout Quality**. Participants were presented with a single edited 3D scene and asked to rate it on a scale from 1 (**Strongly Disagree**) to 5 (**Strongly Agree**) for each criterion. **Adherence to Instruction** evaluates how well the edited scene aligns with the given text instruction, ensuring that modifications accurately reflect the specified changes. **Layout Quality** assesses the overall spatial arrangement and positioning of objects, considering factors such as coherence, realism, and usability within the scene. This evaluation provides a fine-grained assessment of different methods' ability to generate high-quality and instruction-consistent scene modifications.

5. Geometric Evaluation Metric Details

5.1. Collision metric (ColVol)

First, we construct a bounding volume hierarchy with depth of 8 for each object in the scene to approximate its shape using bounding boxes at multiple levels (We show in Fig. 6 the visualization of bounding volumes hierarchy at different levels). We estimate the colliding volumes between pairs of objects using the sum of intersections of bounding volumes at level 8 in the bounding volume hierarchy.

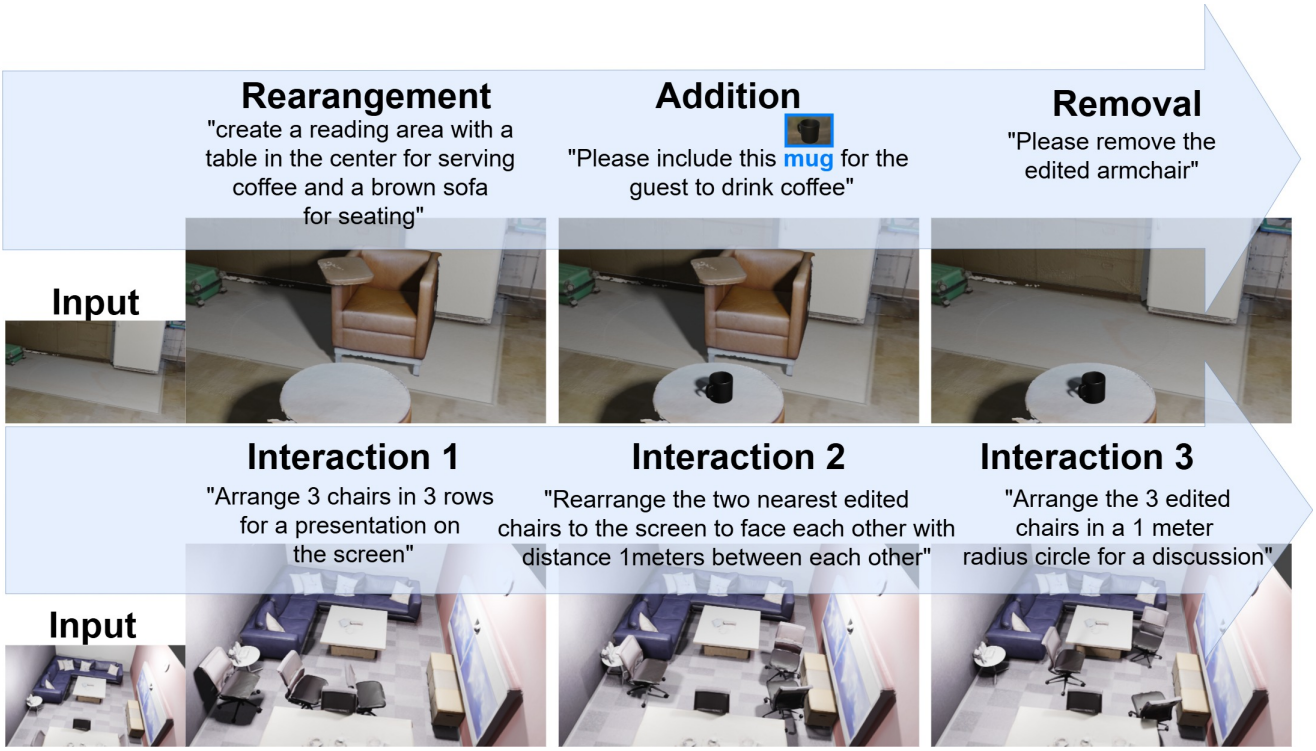

Figure 4. We show in this figure that our method naturally supports interactive editing. Information about the edited object can be passed in as context along with the updated scene graph to help refine the output further. We also demonstrate support for adding new objects: given an asset to insert, we scale it to match the typical height of its category (as suggested by an LLM). For object removal, the LLM is instructed to flag the corresponding node in the graph during the planning stage.

5.2. In boundry metric (InBound)

For all objects that are moved in the edited version of the scene, we report the percentage of points which are inside the flour contour. For this, we compute the signed distance using the floor contour points and floor contour normals.

5.3. Percentage of non floating objects (NoFloat)

For each object that has been moved, we check if it is supported by a support surface or not with 1cm threshlod. An object is considered floating if it is elevated more than 1cm distance from the nearest support surface. In NoFloat metric we report the percentage of objects that are not floating.

6. Estimating graph edges with 3D heuristics

6.1. Estimating 'on top of' support surface relation

We assign each object in the graph nodes N to the closest support surface, provided that the difference between the object's min-imum height and the surface is less than 5 cm.

6.2. Estimating 'against wall' relation

An object o_i is defined as against a wall o_j if its front normal $\vec{F}_i$ aligns with the front normal $\vec{F}_j$ of the wall and the minimum distance between the wall and the object is less than 5cm.

6.3. Estimating 'facing' relation

An object o_i is defined to be 'facing' another object o_j if the front normal $\vec{F}_i$ of o_i is aligned with the directional vector $\vec{d} = c_{o_j} - c_{o_i}$, where c_{o_i} is the center of object o_i.

7. Subgraph identification

In the subgraph identification phase, we use an LLM, Φ, to reduce the set of objects to only relevant classes using prompt 7.1. Then, within these selected classes, we retrieve relevant nodes based on attributes like color, material, and description using prompt 7.2. Next, we identify the key edges with prompt 7.3 that correspond to spatial instructions—for example, an instruction like "organize the top of the cabinet" refers to the 'on top of' edge, with the cabinet serving as its reference node.

7.1. Class pruning prompt

Figure 5. Additional results with ground truth masks on additional scenes. We follow the editing by our planar geometry completion method.

Figure 6. We show in this figure the bounding volumes of the instance object table at different levels in the bounding volume hierarchy BVH. We compute the colliding volumes between objects at level 8 as the sum of intersections in cube meters m^3.

```
Relevant class selection prompt:

You will be given a list of class names and your
    role is to locate the classes of interest.
    The classes of interest are objects to be
    moved and potential target locations. The
    classes don't have to be explicitly mentioned
     in the prompt; you must infer them from the
    context, especially the target locations
    where objects should be moved.

Among the following classes <objects>
    list_of_class_names</objects>, which ones are
     relevant to "{instruction}"?

Please give only the necessary class names to
    execute the task and make sure to return them
     in the tag:
<objects>[place list of objects here]</objects>.

Tips:
- Relevant classes do not necessarily need to be
    explicitly mentioned in the instruction. You
    should analyze the instruction and determine
    the most logical target location based on its
     context. For example, if the instruction is
    to "clear the inside of the fridge," and the
    class list includes a kitchen counter, it
    should also be considered. The kitchen
    counter is the most logical place to place
    the items, as it fits the context of the task
    .
- You must consider only class requirements; don'
    t try to look for objects based on spatial
    relations like "close to" or "on top of." For
     example, "empty the inside of the fridge"
    means that the relevant classes are fridge
    and a potential location where to place the
    items, like a kitchen counter or table.
- Your role is to identify the most relevant
    classes based solely on semantics, without
    considering spatial relationships. For
    example, in the instruction "Move objects
    from inside the cabinet," your focus should
    be on the cabinet and a suitable target
    surface for placing the objects, such as a
    table or the floor. Do not attempt to
    determine which specific objects are inside
    the cabinet from the given list.

Important:
- The classes you return should be in the
    provided list of class objects; don't
    hallucinate any new class names.
- For every object class to be moved, you must
    return the most logical target class where it
     is going to be placed relative to either "
    close to" or "on top of" one of its support
    surfaces.



Examples of objects relevent to instruction:
{Examples}
```

```
Very important: return classes from the provided
    list, don't hallucinte classes outside of it.
     If the instruction indicates prular objects
    you ust retrieve the closest from the set  <
    objects>{list_of_class_names}</objects>.

- The final output must be in XML format:
```

7.2. Instance retrieval with node attributes

```
You are a helpful assistant responsible for
    filtering out irrelevant objects based on
    some color, material, or size that might be
    mentioned in a given instruction. You will
    receive a list of objects, each with an ID,
    color, and material. Your task is to identify
     and return the object IDs that align with
    the instruction. If the instruction does not
    specify any attributes for the desired
    objects, return all object IDs.

Additionally, some objects serve as target
    locations for placing other objects according
     to the instruction. If no specific
    attributes are mentioned for these target
    objects, they should be retained.
If no color or material are requested in the
    instructions return all IDs.


Note: please select object that need to be moved
    as well as target objects.
Objects list with their object ids, class name,
    list of major colors, list of materials:

{Objects_details}

Instruction is :
{Input_instruction}

Please end your thinking with <relevant_ids>[
    place the list of ids which align with the
    instruction here]</relevant_ids>

Tips:
    - Relevant classes do not necessarily need to
         be explicitly mentioned in the
        instruction. You should analyze the
        instruction and determine the most
        logical target location based on its
        context. For example, if the instruction
        is to "clear the inside of the fridge,"
        and the class list includes a kitchen
        counter, it should also be considered.
        The kitchen counter is the most logical
        place to place the items, as it fits the
        context of the task.
    - You must consider only semantic attributes
        like color, material, and specific object
         characteristics as filtering
        requirements, don't try to look for
        objects based on spatial relations like
        close to or on top of. For example 'empty
         the inside of the fridge' means that the
         relevent classes are fridge and
        potential location where to place the
        items like kitchen counter or table.

Very Important:
    - Don't filter any object out unless a color
        or material attribute is requested.
    - Your task is to return the IDs of objects
        strictly from the provided list. For each
         object selected, you must include at
        least one target object ID from the list
        to indicate its placement relative to
        another object (e.g., near the door, on
        the table, or on the floor).

Examples of objects relevent to instruction:
    {Examples}
```

7.3. Identifying relevent edges

```
You are a Helpful Assistant. Your task is to
    determine how an object should be retrieved
    based on the given instructions. Do not try
    to make a plan for placing the objects. You
    must focus only on understanding whether
    retrieval involves relations or not.
    Guideline for Identifying the Object to Move
        and Retrieval Type

        Identify the Object to Move
            Look for action verbs like move,
                place, put to determine what is
                being acted upon.
            Example: "Move the chair." (Chair is
                the object to move).

        Check for Relations
            If a prepositional phrase (e.g., next
                 to the table) describes the
                destination, it is retrieval
                without relations ("Move the
                chair next to the table.").
            If the phrase describes the current
                position, it is retrieval with
                relations ("Move the chair that
                is next to the table.").

        Determine Retrieval Type
            Without Relations: The object is
                retrieved by intrinsic properties
                 (e.g., color, type).
            With Relations: The object is
                retrieved using another object as
                 a reference.

        Edge Cases
            If no reference is mentioned, assume
                retrieval without relations.
            Multiple objects should be checked
                for independent or dependent
                relationships.

    Your role is to analyze retrieval type, not
        to decide where to place objects.

You will receive a list of objects, each with
    details such as their IDs, colors, and
```

```
    materials. The instruction provided may ask
    for objects relative to others in this list.
    Your task is to analyze the instruction and
    identify the objects that can serve as
    references to locate the target objects.

For example:
    If the instruction is: "Could you please
        empty the kitchen counter, then move
        items from the fridge to the kitchen
        counter?", the fridge has a relation "on
        top" with all possible surfaces, and the
        kitchen counter is also referenced as "on
         top." In this case, return the relevant
        object IDs based on these relationships.

Objects list with their object ids, class name,
    list of major colors, list of materials:

{Objects_details}

Instruction is :
{Input_instruction}

Please end your thinking with the following xm
    format
'''xml
<objects>
    <object>
        <id>[please the object id here]</id>
        <surface_ids>[please place the desired
            surface ids here seperated by a comma
            ]</surface_ids>
        <relation>[please place the required
            relations here seperated by comma,
            you must choose from 'on top', '
            facing', 'against wall']</relation>
    </object>
</objects>

            '''

Important regarding surface IDs: A surface is the
     area where objects can be placed in an
    object, each object has surfaces with several
     IDs from 0 and up, the surface with the
    lowest elevation has the highest ID.
Example for surfaces: if there are three surfaces
     with ids 0,1,2,3 the lowest id corresponds
    to the top surface in this case ID 0 where
    the highest ID corresponded to the lowest
    surface in this case 3.

Examples that can help you understanding if the
    instruction requires retrieving with spatial
    constraints or not:
    {Examples}

Hint how to address this task assigned to you:
    First, evaluate the instruction to determine
    whether any objects need to be retrieved in
    relation to others. If no objects are to be
    retrieved in relation to others, return an
    empty XML tag as follows: xml<objects></
    objects>.
```

8. Prompts for planner

In the planning phase, we generate first a plan where the LLM Ψ generates a detailed plan while considering different target locations which define the hypotheses for moving each object, then it selects the best one while taking into account physical plausibility in support surfaces (max height does not accommodate the object to be moved or the surface is full).

8.1. Prompt for generating a plan

```
You will be given a list of objects with their
    ids colors and materials, you have to suggest
     a plan on how to place objects to execute
    the instruction.

Important analysis before planning how objects
    should be moved:
    Analyze the instruction and keep the movement
         of objects to the minimum, while
        insuring the instruction is satisfied.
        For example if the goal is to create a
        seating area to whatch TV, the TV should
        not be moved.


Objects list with their object ids, class name,
    list of major colors, list of materials:

{Objects_details}

Instruction is :
{Input_instruction}

Important: enphasis on objects that should not be
     moved in the plan, an example is " place the
     chairs to watch TV" the TV should remain
    untouched and the chairs should be placed in
    front of it while facing it.
Please end your thinking with <placement_plan>[
    place your detailed plan by specifying object
     ids and class names here]</placement_plan>
Important: enphasis on objects that should not be
     moved in the plan, an example is " place the
     chairs to watch TV" "arrange chairs for
    presentation on a screen" the TV and screen
    should remain untouched and the chairs should
     be placed in front of it while facing it.
{Support_surfaces_details}

Important when suggesting the plan:
    - please suggest a target location for every
        object you want to move relative to other
         objects or the floor, even if the
        instruction is vague.
        For example if an instruction says to
            clear a table, you must identify what
             objects are on the table and suggest
             new locations for these objects
    - First analyze all potential target location
        , and place object relative to the most
        logical targets. Example, an instruction
        'move the chair' has multiple targets but
         the most logical one is close to a table
```

```
        .
    - If the target location is support object,
        please make sure to suggest which surface
         among the object surfaces it should go
        to in the plan
    - Some objects are better stacked on each
        other, for example papers and boxes
        shoulkd be stacked on each other if they
        are to be organized. where the largest
        one with dx,dy is the first.
    - The object level instruction must refer to
        an object (if desired to be placed close
        to it) or one of its surfaces (if desired
         to be placed on a surface) that exists
        in the list of objects.
    - Make the plan with natural language only,
        don't suggest to place objects in
        specific coordinates.
    - It is very important to pay attension to
        objects coordinates, since there are
        multiple objects with same functionality
        but different sizes. e.g. a large plant
        cannot be placed on top of a table, but a
         midsized or small plant one can be which
         gives a better vibe to the space.

When should you stack objects on top of each
    other:
    - Some objects like papers or boxes are
        better stacked on top of each other, in
        this case the plan should be:
        - place largest box on some surface 0 of
            another object or floor.
        - place second smallest box on the placed
             box.
        - and so on ...
    - If three objects a (largest),b(smallest),c(
        medium) are to be stacked on top of each
        other a should be on a surface floor, or
        table, etc as it is the base of the stack
        , c should be on surface ID 0 of a, and b
         should be on surface ID 0 of c.

When to use the coordinates:
    - Each objects has x,y coordinates use them
        to figure out far or close objects if
        mentioned in the instruction. It is
        strict to not include any coorcinates in
        the final placement plan.

How to handle instructions with few details:
    - Try to make a plan that can be physically
        plausible, for example placing 10 sofas
        for a presentation can not be done in one
         row, you have to place them in multiple
        rows in this case 3 rows would be good.
```

8.2. Prompt for converting the plan into hierarchical graph

```
You will be given a list of objects with their
    ids colors and materials and a plan to place
    these object. Your role is to return the
    logical dependency between objects and format
     the placement of objects in a hierarchical
    manner starting from the floor. Please
    strictly follow the placement plan

Important analysis before planning how objects
    should be moved:
    Analyze the instruction and keep the movement
         of objects to the minimum, while
        insuring the instruction is satisfied,
        pay attention to the placement plan, some
         objects are best to be kept unchouched
        you need to reflect that in the object
        level instruction in the hierarchy. For
        example if the goal is to create a
        seating area to whatch TV, the TV should
        not be moved. In this case the TV will
        nest the objects for seating, while
        having the instruction "keep the TV
        untouched".


Placement plan: 'place the table with id 10 near
    the door id 12 and a bottle with id 1 on top
    of the table (surface ID 0 of the table) with
     id 10, and the chair id 50 facing the table
    .'

Hierarchical Structure for Object Placement

    The hierarchy follows a nested dependency
        model, where objects are placed relative
        to their parent objects. This ensures
        spatial constraints are logically
        maintained.
    1. Root Level (Floor)

        The floor is the base of the environment,
             meaning all objects are ultimately
            placed on it.
        The floor itself remains static and
            untouched, serving as the
            foundational layer for all placements
            .

    2. First Nested Level (Door)

        The door (ID 12) is placed directly on
            the floor, meaning it is positioned
            independently.
        Since the door is static like the floor,
            it does not move or act as a
            container for other objects.

    3. Second Nested Level (Table)

        The table (ID 10) is placed near the door
             (ID 12).
        This means the table's position is
            spatially related to the door but not
             contained within it.
        Since the table is a movable object, its
            placement depends on the door's
            position.

    4. Third Nested Level (Chair & Surface)

        The chair (ID 50) is placed facing the
            table (ID 10), making it dependent on
```

```
             the table for its orientation.
        The table s surface (ID 0) is an implicit
             subcomponent of the table and serves
             as a placement area for smaller
            objects.
        While the surface is not a separate
            object, it acts as a reference point
            for placing items on the table.

    5. Fourth Nested Level (Bottle)

        The bottle (ID 1) is placed on top of the
             table (specifically, surface ID 0).
        Since the surface belongs to the table,
            the bottle is indirectly dependent on
             the table s placement.

    Purpose of the Hierarchy

    The structure enforces a logical dependency
        between objects. For example:
        The bottle s placement depends on the
            table.
        The table s placement depends on the door
            .
        The chair and surface placement depends
            on the table.
        The door s placement depends on the floor
            .

    This hierarchical representation reflects
        real-world relationships and ensures
        clarity in placement instructions.


How to approach this:
- You need to figure out the group center and
    what objects but be placed relative to the
    group centers.
- The group centers are placed relative to the
    floor, while the group memebers are placed
    relative to the group centers.
- If an object is placed in relation to other
    object it should be nested in it.


Final format (You must follow this output format)
    :
xml ```
    <object>
        <name>floor</name>
        <id>floor_id</id>
        <instruction>leave the floor untouched as
             floors cannot be moved</instruction>
        <object>
            <name>door</name>
            <id>12</id>
            <instruction>leave the door untouched
                </instruction>
            <object>
                <name>table</name>
                <id>10</id>
                <instruction>place the table
                    close to the door with id
                    12</instruction>
                <object>
                    <name>surface</name>
                    <id>0</id>
                    <instruction>Leave surface ID
                         0 untouched as it is
                        part of the table</
                        instruction>
                        <object>
                            <name>bottle</name>
                            <id>1</id>
                            <instruction>place
                                the bottle on top
                                 of the surface
                                with ID 0</
                                instruction>
                        </object>
                </object>
                <object>
                    <name>chair</name>
                    <id>50</id>
                    <instruction>place the chair
                        id 50 in front of the
                        table, facing the table</
                        instruction>

                </object>

            </object>
        </object>
    </object>
```


Extremely Important for the nesting:
    Even if an object doesn't move it must nest
        related objects.

Now please proceed with the following:

Placement plan :
{**placement_plan**}

Denependency plan:
{**dependency_plan**}

Structured xml hierarchy:
Please proceed with the step by step thinking
    then end it with xml output here
Important notes:
    - If an object is placed in relation to other
         object (on top of, close to, near etc)
        it should be directly nested in it.
    - Please nest the desired surface in its
        correspoding object parent if exists in
        the placement plan.
    - An object is nested only if it should
        remain untouched or placed relative to
        the parent, but not moved from the parent
        .
                Example when not to nest: the cup
                     should not be nested in the
                    fridge in 'move the cup from
                    the fridge'
                Example when to nest: the cup
                    should be nested in surface
                    id 1 and surface id 1 should
                    be nested in the fridge in '
                    place the cup in surface 1 in
```
```

```
                    the fridge'

Important regarding object level instructions:
    - Add the word 'untouched' in the instruction
         if the object is a surface or should not
         be moved. Surface instruction should
        always be 'leave surface untouched'

Very important:
    - You must return the structure that enforces
         a logical dependency (use the dependancy
         plan in order to figure out which object
         is nested in which) between objects to
        figure out which object relates to which
        object before generating the xml nests
    - If an object is not explecitly mentioned to
         be moved in the placement plan you
        should leave it untouched, you your role
        is to structure the placement plan in a
        nested structure following the dependency
         plan


    Extremely important note: You must move only
        the objects that are mentioned to be
        moved in the instruction. For example '
        move glass to the fridge' means the
        fridge must stay in place
```

9. Prompt for hierarchical object placement

```
You will be given a reference object and a list
    of objects that you need to place relative to
     a reference object. Your role is to think
    step by step and seggest new locations,
    orientations, and constraints for the list of
     objects.
    Each object has to be placed following its
        instruction, you must suggest 3D
        coordinate for the base of the object, an
         orientation of the object, and a list of
         constraints with respect to the
        reference object.

    The representation of the reference object:
        - The reference object is represented
            with
            1. Its base coordinate, which
                represents the 3D coordinate of
                the object with the minimum
                elevation (z) in meters and
                center in x and y.
            2. Its dimensions which represent the
                 height(following the z axis),
                the width(following the x axis),
                the depth(following the y axis)
                in meters
            3. Its orientation, which refers to
                the orientation of the object
                around the z axis, in degrees.
            4. Its surfaces which can be used for
                 placing objects, each surface
                has an ID where id 0 represents
                the surface with the highest
                elevation, the elevation is in
                meters.
            5. List of objects that are on top of
                 the object

    The representation of the List of object to
        be placed relative to the reference
        object {Parent_object_name} id {
        Parent_object_id}:
            1. its base coordinate, which
                represents the 3D coordinate of
                the object with the minimum
                elevation (z) in meters and
                center in x=0 and y=0.
            2. Its orientation, which refers to
                the orientation of the object
                around the z axis, in degrees.

    The possible list of constraints with respect
         to the reference object {
        Parent_object_name} id {Parent_object_id}
         are:
        - in_surface : this constraints concerns
            only instructions that require
            placing objects inside or on top of a
             reference object, if the instruction
             requires placing an object on top,
            that means the surface ID is 0 since
            it is the one with the highest
            elevation
        - facing: This constraint concerns
            objects which should be facing the
            reference object in a natural setting
            , for example a chair should be
            facing a referebce object whiteboaard
            .


    Important details when reporting the final
        location and orientation:
        - Don't perform the math operation
            instead report the formula with
            values and operations to get the new
            location or orientation. The allowed
            operations are :*: for multiplication
            , /: for division, -: for minus, +:
            for summation, cos: for cosine
            function,  sin: for sin function.
            angles should be in degrees.

    Hint on relations between objects:
        - If an object is facing another, its
            orientation should be 180 degrees
            minus the orientation of the other
            object
        - if an object is facing the same
            direction as another, both should
            have the same orientation

    Important details on the reference object's {
        Parent_object_name} id {Parent_object_id}
         orientation and location:
        - the reference object is oriented
            towards (meaning its front) the
            positive direction of the x-axis. And
             its base coordinate is located near
            the origin.
```

Output format:
Please end your step by step thinking with the following xml which summarizes the new locations , new orientations and list of constraints of objects with respect to the reference object relative to the reference object {**Parent_object_name**} id {**Parent_object_id**}:

```
xml ```
<objects>
    <object>
        <id>[please place the object ID here ]<id>
        <name>[please place the object name here]<name>
        <new_base_coordinate>[please place here the new base coordinate of the object]</new_base_coordinate>
        <new_orientation>[please place here the new orientation the object]</new_orientation>
        <constraints>
            <facing>[please mention here wheather the object should be facing the reference object {Parent_object_name} id {Parent_object_id} or not, answer with yes or no]<facing>
            <in_surface>[If the object should be in one of the surfaces place the surface ID here, if not place None]<in_surface>
            <distanace_to_reference>[please place the distance to the reference object here]<distanace_to_reference>
        </constraints>
    <object>
<objects>

```
```

Reference Object:
{**parent_object_details**}

List of Objects to be placed with their instructions:
{**objects_to_be_placed**}

{**floor_details**}

Now please proceed with your suggested new coordinates and orientations, taking into account that the reference object x center and y center are both 0,0 and this is similar to its base coordinate. It is also important to note that all objects are represented with base coordinate which is the center of the object in x and y and minimum elevation in z.

Important: the front of the reference object is in the positive x axis, thus if an object should be in front of it it must be placed at x>0 and y that is bounded.

Important: you must use place the objects elevation (z axis) at the floor level if the object should be on the floor.
Important regarding in_surface constraint: if the reference object is a surface of an object, please put the ID of the surface (which is a number, don't put something like fridge's surface or table's surface) in this constraint.

Very important: Pay attention to the instruction for each object, if an object should not be moved please don't include it in the xml list. If no object should be moved return an empty xml <objects></objects>

Left, right, front, and back sides conventions:
Left side with respect to the reference object is : y < 0
Right side with respect to the reference object is : y > 0
Front side with respect to the reference object is : x > 0
Back side with respect to the reference object is : x < 0

Very important when placing objects against walls or other objects:
If an object is placed against a wall or another object <facing> should be no and the orientation of the object to be place should be the same as the reference object
If an object is against the wall, its x should be half the dx dimension of the object and y can range from -dy_wall/2 to dy_wall/2

Very important for the <facing> constraint which are with respect to the reference object {**Parent_object_name**} id {**Parent_object_id**}:
If an object should be facing the same direction as the reference object {**Parent_object_name**} id {**Parent_object_id**} <facing> constraint should be set to no
Example (helpful for reasoning only):
- place chairs with respect to another reference object chair for a role that requires them to be in the same direction like guest watching TV, means that <facing> is no
- place chairs with respect to another reference object TV for a role that requires them to be in the same direction like guest watching TV, means that <facing> is yes
```

References

[1] Zhiheng Liu, Hao Ouyang, Qiuyu Wang, Ka Leong Cheng, Jie Xiao, Kai Zhu, Nan Xue, Yu Liu, Yujun Shen, and Yang Cao. Infusion: Inpainting 3d gaussians via learning depth completion from diffusion prior. *arXiv preprint arXiv:2404.11613*, 2024. 1

[2] Jonas Schult, Francis Engelmann, Alexander Hermans, Or Litany, Siyu Tang, and Bastian Leibe. Mask3d: Mask transformer for 3d semantic instance segmentation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8216–8223. IEEE, 2023. 1, 2

[3] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 2

[4] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 2