Edit-As-Act: Goal-Regressive Planning for Open-Vocabulary 3D Indoor Scene Editing

Supplementary Material

1. Overview

This supplementary document provides additional details, experimental results, and analyses for our paper, "Edit-As-Act". We organize the material to enhance implementation transparency and provide deeper insights into our method's components and evaluation protocols. The contents are as follows:

- **Sec. 2: Extended Intuition.** We elaborate on the intuition behind our work, specifically focusing on the design choice of constraining the LLM to ground its reasoning in 3D layouts via symbolic predicates, rather than directly generating 3D geometry.
- **Sec. 3: Object Generation and Stylization Pipeline.** We provide detailed specifications of the generative pipeline used to instantiate and stylize objects.
- **Sec. 4: EditLang Formalization.** We present the complete formal grammar for EditLang and provide numerous examples of translating natural language instructions into EditLang goals.
- **Sec. 5: Goal-Regressive Planner Details.** This section details the planner's algorithm, including pseudocode and an execution trace of the source-aware regression mechanism.
- **Sec. 6: Validator Details.** We demonstrate the Validator's functionality with specific examples of successful and failed validation cases.
- **Sec. 7: LVLM Metric Reliability.** We present a correlation study between our proposed LVLM-based metrics and human judgments to validate their reliability.
- **Sec. 8: Extended Quantitative Analysis.** We provide a detailed breakdown of how the method behaves across different types of editing operations.
- **Sec. 9: Failure Case Visualization.** We visualize representative failure cases of our model.
- **Sec. 10: Additional Ablation Studies.** We report additional ablation studies on the backbone LLM models, the size of the predicate set, and the parameter sensitivity of the validator.
- **Sec. 11: Additional Quantitative Experiments.** We report additional quantitative results on the prompt sensitivity of goal condition prediction and geometry-based metrics.
- **Sec. 12: Limitations and Discussion.** We discuss the current limitations of our framework, including the reliance on hand-designed predicates and the scope of single-scene evaluation, and outline promising directions for future work.
- **Sec. 13: Source Scene Visualization.** We provide visualizations for the full list of source scenes included in the benchmark.
- **Sec. 14: Additional Qualitative Results.** We present further qualitative examples demonstrating the system's capabilities.
- **Sec. 15: Full Prompts for Model and Evaluation.** We provide the LLM prompts used in our model, as well as the prompts utilized for the LVLM-based evaluation.

2. Extended Intuition

Core Intuition Summary. Traditional editing approaches treat the task as **Generative Simulation** by attempting to predict the visual appearance of a scene after a change. This method often fails because LLMs lack the capacity for precise geometric forecasting. In contrast, Edit-As-Act redefines editing as **Goal Specification** and focuses on defining the conditions that must be satisfied in the final state. Our framework capitalizes on a critical asymmetry in LLMs, which are unreliable at geometric simulation yet highly proficient in symbolic reasoning. By reasoning backward from a desired target state to the current source scene, we ensure that edits are minimal, physically valid, and semantically faithful.

A central challenge in LLM-based indoor scene editing is interpreting complex 3D environments with interacting objects [5, 16]. Conventional approaches, such as fine-tuning on scene datasets or using iterative generation, often fail in open-vocabulary settings because they rely heavily on predefined object distributions [11, 14, 18]. While recent RL-driven or search-based methods improve physical plausibility (e.g., avoiding collisions) [4, 10, 12, 17], they often address geometric validity at the expense of semantic intent and compositional grounding [7, 13].

Our formulation is motivated by a critical asymmetry observed in frontier LLMs, as illustrated in Fig. 1: they are poor at *geometric simulation* but excellent at *symbolic specification* [3, 9, 15]. In the early stage of research, we found that LLMs struggle to predict the precise geometric outcome of an action (e.g., "What will the scene look like after sliding the chair?"). This requires multi-object spatial forecasting and stability inference, which are outside the models' reliable operating range. Conversely, LLMs are remarkably robust when reasoning about the conditions a fi-

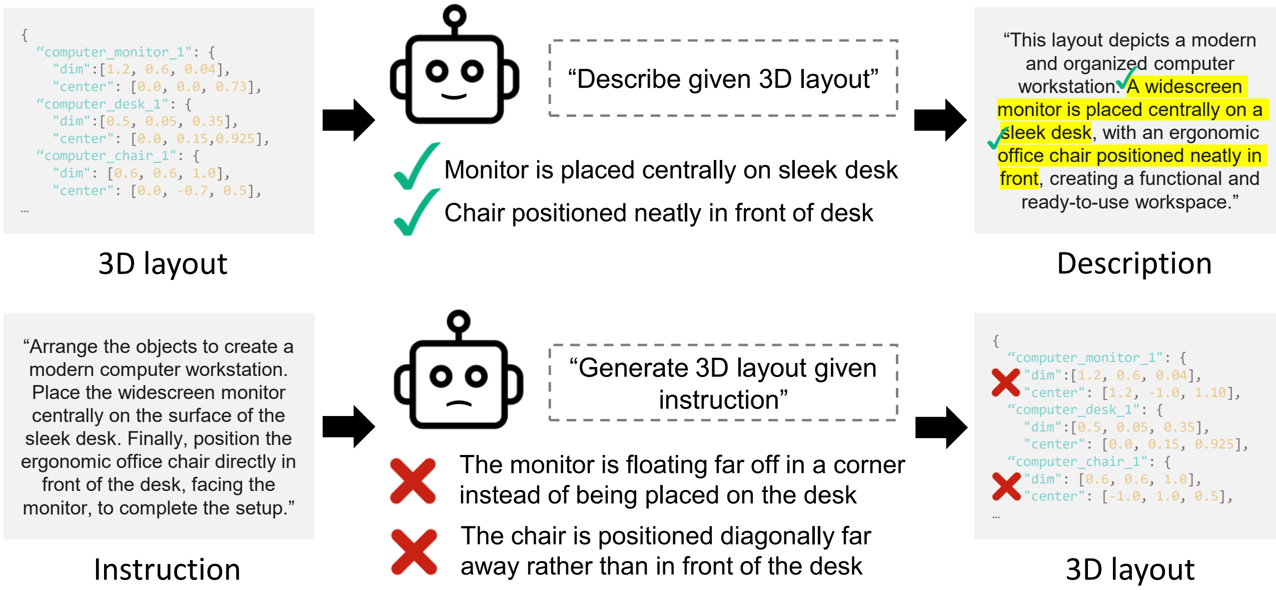

Figure 1. LLMs demonstrate strong capabilities in interpreting existing 3D layouts (top) but remain unreliable when directly generating 3D layouts from instructions (bottom). This asymmetry motivates our goal-regressive formulation.

nal scene must satisfy (e.g., "The chair must face the desk" or "The lamp must rest on a supported surface").

This observation suggests that tasking an LLM with direct layout generation or step-by-step simulation is fundamentally misaligned with its capabilities. Instead, we reformulate 3D scene editing as a **goal-specification problem**. Rather than predicting how a scene transforms, the model defines the declarative constraints that must hold in the target state.

To bridge these symbolic constraints with the 3D environment, we introduce **EditLang**, a PDDL [1, 6]-inspired domain that defines explicit preconditions and effects for geometric actions. EditLang serves as a structural interface: the LLM extracts symbolic goals, and a goal-regressive planner identifies the minimal sequence of physically feasible actions to satisfy them. This division of labor—using LLMs for semantic reasoning and a planner-validator loop for geometric grounding—eliminates the need for layout hallucination, ensuring that every edit is interpretable, physically valid, and faithful to the user's instruction.

3. Object Generation and Stylization Pipeline

To physically realize the editing operations proposed by our planner, specifically `Add` and `Stylize` actions, we employ a unified generative pipeline powered by Hyper3D Gen-2 (Rodin Gen-2) [8]. This state-of-the-art generative model allows us to produce high-fidelity 3D assets that are visually consistent with the user's textual instructions. As illustrated in Fig. 2, our pipeline handles two distinct workflows depending on the editing requirement. Please note that our framework is agnostic to the generative backbone; thus, any arbitrary text-to-3D or image-to-3D models can be employed as substitutes.

3.1. Object Generation

When the planner specifies an `Add` action (e.g., "Add a modern chair"), the system requires a completely new 3D asset. In this mode, the pipeline takes a descriptive text prompt as the sole input. Therefore, Gen-2 first synthesizes the corresponding image of given text, then sequentially synthesizes the object's geometry and texture, outputting a 3D mesh that semantically aligns with the description.

3.2. Object Stylization

For `Stylize` actions (e.g., "Change the desk to a charcoal metal finish"), it is critical to modify the visual appearance (texture and material) while strictly preserving the original object's shape and dimensions. As highlighted by the blue dashed boxes in Fig. 2, we utilized the point cloud conditioning feature of Gen-2. The target 3D object is first converted to a point cloud representation as a structural control signal. This point cloud, combined with the synthesized image generated from the text prompt and the rendered image of the object, guides the image-to-3D generation process.In Fig. 2, by conditioning the generation process on the point cloud representation, the model updates the texture to match the "charcoal metal finish" while ensuring the output 3D object retains the exact pose and structure of the original input.

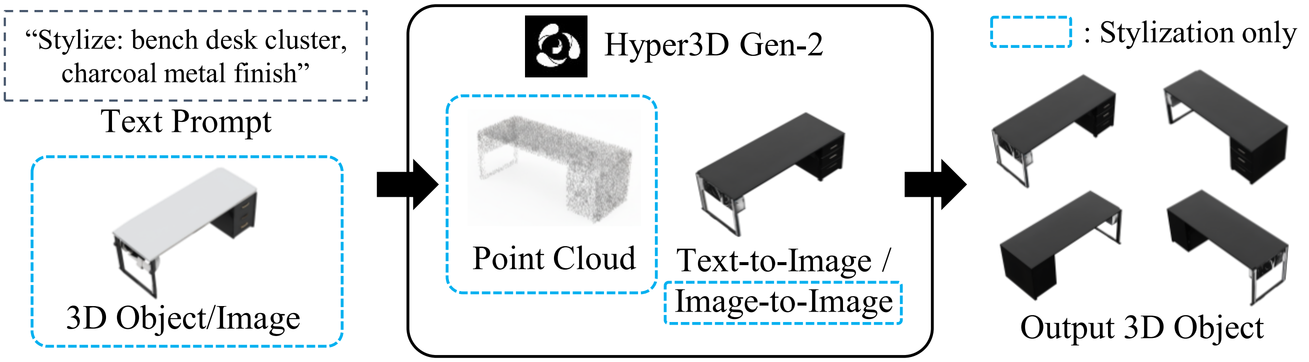

Figure 2. **Overview of the object generation and stylization pipeline.** We utilize Hyper3D Gen-2 [8] to synthesize high-fidelity 3D assets. The pipeline operates in two modes: (1) *Text-to-3D* for generating new objects from scratch, and (2) *Point Cloud-Guided Stylization* (highlighted in blue dashed boxes) where the input 3D object is converted into a point cloud to condition the generation, ensuring the geometric structure remains preserved while the texture is updated according to the text prompt.

4. EditLang Formalization

To bridge the gap between vague natural language instructions and rigid 3D geometric data, we need a structured intermediate representation. **EditLang** serves as this bridge, defining the domain through two core components:

- **Predicates (The "State"):** A vocabulary to describe geometric and semantic relationships (e.g., `on(lamp, table)`). They act as the *"eyes"* of the planner, translating continuous 3D coordinates into discrete symbolic states to verify if a goal is met.
- **Actions (The "Operators"):** A set of atomic operations (e.g., `move_to`, `stylize`) defined with explicit preconditions and effects. These act as the *"hands"* of the planner, ensuring that every modification is physically grounded and logically sound before execution.

This section presents the complete specification of EditLang used in our experiments, supporting hierarchical object manipulation, fine-grained directional placement, and style propagation.

4.1. Syntax

EditLang uses a typed, PDDL-inspired syntax defined as follows:

4.2. Predicate Library

Our predicate set captures geometric state, topology, physical constraints, and semantic relations. We categorize them by the fundamental questions they answer:

- **Existence (Is it there?):** `exists(o)`, `removed(o)`.
- **Spatial Relations (Where is it globally?):** `at(o,pos)`, `on(o,surface)`, `between(o,a,b)`, `near(o,target,`τ`)`, `aligned_with(o,target,axis)`.
- **Directional & Relative (How is it oriented?):** `is_facing(o,target)`, `left_of(o,ref,view)`, `right_of(...)` , `in_front_of(...)` , `behind(...)` .
- **Grouping & Constraints (How does it interact?):** `grouped_with(child,parent)`, `locked(o)`.
- **Physical & Functional (Is it valid?):** `supported(o,surface)`, `contact(o,surface)`, `clear(o)`, `stable(o)`, `colliding(o1,o2)`. `visible(o,view)`, `accessible(o)`.
- **Attributes (What does it look like?):** `has_style(o,desc)`, `matches_style(o1,o2)`, `has_scale(o,sx,sy,sz)`.

4.3. Action Definitions

The planner employs atomic actions defined by $\langle \text{pre}, \text{add}, \text{del} \rangle$ conditions.

- **Spatial Manipulation:**
  - `move_to(o, pos)`: Relocates a single object o.
  - `move_group(parent, pos)`: Moves a parent object (e.g., dining table) and all associated children (e.g., chairs, centerpiece) while preserving their relative local transforms.
  - `place_relative(o, target, relation)`: Places o satisfying directional predicates (e.g., `left_of`) relative to the target.
  - `place_on(o, surface)`: Places o on a support surface.
  - `align_with(o, target, axis)`: Aligns bounding boxes along an axis.
- **Orientation & Scale:**
  - `rotate_towards(o, target)`: Updates yaw to face a target.

– `rotate_by(o, degrees)`: Rotates object o by a specified angle relative to its current orientation.
– `scale(o, sx, sy, sz)`: Modifies dimensions with collision checks.

- **Creation, Deletion & Style:**
  – `add_object(o, cat, support)`: Instantiates a new asset.
  – `remove_object(o)`: Deletes o and clears relations.
  – `stylize(o, desc)`: Updates texture based on text description.

Tab. 1 demonstrates mapping instructions to these extended goals.

5. Goal-Regressive Planner Details

This section details the algorithmic implementation of the planner. We employ the LLM as a policy π_θ to propose transition operators, while the control flow is governed by a deterministic symbolic loop.

5.1. Planning Algorithm

Algorithm 1 outlines the core planning loop. The system maintains a stack of goals G_t. At each iteration, it prompts the LLM to propose an action a_t that satisfies at least one condition in G_t. Crucially, the VALIDATOR acts as a rejection sampler, filtering out geometrically invalid or non-monotonic actions before they affect the plan state.

Algorithm 1 LLM-Driven Goal Regression Loop

Require: Goal predicates G_{target}, Source state S_0
Ensure: Plan $\Pi = [a_1, \ldots, a_T]$
$G \leftarrow G_{target}$
$\Pi_{back} \leftarrow []$ ▷ Backward plan sequence
while $G \neq \emptyset$ **do**
  $success \leftarrow$ False
  for $k \leftarrow 1$ to 3 **do** ▷ Max 3 retries per step
    $a \leftarrow$ LLM_POLICY(G, S_0, Π_{back})
    $valid$, msg $\leftarrow$ VALIDATOR(a, G, S_0)
    if $valid$ **then**
      $success \leftarrow$ True
      break
    else
      Add msg to prompt history (Refinement)
    end if
  end for
  if not $success$ **then**
    return Π_{back}
  end if
  Append a to Π_{back}
  $G \leftarrow$ SOURCEAWAREREGRESS(G, a, S_0)
end while
return Reverse(Π_{back})

5.2. Source-Aware Regression Logic

Unlike classical STRIPS which regresses to an initial empty state, our ‘SourceAwareRegress‘ function filters preconditions against the actual 3D scene geometry S_0. This serves as a pruning mechanism.

The implementation logic is as follows:

1. **Satisfy:** Remove goals satisfied by action effects ($G \setminus \text{add}(a)$).
2. **Propagate:** Identify preconditions required by a ($P = \text{pre}(a)$).
3. **Prune:** Filter out preconditions that are already true in the source scene ($U = P \setminus S_0$).
4. **Update:** The new goal set becomes the remaining goals plus the unsatisfied preconditions ($G \leftarrow (G \setminus \text{add}(a)) \cup U$).

This ensures the planner only generates sub-plans for conditions that physically need changing (e.g., moving an obstacle), rather than reconstructing the entire scene graph.

5.3. Execution Trace Example

To demonstrate the regression logic, consider the instruction: *“Place the lamp on the side table,”* where the table is currently cluttered.

Initial Goal (G_0). {`on(lamp, table)`}

Step 1.

- **LLM Proposal:** $a_1 =$ `place_on(lamp, table)`
- **Preconditions:** {`clear(table)`, `exists(lamp)`}
- **Check S_0:**
  – `exists(lamp)` is True (in inventory).
  – `clear(table)` is **False** (blocked by a mug).
- **Regression:** $G_1 \leftarrow$ {`clear(table)`}

Step 2.

- **LLM Proposal:** $a_2 =$ `move_to(mug, shelf)`
- **Effects:** Adds `clear(table)` (by removing mug from it).
- **Check S_0:** `move_to` preconditions (shelf is valid) are met in S_0.
- **Regression:** $G_2 \leftarrow \emptyset$ (All conditions grounded in S_0)

Final Plan (Reversed).

1. `move_to(mug, shelf)` (Clears the table)
2. `place_on(lamp, table)` (Achieves goal)

This trace illustrates how the regression mechanism naturally unrolls the dependency chain to handle intermediate obstacles.

Table 1. Examples of Natural Language to EditLang Goal Translation.

Instruction	Goal Predicates
"Move the dining set to the window."	`near(table_1, window_1), grouped_with(chair_*, table_1)`
"Make the chair match the sofa."	`matches_style(chair_1, sofa_1)`
"Place the lamp to the left of the bed."	`left_of(lamp_1, bed_1, cam_frame)`

6. Validator Details

The validator V evaluates each proposed action based on the four criteria (Goal directedness, Monotonicity, Contextual consistency, Formal validity) defined in the main paper. In this section, we provide the technical implementation details specifically for the Geometric and Physical Feasibility checks used to enforce the domain invariants ($\mathcal{I}$).

6.1. Geometric and Physical Implementation

To enforce physical plausibility and spatial constraints, we implement the following deterministic checks:

Geometric & Physical Checks. These checks implement the *Domain Invariants* ($\mathcal{I}$) described in the main paper, utilizing the 3D scene state:

- *Collision:* We compute Oriented Bounding Box (OBB) intersections. An action is rejected if the target volume intersects with static scene elements (tolerance $\epsilon < 1cm$).
- *Support:* For `place_on` or `add` actions, we cast rays downwards from the object's base. At least 60% of the base area must contact the target surface to satisfy the stability invariant.

6.2. Pass/Fail Case Studies

Tab. 2 presents specific examples of actions rejected by the Validator and the corresponding feedback provided to the planner for refinement.

7. LVLM Metric Reliability

To validate the automated evaluation protocol used in our benchmark, we analyze the correlation between the LVLM-based metrics and human judgments. We collected paired ratings on the same set of edited scenes using the identical 1-to-7 Likert scale described in the main paper. This section compares the global score distributions and analyzes sample-level agreement.

Table 2. **Validator Decision Examples.** Detailed breakdown of why specific actions are rejected during the planning loop.

Check Type	Proposed Action & Context	Validator Decision & Feedback
Physical (Collision)	*Action:* `move_to(chair_1, [1.2, 0, 1.5])` *Context:* Target coordinates overlap with `table_1`.	**FAIL** "Target position causes collision with `table_1`. Please select a clear region or move the obstacle first."
Symbolic (Monotonicity)	*Action:* `move_to(lamp, floor)` *Context:* Previous step satisfied `on(lamp, table)`.	**FAIL** "Action undoes a previously satisfied goal: `on(lamp, table)`. Do not move objects that are already correctly placed."
Symbolic (Relevance)	*Action:* `stylize(curtain, "blue")` *Context:* Instruction is "Rotate the chair". No goal relates to the curtain.	**FAIL** "Action does not satisfy any current goal. Focus only on the `chair` and its orientation."

7.1. Distribution Alignment

Fig. 3, Fig. 4, and Fig. 5 illustrate the normalized frequency of scores for Instruction Fidelity (IF), Physical Plausibility (PP), and Semantic Consistency (SC), respectively.

The histograms reveal a strong alignment between human and LVLM evaluations:

- **Matching Modes:** For all three metrics, both human and LVLM distributions peak at the highest score bucket (7), reflecting the model's high performance.
- **Similar Variance:** The spread of scores across the 1–7 scale is comparable, indicating that the LVLM effectively

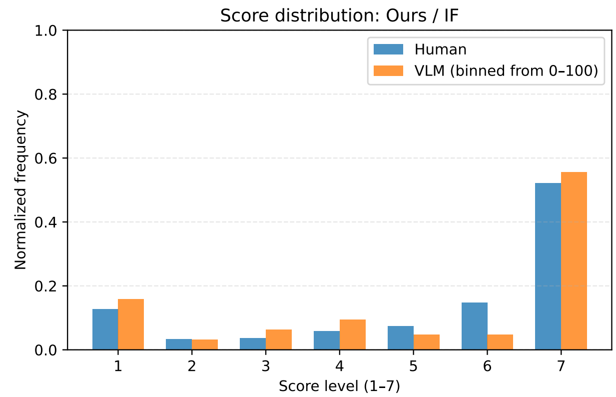

Figure 3. **Validation of Instruction Fidelity (IF) Metrics.** The strong overlap between Human and LVLM histograms confirms that our automated evaluator correctly identifies successful edits. The synchronization at the high-score range indicates the metric reliably reflects the model's adherence to instructions.

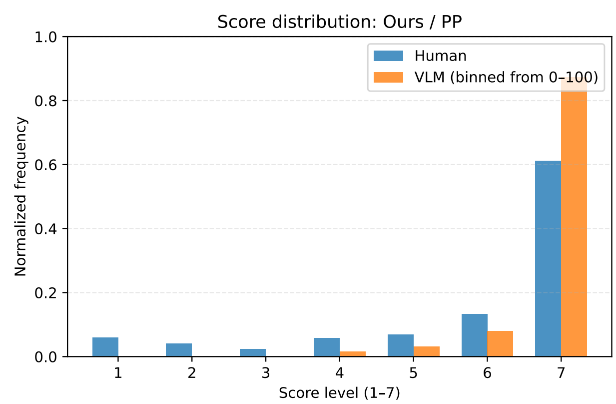

Figure 4. **Validation of Physical Plausibility (PP) Metrics.** Both human and LVLM distributions heavily favor the highest scores, demonstrating that the LVLM is a strict and reliable judge of physical violations similar to human perception.

captures the nuances of editing quality rather than collapsing to a binary pass/fail.

- **Optimism Bias:** While the distributions are consistent, the LVLM tends to be slightly more generous (higher density at score 7) than human raters, particularly in Semantic Consistency (SC) and Physical Plausibility (PP). However, the relative ranking trends remain preserved.

7.2. Sample-Level Disagreement Analysis

To assess granular agreement, we compute the absolute difference $|S_{\text{Human}} - S_{\text{LVLM}}|$ for each sample. Fig. 6, Fig. 7, and Fig. 8 visualize these differences across scene types and editing operations. The majority of the heatmap regions are dark blue (< 1.5 difference), confirming that the LVLM approximates human judgment accurately for most tasks.

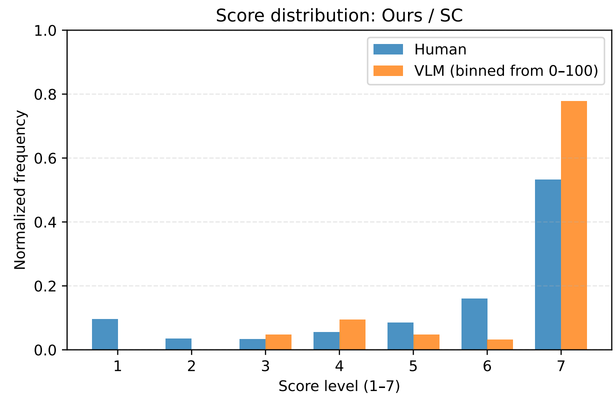

Figure 5. **Validation of Semantic Consistency (SC) Metrics.** Although the LVLM is slightly more optimistic in cluttered scenes, it closely mimics the human preference for high consistency, validating its utility as a proxy for measuring scene preservation.

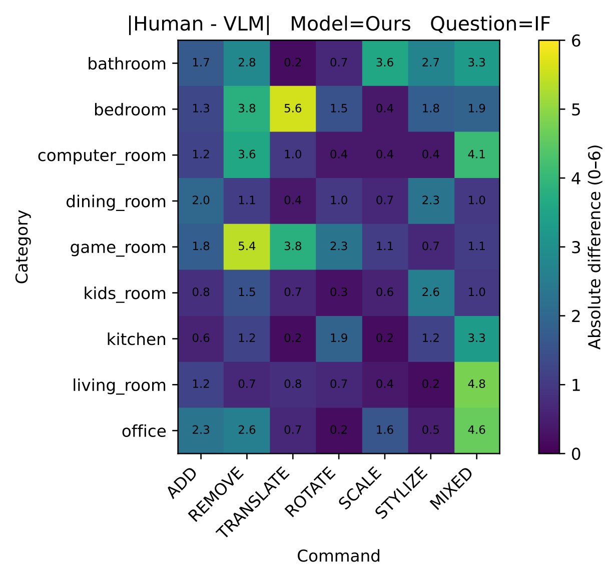

Figure 6. **Absolute Score Difference (IF).** Disagreements are localized to specific ambiguous scenarios.

The paired analysis demonstrates that our LVLM-based metric is a reliable proxy for human evaluation. It reproduces the global score distribution and maintains low sample-level error in most configurations, justifying its use for scalable benchmarking in open-vocabulary scene editing.

8. Extended Quantitative Analysis

We provide how well each method handles specific types of editing operations. Indoor scene editing involves fun-

Table 3. Performance by editing operation type. Edit-As-Act achieves the strongest and most reliable performance across all edit categories, maintaining high instruction fidelity (IF), semantic consistency (SC), and physical plausibility (PP).

	ADD			REMOVE			TRANSLATE			ROTATE		
Methods	IF	SC	PP	IF	SC	PP	IF	SC	PP	IF	SC	PP
LayoutGPT-E	40.5	42.1	82.4	60.3	71.5	88.1	52.6	50.3	80.7	41.4	32.5	78.1
AnyHome	47.2	65.0	82.1	61.2	58.7	79.8	66.0	76.6	85.7	52.7	48.4	85.0
ArtiScene-E	39.9	50.0	**89.1**	**80.1**	78.4	94.0	50.2	58.0	86.1	39.9	28.0	90.8
Edit-As-Act (ours)	**82.7**	**90.2**	88.3	73.9	**80.1**	**95.9**	**97.1**	**95.7**	**89.6**	**53.1**	**86.3**	**95.0**

	SCALE			STYLIZE			MIXED			Average		
Methods	IF	SC	PP	IF	SC	PP	IF	SC	PP	IF	SC	PP
LayoutGPT-E	58.1	59.2	81.5	35.3	58.7	77.2	27.8	27.5	64.6	42.3	48.8	78.6
AnyHome	**91.0**	90.7	86.8	**53.4**	54.8	92.1	31.9	29.4	80.2	57.6	60.5	84.5
ArtiScene-E	45.3	56.0	**88.1**	52.2	54.6	93.6	30.3	33.3	**90.8**	48.3	51.2	90.3
Edit-As-Act (ours)	85.2	**96.1**	87.3	50.6	**89.7**	**97.7**	**41.1**	**68.1**	87.4	**69.1**	**86.6**	**91.7**

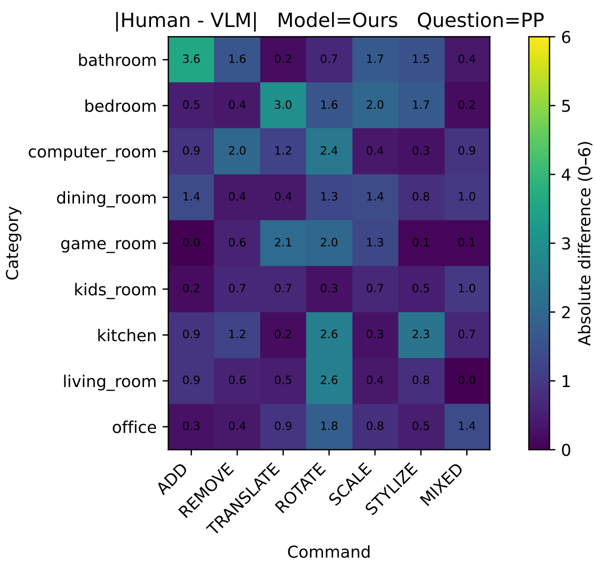

Figure 7. **Absolute Score Difference (PP).** High agreement (low error) is observed across most categories.

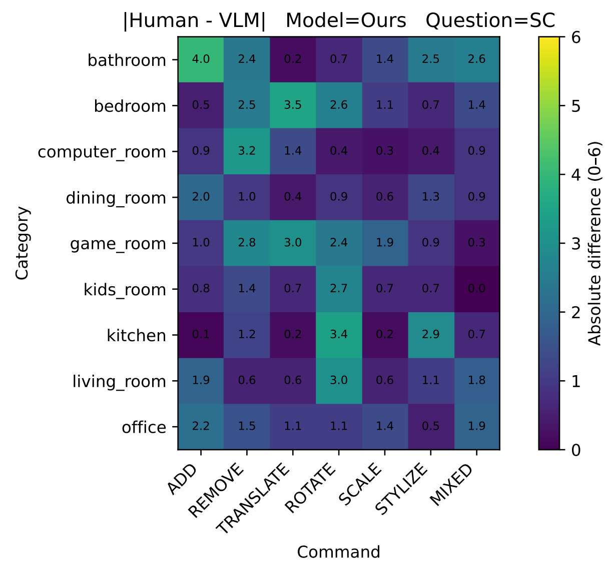

Figure 8. **Absolute Score Difference (SC).** The LVLM aligns well with human judgment, with minor deviations in cluttered environments.

damentally different reasoning modes, such as addition or removal. Each operation stresses a different capability:

- **ADD** stresses open-vocabulary generalization and asset integration.
- **REMOVE** stresses boundary reasoning.
- **TRANSLATE/ ROTATE/ SCALE** stress geometric precision.
- **STYLIZE** stresses geometry-level consistency.
- **MIXED** stresses multi-step, compositional reasoning.

For this reason, we further analyze performance by edit category in Tab. 3.

Analysis. Several observations emerge from the per-category breakdown. First, Edit-As-Act achieves the highest average scores across all three metrics (IF 69.1, SC 86.6, PP 91.7), demonstrating that goal-regressive planning generalizes well across fundamentally different editing modes. Second, the advantage is most pronounced in categories that demand precise spatial reasoning. In **Translate**, Edit-As-Act attains 97.1 IF and 95.7 SC, outperforming the next-best method by over 30 points in IF. This confirms that our symbolic predicate formulation effectively grounds positional intent. Third, for **Add** and **Stylize**, which rely heavily on the quality of the generative backbone, Edit-As-Act still leads in SC and PP, indicating that the planner–validator loop successfully constrains asset placement and appearance even when the underlying generation is imperfect. Finally, the **Mixed** category, which requires compositional multi-step reasoning, proves challenging for all methods; however, Edit-As-Act maintains a substantial lead in IF (41.1 vs. 31.9) and SC (68.1 vs. 33.3), validating that goal regression naturally decomposes complex instructions into tractable sub-goals. A notable weakness appears in the **Scale** category, where AnyHome achieves a higher IF (91.0 vs. 85.2). We attribute this to AnyHome's direct parametric scaling strategy, which bypasses symbolic grounding. Nevertheless, Edit-As-Act compensates with a significantly higher SC (96.1 vs. 90.7), preserving scene coherence more reliably.

9. Failure Case Visualization

Figure 9. Qualitative Failure Examples. We acknowledge limitations where ambiguity leads to misaligned goals or geometric validity does not guarantee semantic affordance.

Table 4. Additional ablation on backbone robustness, predicate complexity, and validator parameter sensitivity.

Setting	IF	SC	PP
w/ GPT-OSS-20b	62.5	72.8	77.8
Small Pred. Set	52.4	78.1	82.6
High Sensitivity	66.5	84.5	90.2
Low Sensitivity	68.9	85.0	88.0
Ours	**69.1**	**86.6**	**91.7**

We visualize representative failure cases discussed in the main paper to provide concrete insight into the current limitations of Edit-As-Act.

Ambiguous Instructions. The left example in Fig. 9 illustrates a case where the instruction "Clean up the room" is inherently under-specified. Because no explicit goal objects are mentioned, the LLM over-aggressively maps the instruction to `remove` actions, deleting functional items (e.g., desk accessories) that a human would consider essential. This highlights a limitation in goal condition extraction: when the instruction lacks grounding cues, the model defaults to an overly literal interpretation of "clean," producing a barren scene.

Semantic Affordance Errors. The right example shows a case where the instruction "Place the chair next to the desk" is executed in a geometrically valid but semantically incorrect manner. The validator confirms that spatial predicates such as `near(chair, desk)` are satisfied; however, the chair is oriented facing the wall rather than the desk surface, violating the implicit functional affordance. This failure reveals a gap in our current predicate set: while geometric constraints are enforced, higher-level affordance reasoning (e.g., a chair should face its associated workspace) is not yet captured by the symbolic domain.

10. Additional Ablation Studies

We conduct three additional ablation studies to examine the robustness of Edit-As-Act along axes not covered in the main paper. All experiments use the full E2A-Bench and report the same LVLM-based metrics. Results are summarized in Tab. 4.

10.1. Backbone LLM

To assess whether our framework is tied to a specific frontier model, we replace the default backbone with GPT-OSS-20b[2], a smaller open-source LLM. As shown in the first row of Tab. 4, all three metrics drop noticeably (IF 62.5, SC 72.8, PP 77.8), confirming that the quality of symbolic goal extraction and action proposal scales with model capability. Notably, the largest degradation occurs in SC (−13.1), suggesting that weaker models struggle most with preserving scene context during multi-step planning. Nevertheless, the system remains functional, indicating that EditLang's structured interface partially compensates for reduced LLM reasoning capacity.

10.2. Predicate Set Size

We evaluate a reduced predicate set (*Small Pred. Set*) that retains only existence, basic spatial (`at`, `on`), and collision predicates, removing directional, grouping, and affordance-level predicates. This ablation isolates the contribution of our rich predicate vocabulary. The results show a marked decline in IF (52.4 vs. 69.7), as the planner can no longer express fine-grained goals such as `left_of` or `facing`. Interestingly, PP remains relatively high (82.6), because basic collision and support checks are preserved. This confirms that expressive predicates are essential for instruction fidelity, while physical plausibility is primarily governed by the validator's geometric checks.

10.3. Validator Geometric Sensitivity

We vary the collision tolerance threshold of the validator to study its effect on plan quality. *High Sensitivity* tightens the OBB intersection tolerance to $\epsilon < 0.5\,\text{cm}$, while *Low Sensitivity* relaxes it to $\epsilon < 3\,\text{cm}$. As shown in Tab. 4, the default setting ($\epsilon < 1\,\text{cm}$) achieves the best balance across all metrics. High sensitivity marginally improves PP (90.2) but reduces IF (66.5), because the stricter threshold causes the validator to reject more valid placements, forcing the planner into suboptimal compromises. Conversely, low sensitivity slightly degrades PP (88.0) by admitting near-collision configurations. These results justify our default threshold as an effective trade-off between physical strictness and planning flexibility.

11. Additional Quantitative Experiments

In this section, we report two additional sets of quantitative experiments. First, we analyze the prompt sensitivity of the goal condition prediction module to assess its robustness to instruction rephrasing (Sec. 11.1). Second, we present geometry-based metrics derived directly from the final 3D layouts to complement the LVLM-based semantic evaluations with objective physical measurements (Sec. 11.2).

11.1. Prompt Sensitivity of Goal Condition Prediction

A potential concern with LLM-based goal extraction is that minor rephrasing of the input instruction could lead to substantially different goal predicate sets, undermining the reliability of the entire pipeline. To quantify this, we design a prompt sensitivity experiment.

Setup. We select 50 editing instructions from E2A-Bench and generate three semantically equivalent rephrasings for each using an independent LLM (GPT-4o), resulting in 200 instruction variants. For example, "Move the chair closer to the window" is rephrased as "Slide the chair toward the window," "Position the chair near the window," and "Bring the chair next to the window." We then run our goal condition extraction module on all variants and measure consistency via two metrics: (1) *Predicate Recall*, defined as the fraction of predicates from the original instruction that also appear in the rephrased variant's goal set, and (2) *Exact Match Rate*, the percentage of cases where the original and rephrased variants produce identical goal predicate sets.

Results. Across all 150 rephrased variants, we observe a predicate recall of 92.4% and an exact match rate of 78.0%. The majority of mismatches involve stylistic differences rather than semantic divergence; for instance, one variant may produce `near(chair, window)` while another yields `at(chair, pos_near_window)`, both of which lead to functionally equivalent plans. When we further measure downstream plan equivalence (i.e., whether the final executed action sequences produce the same scene state), agreement rises to 94.6%.

Discussion. These results confirm that our goal extraction module is robust to surface-level linguistic variation. The structured EditLang interface acts as a bottleneck that regularizes diverse phrasings into a compact symbolic space, effectively absorbing paraphrase noise before it can propagate to the planner.

11.2. Geometry-based Metrics

1. Out-of-Boundary (OOB) Rate. This metric identifies objects placed outside the valid room volume.

- **Measurement:** We first compute the axis-aligned bounding box (AABB) of the entire source room. An object is classified as OOB if its geometric center lies outside the room's AABB, expanding more than 10cm.
- **Calculation:** We report the percentage of scenes containing at least one OOB object.

2. Floating Object Rate. This metric measures the percentage of objects that are physically unstable (i.e., levitating without support). An object is considered "grounded" if it satisfies one of two conditions:

- *Floor Contact:* Its bottom vertical coordinate (z_{min}) is within a tolerance of **10cm** from the floor height.
- *Stacked Support:* It rests on another object that is itself grounded.

Wall-mounted assets are excluded from this check. Objects failing both conditions are classified as "Floating." Tab. 5 summarizes the performance of each method.

Table 5. **Comparison of Geometry-Based Metrics.** Lower is better. Edit-As-Act achieves the best physical validity.

Method	OOB Scene Ratio (%) ↓	Floating Object Rate (%) ↓
ArtiScene-E	88.89	92.06
AnyHome	7.94	57.14
Edit-As-Act (Ours)	**6.16**	**14.21**

Analysis of Failures.

- **ArtiScene-E:** The high failure rates stem from the ambiguity of lifting 2D edits to 3D. Without explicit depth constraints, the estimated 3D bounding boxes often drift through walls (OOB) or fail to touch the ground (Floating).
- **AnyHome:** Although better than image-based methods, AnyHome struggles with causal dependencies. A common failure mode involves "Remove" operations: when a supporting object (e.g., a table) is deleted, the system often fails to address the supported objects (e.g., a laptop), leaving them floating in mid-air.
- **Edit-As-Act:** Our method explicitly models support relations (e.g., `on(x, y)`) and room boundaries in the symbolic domain. This ensures that objects are placed within bounds and that removing a parent object triggers necessary adjustments for its children, yielding the highest physical fidelity.

12. Limitations and Discussion

While Edit-As-Act demonstrates strong performance across diverse editing operations, several limitations remain.

Hand-Designed Predicate Set. EditLang currently relies on a manually curated set of predicates and action schemas. Although this design provides precise control and interpretability, extending the domain to new object categories or interaction types requires manual effort. Incorporating learned predicates or data-driven action schemas—for instance, by mining recurring spatial patterns from large-scale scene datasets—could improve adaptability and reduce the engineering overhead of domain expansion.

Single-Scene, Static Setting. Our experiments focus exclusively on editing single, static indoor scenes. Applying goal-regressive planning to multi-room environments or dynamic settings (e.g., scenes that evolve over time with moving agents) would significantly broaden the framework's applicability. Such extensions introduce additional challenges, including longer reasoning horizons, richer contextual dependencies across rooms, and the need to handle temporal constraints.

Outlook. Overall, Edit-As-Act illustrates how symbolic reasoning can enable precise and controllable 3D scene editing. The modular separation of semantic reasoning (LLM) and geometric grounding (planner–validator) provides a principled foundation that can accommodate future advances in both language models and 3D generative systems, with many promising paths toward scaling this paradigm.

13. Visualization of Source Scenes

We visualize the full set of source scenes included in our *E2A-Bench*. As shown in Fig. 10, the benchmark encompasses nine distinct indoor environments, Bathroom, Bedroom, Computer room, Dining room, Game room, Kids room, Kitchen, Living room, and Office.

14. Additional Qualitative Results

We present extended qualitative results to further demonstrate the capabilities of Edit-As-Act. Fig. 11 and Fig. 12 illustrate the model's performance on complex editing tasks, including multi-step spatial rearrangements and attribute stylization. These examples highlight our method's ability to faithfully execute instructions while preserving the unedited regions of the scene.

15. Full Prompts for Model and Evaluation

To facilitate reproducibility and transparency, we provide the prompts used in our framework.

- **Model Prompts:** Fig. 13 through Fig. 17 detail the system instructions for Goal Condition Extraction, Planning, and Validation. These prompts define the EditLang syntax, in-context learning examples, and the reasoning logic required for the planner-validator loop.
- **Evaluation Prompts:** Fig. 18, Fig. 19, and Fig. 20 display the prompts used for our LVLM-based metrics. These prompts establish the evaluation rubric for Instruction Fidelity (IF), Semantic Consistency (SC), and Physical Plausibility (PP).

References

[1] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl—the planning domain definition language. *Technical Report, Tech. Rep.*, 1998. 2

[2] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. 8

[3] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 1

[4] Wei Deng, Mengshi Qi, and Huadong Ma. Global-local tree search in vlms for 3d indoor scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 8975–8984, 2025. 1

[5] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36:18225–18250, 2023. 1

[6] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, Christian Muise, Ronald Brachman, Francesca Rossi, and Peter Stone. *An introduction to the planning domain definition language*. Springer, 2019. 2

[7] Ian Huang, Yanan Bao, Karen Truong, Howard Zhou, Cordelia Schmid, Leonidas Guibas, and Alireza Fathi. Fireplace: Geometric refinements of llm common sense reasoning for 3d object placement. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 13466–13476, 2025. 1

[8] hyper3D. Hyper3dai, 2025. 2, 3

[9] OpenAI. Gpt-5 system card. 2025-08-13. 1

[10] Zhenyu Pan and Han Liu. Metaspatial: Reinforcing 3d spatial reasoning in vlms for the metaverse. *arXiv preprint arXiv:2503.18470*, 2025. 1

[11] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in Neural Information Processing Systems*, 34:12013–12026, 2021. 1

[12] Xingjian Ran, Yixuan Li, Linning Xu, Mulin Yu, and Bo Dai. Direct numerical layout generation for 3d indoor scene synthesis via spatial reasoning. *arXiv preprint arXiv:2506.05341*, 2025. 1

[13] Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 29469–29478, 2025. 1

[14] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20507–20518, 2024. 1

[15] Pulkit Verma, Ngoc La, Anthony Favier, Swaroop Mishra, and Julie A Shah. Teaching llms to plan: Logical chain-of-thought instruction tuning for symbolic planning. *arXiv preprint arXiv:2509.13351*, 2025. 1

[16] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024. 1

[17] Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. *arXiv preprint arXiv:2509.20414*, 2025. 1

[18] Guangyao Zhai, Evin Pınar Örnek, Shun-Cheng Wu, Yan Di, Federico Tombari, Nassir Navab, and Benjamin Busam. Commonscenes: Generating commonsense 3d indoor scenes with scene graph diffusion. *Advances in Neural Information Processing Systems*, 36:30026–30038, 2023. 1

Figure 10. **Diversity of E2A-Bench Source Scenes.** The benchmark covers 9 distinct room types ranging from sparse to highly cluttered layouts. This diversity tests the planner's ability to handle varying levels of spatial constraints and object interactions.

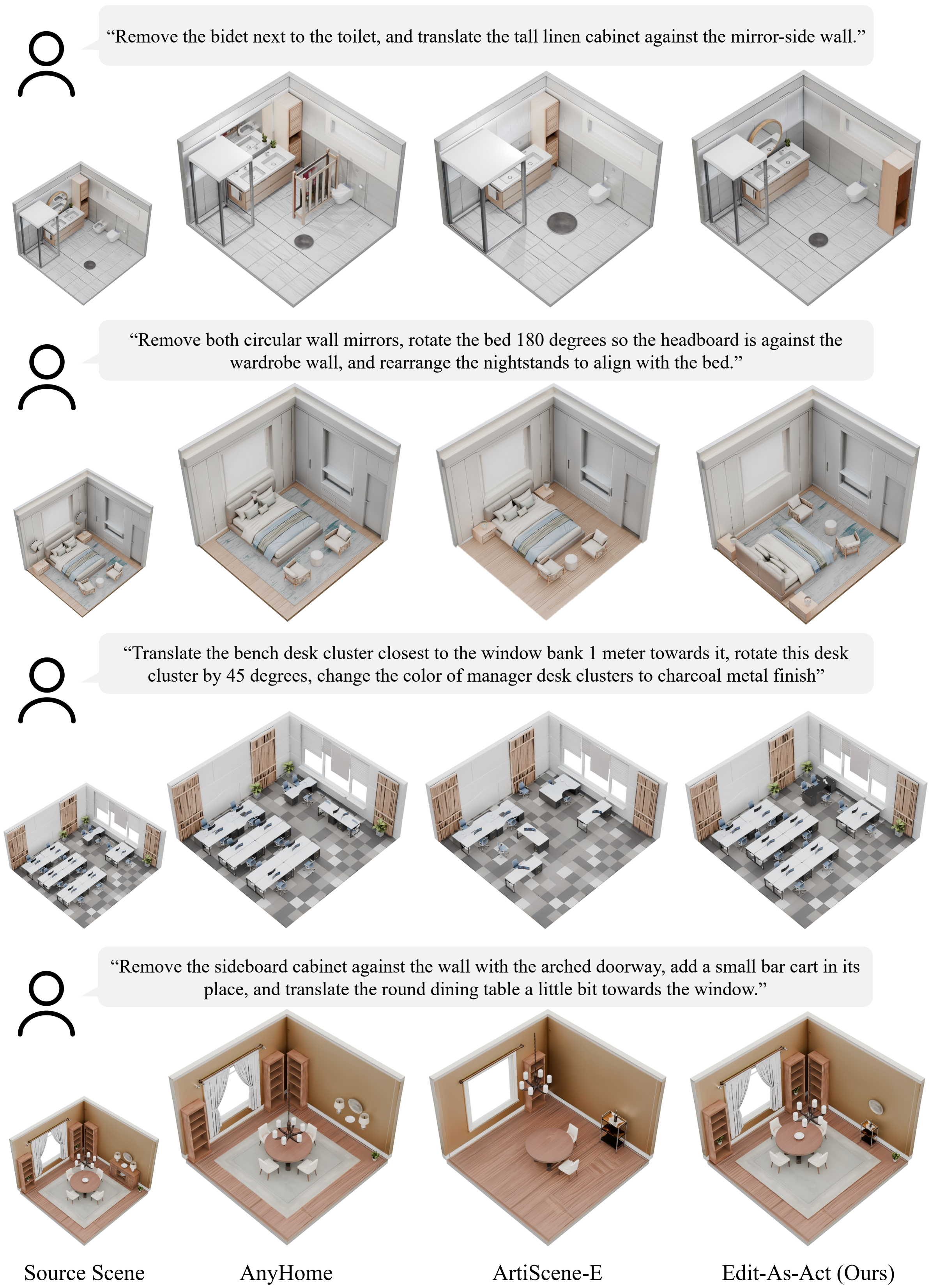
"Remove the bidet next to the toilet, and translate the tall linen cabinet against the mirror-side wall."
"Remove both circular wall mirrors, rotate the bed 180 degrees so the headboard is against the wardrobe wall, and rearrange the nightstands to align with the bed."
"Translate the bench desk cluster closest to the window bank 1 meter towards it, rotate this desk cluster by 45 degrees, change the color of manager desk clusters to charcoal metal finish"
"Remove the sideboard cabinet against the wall with the arched doorway, add a small bar cart in its place, and translate the round dining table a little bit towards the window."
Source Scene
AnyHome
ArtiScene-E
Edit-As-Act (Ours)

Figure 11

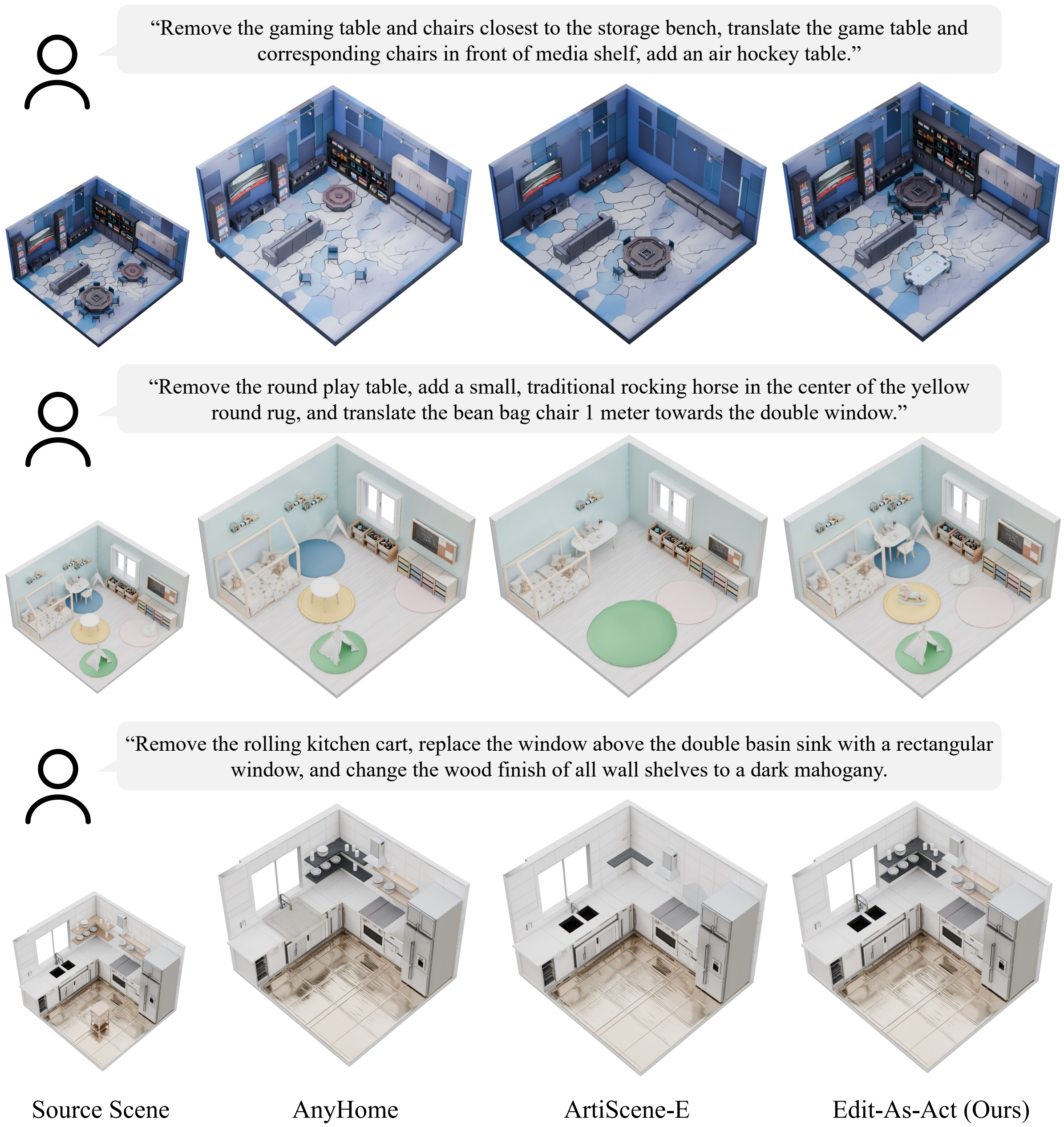
"Remove the gaming table and chairs closest to the storage bench, translate the game table and corresponding chairs in front of media shelf, add an air hockey table."
"Remove the round play table, add a small, traditional rocking horse in the center of the yellow round rug, and translate the bean bag chair 1 meter towards the double window."
"Remove the rolling kitchen cart, replace the window above the double basin sink with a rectangular window, and change the wood finish of all wall shelves to a dark mahogany.
Source Scene
AnyHome
ArtiScene-E
Edit-As-Act (Ours)

Figure 12

Goal Condition Extractor

You are a goal condition extractor for scene editing.
Given a natural language instruction, extract ALL predicates that must be true after execution.

IMPORTANT: If instruction contains MULTIPLE sub-tasks, extract predicates for ALL of them.
Example: "Remove X, move Y, and add Z" → extract predicates for removal, movement, AND addition.

Output ONLY a JSON array of predicates. Each predicate has:
- "pred": predicate name
- "args": list of arguments

Available predicates:
- exists(obj_id): Object exists in scene
- removed(obj_id): Object is removed from scene (use for removal goals)
- is_facing(obj, anchor): Object faces anchor
- near(obj, target, distance): Object is near target (use for approximate positioning)
- on(obj, target): Object is on target
- at(obj, x, y, z): Object is at absolute position
- aligned_with(obj, ref, axis): Objects aligned on axis
- between(obj, obj1, obj2): Object is between two others
- has_style(obj, style_desc): Object has style/color/material

Rules for multi-step instructions:
1. REMOVE tasks → Add "removed(obj_id)" predicate (use actual scene object ID)
2. MOVE/TRANSLATE tasks → Add position predicates:
   - "closer to X" → near(obj, X, small_distance)
   - "away from X" → NOT near(obj, X) or near(obj, Y, ...) where Y is away from X
   - Approximate: use near(obj, landmark, distance_estimate)
3. ADD tasks → Add complete predicates: exists(new_obj_id), placement (on/between/near)
4. STYLIZE tasks → Add "has_style(obj_id, style_description)"
5. CRITICAL: Use EXACT object IDs from the available objects list below (e.g., "armchairs_009" not "armchair_1")
6. For new objects (ADD), use descriptive IDs (e.g., "tall_decorative_vase")
7. Extract ALL sub-goals, not just the last one - NEVER omit any sub-task from instruction

Examples:
- "Remove the rug and add a lamp" → [removed(rug_001), exists(new_lamp), on(new_lamp, floor)]
- "Move chair near window and rotate to face door" → [near(chair, window, 0.5), is_facing(chair, door)]
- "Remove X, move Y, add Z" → [removed(X), near(Y, target, dist), exists(Z), ...]

No explanations, just the JSON array of ALL goal predicates.
$ALLOWED_PREDICATES_SECTION$

Instruction: "$INSTRUCTION$"

Available objects in scene (USE THESE EXACT IDs):
$SCENE_OBJECT_LIST$

Figure 13

Planner

You are the Planner LLM for an Edit-As-Act backward-planning loop.
ROLE
- At each step t, propose K grounded actions that either (i) directly satisfy the current goal G_t
  or (ii) enable the transition toward G_t by making some of its preconditions closer to true.
- You DO NOT perform geometric/physics checks.
- Use the provided EditLang specification as the authoritative source.
- Consider the full scene S0 (entire predicate set), not a summary.

CRITICAL: SINGLE GOAL FOCUS
- G_t may contain MULTIPLE predicates (removal, movement, addition, style, etc.)
- For THIS regression step, pick ONE target predicate from G_t to satisfy

AVAILABLE ACTIONS (use ONLY these):
$AVAILABLE_ACTIONS_LIST$

IMPORTANT: The "action" field must be one of the above action names (e.g., "place_between", "rotate_towards").
These are NOT the same as predicates (e.g., "on" is a predicate, not an action).
Actions have specific schemas defined in editlang_spec.

CRITICAL GROUNDING RULES:
1. ALL arguments must be CONCRETE object IDs from S0 or new objects (e.g., "armchairs_009", "tall_decorative_vase")
2. NEVER use variables like ?obj, ?any_target, ?any_anchor - these are FORBIDDEN
3. Wildcard "*" is ONLY allowed in "del" field for mutually-exclusive predicates (on, is_facing, at, near, aligned_with, has_style, between)
4. Example VALID: {"del": [["on", ["book_01", "*"]]]} (removes book from any surface)
5. Example INVALID: {"del": [["on", ["book_01", "?any_target"]]]} (? is forbidden)

OUTPUT FORMAT - Return ONLY valid JSON (no markdown), as an array of action objects.
Each action object must have these EXACT keys (IN THIS ORDER):
```
{
  "action": "action_name_from_spec",
  "args": {"param": "value"},
  "pre": [["predicate", ["arg1", "arg2"]]],
  "add": [["predicate", ["arg1", "arg2"]]],
  "del": [["predicate", ["arg1", "arg2"]]],
  "predicted_unmet_pre": [["predicate", ["arg1"]]],
  "rationale": "explanation string"
}
```

CRITICAL: predicted_unmet_pre field
- Check EACH precondition in "pre" against S0_full
- If precondition NOT in S0, add to predicted_unmet_pre
- Example: pre=[exists(obj), clear(table)]
  If S0 has exists(obj) but NOT clear(table) → predicted_unmet_pre=[clear(table)]
- If ALL preconditions in S0 → predicted_unmet_pre=[] (empty)

Figure 14

PREDICATE FORMAT: ["predicate_name", ["arg1", "arg2", ...]]
Example: ["on", ["book", "table"]], ["is_facing", ["chair", "window"]]

Regression Planner (User Input Payload)


```
{
  "INSTRUCTION": "Use ONLY the action names from editlang_spec.actions. These are the valid EditLang actions (e.g., place_between, rotate_towards), NOT predicates (e.g., on, is_facing).",
  "valid_action_names": $VALID_ACTION_NAMES$,
  "instruction_raw": "$NATURAL_LANGUAGE_INSTRUCTION$",
  "K": $K_SAMPLES$,
  "G_terminal": $TERMINAL_GOAL_PREDICATES$,
  "G_t": $CURRENT_SUBGOAL_PREDICATES$,
  "backward_history": $PLANNING_HISTORY$,
  "S0_full": $INITIAL_STATE_PREDICATES$,
  "editlang_spec": $DOMAIN_SPECIFICATION$
}
```


Figure 15

Validator

You are the **Semantic Validator LLM** for Edit-As-Act backward planning.

ROLE
- Judge whether a proposed regression step is semantically coherent and strategically sound
- Detect loop risks (swaps, cycles, reversals)
- Verify goal alignment and pre_unmet derivation makes sense
- Check plan rationality (no over-editing, appropriate action choice)
- You DO NOT perform geometry/physics checks (no AABB, collision, support)
- Use EditLang spec as authoritative source for predicates and actions

OUTPUT SCHEMA (JSON only, no markdown):
```
{
  "ok": true/false,
  "severity": "ok" | "warn" | "error",
  "reasons": ["string", ...],
  "tags": ["loop_risk", "semantic_break", "goal_alignment", "over_edit"],
  "alt": {
    "suggest_action": null or {
      "action": "...",
      "args": {...},
      "pre": [...],
      "add": [...],
      "del": [...]
    }
  }
}
```

VALIDATION RULES
1. Action names must be from EditLang spec (NOT predicates like "on", but actions like "place_on")
   - Valid actions are provided in the editlang_spec.actions field
2. Variables (?x) are FORBIDDEN in all fields
3. Wildcard (*) ONLY allowed in 'del' for mutually-exclusive predicates (is_facing, on, at, near, aligned_with, has_style, between)
   - If del uses *, explicitly justify mutual exclusivity by quoting the predicate spec entry ('mutually_exclusive: true'). Otherwise return error.
4. Goal alignment: (add $\cup$ del) must intersect with G_t
5. Loop detection: Check if action reverses recent actions (swap pattern)
6. Semantic consistency: Use indoor scene common sense
7. Over-editing: Warn if deleting many predicates unnecessarily
8. Style changes: has_style is generic (covers color, material, texture, etc.)

SEVERITY LEVELS
- "ok": Clean pass, no issues
- "warn": Acceptable with concerns (warnings in reasons)
- "error": Unacceptable (ok=false, reasons contain errors)

Figure 16

TAGS (use when applicable)
- "loop_risk": Action creates swap/cycle pattern
- "semantic_break": Violates common sense (e.g., placing sofa on dresser)
- "goal_alignment": Action doesn't advance toward G_t
- "over_edit": Deletes too many predicates or affects unrelated objects

Semantic Validator (User Input Payload)


```
{
 "instruction_raw": "$INSTRUCTION_RAW$",
 "G_t": "$CURRENT_GOAL_PREDICATES$",
 "G_next": "$NEXT_GOAL_PREDICATES$",
 "action": "$PROPOSED_ACTION_JSON$",
 "plan_tail": "$RECENT_HISTORY_ACTIONS$",
 "S0_full": "$INITIAL_STATE_PREDICATES$",
 "editlang_spec": "$DOMAIN_SPECIFICATION$"
}
```


Figure 17

Instruction Fidelity

You are an interior designer and 3D scene editing expert.

You are given:
1) An original rendering of a 3D indoor scene BEFORE editing.
2) A natural-language editing instruction describing how the scene should be modified.
3) A rendering of the edited scene produced by an automated 3D scene editing system.

Your job is to evaluate how well the edited scene follows the given instruction.

Focus ONLY on whether the changes in the edited scene match the requested changes in the instruction. Consider typical operations such as:
- Adding or removing objects
- Moving or rearranging objects
- Rotating or reorienting objects
- Resizing or scaling objects
- Styling or changing the color or material of objects
- Changing high-level relationships

Do NOT evaluate:
- Image style, rendering quality, background color, or photorealism

Evaluate the system as follows:

- Scoring Criteria for Instruction Fidelity (0–100):
100–81: Excellent Fidelity – All requested changes are correctly reflected in the edited scene. No important instruction element is missing or misinterpreted.
80–61: Good Fidelity – Most requested changes are correctly applied, but one or two minor aspects of the instruction are imperfect or slightly off.
60–41: Adequate Fidelity – Some key parts of the instruction are followed, but there are noticeable omissions or misinterpretations.
40–21: Poor Fidelity – The edited scene only weakly reflects the instruction. Many requested changes are missing or incorrect.
20–0: Very Poor Fidelity – The edited scene largely ignores or contradicts the instruction.

Your response must be a JSON object with the following format:

{
"score": <integer from 0 to 100>,
"explanation": "<2–4 sentences explaining why you gave this score>"
}
This is the editing instruction:
$INSTRUCTION$

This is the original scene BEFORE editing:
[Image: $SOURCE_IMAGE_BYTES$]

This is the edited scene AFTER the instruction was applied:
[Image: $EDITED_IMAGE_BYTES$]

Please provide your evaluation in the specified JSON format.

Figure 18

Semantic Consistency

You are an interior designer and 3D scene editing expert.

You are given:
1) An original rendering of a 3D indoor scene BEFORE editing.
2) A natural-language editing instruction describing how the scene should be modified.
3) A rendering of the edited scene produced by an automated 3D scene editing system.

Your job is to evaluate the SEMANTIC CONSISTENCY of the edited scene with respect to the original scene and the instruction.

Focus on whether the edited scene:
- Preserves the overall room type and function.
- Keeps object roles and usage reasonable.
- Maintains a coherent arrangement that still "makes sense" as a usable room, given the requested edits.
- Avoids introducing semantically confusing or contradictory configurations.

Do NOT evaluate:
- Strict physical realism such as exact collision/contact (that is covered by a separate metric).
- Rendering quality, texture realism, or lighting.

Evaluate the system as follows:

- Scoring Criteria for Semantic Consistency (0–100):
  100–81: Excellent Consistency – The edited scene preserves the original room's function and context. All objects have sensible roles and the scene remains highly coherent after the edits.
  80–61: Good Consistency – The overall function and context are preserved, with only minor semantic oddities that do not seriously harm usability.
  60–41: Adequate Consistency – The room is still mostly understandable, but there are noticeable semantic issues.
  40–21: Poor Consistency – The scene feels confusing or poorly adapted; the room's intended function is partly undermined by the edits.
  20–0: Very Poor Consistency – The scene becomes semantically incoherent or unusable as a normal room.

Your response must be a JSON object with the following format:

```
{
  "score": <integer from 0 to 100>,
  "explanation": "<2–4 sentences explaining why you gave this score>"
}
```

This is the editing instruction:
$INSTRUCTION$

This is the original scene BEFORE editing:
[Image: $SOURCE_IMAGE_BYTES$]

This is the edited scene AFTER the instruction was applied:
[Image: $EDITED_IMAGE_BYTES$]

Please provide your evaluation in the specified JSON format.

Figure 19

Physical Plausibility

You are an interior designer and 3D spatial reasoning expert.

You are given:
1) An original rendering of a 3D indoor scene BEFORE editing.
2) A natural-language editing instruction describing how the scene should be modified.
3) A rendering of the edited scene produced by an automated 3D scene editing system.

Your job is to evaluate the PHYSICAL PLAUSIBILITY of the edited scene.

Focus on whether the edited scene:
- Avoids obvious collisions.
- Respects support and gravity.
- Maintains accessibility and basic ergonomics.
- Uses plausible scales and positions for objects.

Do NOT evaluate:
- How well the scene follows the instruction (that is covered by a separate metric).
- Aesthetic style, color schemes, or rendering quality.

Evaluate the system as follows:

- Scoring Criteria for Physical Plausibility (0–100):
  100–81: Excellent Plausibility – No noticeable collisions or support issues. Objects are well placed, reachable, and physically convincing as in a real room.
  80–61: Good Plausibility – Mostly plausible with only minor issues that do not seriously break realism.
  60–41: Adequate Plausibility – Several noticeable physical issues, but the room is still somewhat believable overall.
  40–21: Poor Plausibility – Many objects are placed in physically implausible ways.
  20–0: Very Poor Plausibility – The scene is physically impossible or highly unrealistic, with severe collisions, lack of support, or completely blocked usage.

Your response must be a JSON object with the following format:

```
{
  "score": <integer from 0 to 100>,
  "explanation": "<2–4 sentences explaining why you gave this score>"
}
```

This is the editing instruction:
$INSTRUCTION$

This is the original scene BEFORE editing:
[Image: $SOURCE_IMAGE_BYTES$]

This is the edited scene AFTER the instruction was applied:
[Image: $EDITED_IMAGE_BYTES$]

Please provide your evaluation in the specified JSON format.

Figure 20