VOXPROG: 3D SPATIAL REASONING THROUGH VOXEL PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models' abilities to interpret tasks and generate programs offer a promising basis for 3D spatial reasoning. Yet applying these abilities to spatial instructions requires generated programs to construct geometric references that language alone does not fully specify. In this work, we introduce VoxProg, which addresses this challenge through *voxel programming*: expressing task interpretations as executable programs that operate on local voxel views of scene surfaces. By organizing surface geometry on regular grids with explicit coordinates and neighborhood relations, these programs construct task-relevant regions and boundaries while retaining spatial extent and shape. A downstream pose program then uses this geometry to compute target object poses, given explicit scene geometry and known initial object poses. Visual and numerical execution feedback allows the model to inspect intermediate structures and iteratively revise its programs. We also introduce VoxProg-Bench, a benchmark of 71 open-ended placement tasks whose diverse and complex spatial requirements must be inferred from the interplay of instructions, scene contexts, and object geometries. Its evaluation combines Intent Alignment, measured by image comparison with the ground-truth placement, with Physical Validity checks. Code will be released upon acceptance.

1 INTRODUCTION

Natural-language spatial instructions often specify goals whose geometric requirements must be inferred from the scene. Placing a bottle inside an open display holder requires identifying the holder's receiving structure and determining how the bottle should be positioned and oriented to fit within it. Similarly, extending a row of cups requires inferring the direction and spacing of the existing arrangement to derive the next placement target. Here, the geometric references that define the goal are themselves outcomes of spatial computation.

Existing approaches are limited by both the task requirements their interfaces can express and the geometric computations their representations support. Predefined task interfaces couple task coverage to task-specific implementations, limiting generalization to new spatial requirements. Geometric representations present a different difficulty: sparse keypoints can leave regions and local surface shapes under-specified (Huang et al., 2025b; Hua et al., 2025; Manuelli et al., 2022), while raw meshes require programs to organize irregular surface elements into the regions and references a task needs. Open-ended spatial reasoning therefore requires computations that adapt to the instruction and a spatial representation that organizes the relevant scene geometry for those computations.

To address these requirements, we formulate *voxel programming*, in which a language model expresses its understanding of a spatial task as a program that constructs and operates on local voxel views (Fig. 1). These views organize scene surfaces on regular grids, directly exposing spatial coordinates and neighborhood relations. Programs can use this structure to select regions, isolate boundaries, and measure their spatial extent and shape. The representation provides a common basis for geometric computation, while the instruction determines which geometry matters and how it should be used to determine the target position and orientation.

VoxProg instantiates this formulation by coupling programmatic geometric construction with visual grounding and execution feedback. From the instruction and available visual anchors, the model generates an *evidence program* that constructs task-relevant geometry and describes the intended

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



Figure 1: **Task-relevant geometry construction with VoxProg.** A voxel program selects support regions for controller placement. The lower examples illustrate containment, clearance, and arrangement requirements.

relations between its regions. These relations guide the generation of a *pose program*, which measures the selected regions to determine how the object should be positioned and oriented. The model inspects intermediate selections and candidate placements using numerical outputs and renderings, and can revise its programs in response.

We introduce VoxProg-Bench, a benchmark of 71 placement tasks spanning functional use, geometric fit, and contextual arrangements. Their spatial requirements arise from the interplay of instructions, scene contexts, and object geometries. We also propose an evaluation suite that pairs relative visual alignment with geometric validity checks. Its IA Rank Score aggregates within-task rankings by similarity to ground-truth images. Physical Validity checks whether placements are supported and free of excessive interpenetration.

Contributions. ❶ We formulate voxel programming and develop VoxProg, a system that generates programs to construct task-relevant geometry and compute target object poses. ❷ We introduce VoxProg-Bench, a benchmark of open-ended placement tasks spanning functional goals, geometric fit, and contextual arrangements. ❸ We propose an evaluation suite using perceptual rankings for intent alignment and geometric checks for physical plausibility.

On VoxProg-Bench, VoxProg produces placements that more closely match the ground-truth placements than those of the baseline methods. The largest IA Rank Score margin over the strongest baseline occurs on geometric placement tasks, where object contours determine fit and clearance. Constructing geometric evidence before pose synthesis improves intent alignment over direct mesh placement; using local voxel views for that construction brings a further gain.

2 Related Work

Spatial task specification. Language-guided placement has been studied through layout prediction, rearrangement search, and geometric optimization. Scene graphs encode relationships among objects for scene synthesis (Chen et al., 2025; Zhai et al., 2023; Lin & Mu, 2024), while scene-layout and object-placement methods turn language into spatial configurations (Feng et al., 2023; Yang et al., 2024; Öcal et al., 2024; Sun et al., 2025; Huang et al., 2025a; Abdelreheem et al., 2025). For rearrangement, language-conditioned structure generation (Liu et al., 2022; 2023) and search with pattern priors (Chang et al., 2024) provide ways to coordinate multiple object placements. Manipulation methods make finer geometric requirements explicit through semantic keypoints and directions, yielding compact objectives for pose optimization (Manuelli et al., 2022; Hua et al., 2025; Huang et al., 2025b). Associating these references with object parts and functions further allows the objectives to reflect how objects are intended to interact through the parts that serve their functional roles (Qi et al., 2025; Pan et al., 2025; Zhu et al., 2025).

VoxProg addresses the construction of the geometry on which such objectives depend. Through generated programs, the model selects relevant surface regions and derives the positions, directions, or boundaries needed for a placement. Retaining these regions makes their extent and local shape available for subsequent measurements, allowing the geometric references to be chosen according to the instruction. The same object can therefore support different task-specific interpretations within a shared computational framework. Task specification in VoxProg encompasses both the desired spatial relations and the computations that establish their geometric meaning in the scene.

Language models for spatial computation. Executable programs allow language models to combine semantic reasoning with external computation. Numerical interpreters (Gao et al., 2023) and modular perception systems (Gupta & Kembhavi, 2023; Surís et al., 2023) provide operations that a model can compose to solve a problem. In robotics, generated code has supported task planning (Singh et al., 2023), the integration of control calls with geometric calculations (Liang et al., 2023; Mu et al., 2024), and spatial objective construction for motion planning (Huang et al., 2023). Geometric processing also appears within generated programs, including object-part mask refinement and constraint evaluation (Tang et al., 2025), as well as primitive fitting for trajectory synthesis (Lin et al., 2025). For visual spatial reasoning, explicit reference frames and objectives further help organize tool-based computation (Chen et al., 2026). These applications illustrate how the operations exposed to a model shape the computations it can express.

We develop *voxel programming* to let a language model formulate and carry out the spatial computations required by a placement task. Its scope includes constructing the geometry to reason over and composing the operations that turn it into a target pose. Local voxel views support this process by exposing scene surfaces through regular spatial indexing and neighborhood operations. Task-specific logic is expressed in the generated program, while the underlying spatial operations remain reusable across instructions. Execution feedback exposes intermediate geometry and resulting placements for inspection, allowing the model to revise its computations against their intended spatial effects.

3 METHOD

Given a natural-language instruction $\mathcal{I}$ and a 3D scene $\mathcal{S} = \{o_n\}_{n=1}^{N}$ with accessible object geometry and poses, VoxProg generates and executes programs to determine a target object pose. The model interprets the instruction in the context of the scene, identifying the spatial requirements that the resulting placement should satisfy. It expresses this interpretation through two executable programs: an *evidence program* constructs task-relevant geometry using local voxel views, and a *pose program* uses these regions to compute the object's target position and orientation.

An evidence graph organizes the spatial references needed for the task and their dependencies, while visual grounding supplies initial anchors (§3.1). The evidence program constructs and operates on local voxel views to construct these regions (§3.2). During program construction, visual and numerical feedback from execution allows the model to inspect intermediate results and revise its code (§3.3). The resulting geometry remains available to the pose program, which uses it together with the desired spatial relations to determine the target position and orientation (§3.4). Fig. 2 illustrates this process on the controller example.

3.1 TASK INTERPRETATION AND VISUAL GROUNDING

A large language model (LLM) represents its interpretation of the task as an evidence graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Nodes identify visual anchors or task-relevant geometry to be constructed, together with their semantic roles. An edge (u, v) indicates that constructing v uses information from u. In the controller example of Fig. 2, visual cues help locate the grips and the stand regions whose support geometry must be computed to position the controller on the stand.

Routing determines whether a requested reference is obtained through visual grounding or geometric computation. For a visible semantic reference, `Locate` uses a vision-language model (VLM) to identify its pixel location in a rendered 2D image. The camera parameters define a ray through this pixel. Raycasting against the scene geometry gives a 3D surface point that serves as a spatial anchor. References requiring geometric construction are passed to the program together with their descriptions and available anchors. The descriptions specify the intended geometry, while the anchors provide locations from which to examine it.



Figure 2: **From a spatial instruction to an object pose.** Grounding identifies controller grips and stand supports. The evidence program stores their geometry in $\mathcal{R}$ and records the intended relation $\mathcal{C}$. Guided by that relation, the pose program computes a candidate pose from $\mathcal{R}$, the scene, and the current pose.

3.2 Voxel Programming

Generated programs use local voxel views as a working representation of scene geometry. A view $U = \texttt{VoxelView}(s, h)$ is constructed from source geometry s at voxel size $h > 0$ and represents the cells that intersect its surface. The returned mesh proxy organizes this geometry on a regular grid and exposes coordinates and geometric attributes. Programs can use coordinate conditions and neighborhood relations to reason about the geometry relevant to the instruction. A known transform relates the view to the scene, so selected regions can be located in world coordinates.

Each task-relevant geometric selection forms a *witness* in the evidence record $\mathcal{R}$. For a geometric witness v, the program constructs a local view U_v from source geometry s_v at voxel size h_v, and computes a binary mask m_v selecting its surface cells:

$$U_v = \texttt{VoxelView}(s_v, h_v), \qquad W_v = \{c \in U_v : m_v(c) = 1\}. \tag{1}$$

The mask is computed by the generated program and can combine coordinate conditions, spatial masks, and local adjacency, using visual anchors and previously constructed witnesses where needed. The record stores the selected geometry, its source object and voxel view, and the references used in its construction. The pose program can then derive task-dependent positions, directions, and surface measurements from this geometry.

Voxel size is a program-controlled choice because the spatial detail needed for reasoning varies across computations. Fine resolution over all available geometry would enlarge the working representation even when only a small region needs that detail. Local views allow the program to use finer resolution for precise computations and coarser representations elsewhere. Each view is derived from the source mesh, which remains available for surface queries and for constructing another view at a different resolution. Coarse views can lose narrow structures, so the selected resolution must retain the detail required by the instruction.

3.3 Evidence Program Generation and Refinement

Given the instruction, evidence graph $\mathcal{G}$, scene $\mathcal{S}$, and available visual anchors $\mathcal{A}$, the LLM generates an executable evidence program P_{e}. The program constructs the required witnesses and emits an evidence record together with the intended placement relations:

$$(\mathcal{R}, \mathcal{C}) = P_{\mathrm{e}}(\mathcal{S}, \mathcal{A}), \tag{2}$$

where $\mathcal{R}$ stores the constructed geometry and $\mathcal{C}$ describes the desired placement in terms of its witnesses. These descriptions identify which regions participate in the placement and how they

should relate in the intended outcome. Construction follows the dependencies in $\mathcal{G}$, allowing later computations to use previously constructed witnesses.

Before submitting a complete candidate, the model states the geometric conditions each witness should satisfy, including relevant structure and geometry to exclude. This plan connects the intended role of each witness to the program that constructs it and identifies questions requiring inspection.

When a geometric question remains unresolved, the model can execute a probe on the current scene. A probe returns numerical outputs and rendered selections, allowing the model to inspect intermediate geometry and revise its construction. Exploration and the complete evidence program use the same voxel interface for selecting regions and querying their geometry.

The complete candidate program is executed, and geometric checks and a vision-language critic assess the witnesses against their stated roles in the instruction. Feedback identifies witnesses requiring revision and supplies observations for refining the program.

3.4 Pose Program Generation and Refinement

Given the evidence record $\mathcal{R}$ and placement relations $\mathcal{C}$, the LLM generates an executable pose program P_{p}. It determines how the moved object should be positioned and oriented in the scene, deriving the required spatial quantities from the selected regions. For one moved object, executing the program returns a candidate pose:

$$T = P_{\mathrm{p}}(\mathcal{R}, \mathcal{S}, T_{\mathrm{cur}}), \qquad T \in SE(3), \tag{3}$$

where T_{cur} is the object's current world-frame pose and $\mathcal{S}$ provides access to scene geometry and initial poses. The placement relations guide program generation; the program determines which measurements to compute and how to use them to obtain T.

Each witness also gives access to the source surface covered by its selected voxels. This surface is obtained by clipping the source mesh to the selection, preserving source-face correspondence. Measurements that depend on continuous surface geometry can therefore use the relevant source patch to compute geometric quantities on the original mesh surface.

The program returns an absolute world-frame pose, written as a homogeneous transformation T. Evidence geometry is expressed in the initial scene state. For an object with initial pose T_0, its initial world-space point $\mathbf{x}$ is evaluated under the candidate pose as

$$\widetilde{\mathbf{x}}' = T T_0^{-1} \widetilde{\mathbf{x}}, \tag{4}$$

where the tilde denotes homogeneous coordinates. This preserves the reference state of the evidence as candidate poses are revised, keeping their evaluation tied to the same initial geometry.

The candidate pose is applied to the scene, and the model receives a rendered observation together with auxiliary numerical diagnostics. It can revise the pose program in response to this feedback.

4 Experiments

We evaluate VoxProg on language-guided 3D object placement across functional, geometric, and contextual requirements. We describe the setup in §4.1, compare placement methods in §4.2, examine three geometric programming approaches and external placement tasks in §4.3, and analyze failure profiles in §4.4 through physical rejection categories and a visually ambiguous contact case.

4.1 Experimental setup

Benchmark. We evaluate on VoxProg-Bench, a benchmark of 71 individually curated language-guided 3D placement tasks. Each case presents an initial tabletop scene and a free-form instruction specifying the desired placement of an object. We emphasize diverse instructions requiring fine-grained reasoning about functional object parts, geometric fit, and contextual arrangements. An instruction can involve multiple spatial requirements, but each case is assigned to one group according to its primary spatial requirement for the comparisons reported below:

- **Functional:** The placement must support a specific use, such as placing a cup beneath a grinder chute to catch grounds, returning headphones to a stand, or orienting a funnel for pouring.

Method	IA Rank Score ↑				Physical Validity (%) ↑			
	Overall	Func.	Geom.	Context.	Overall	Func.	Geom.	Context.
GenManip	25.59	25.00	32.14	22.99	73.2	67.9	92.9	69.0
GenSim2-kPAM	42.72	34.52	35.71	<u>54.02</u>	77.5	<u>78.6</u>	85.7	72.4
Direct Mesh	<u>48.59</u>	<u>53.57</u>	<u>44.05</u>	45.98	<u>87.3</u>	75.0	**100.0**	<u>93.1</u>
VoxProg	**83.10**	**86.90**	**88.10**	**77.01**	**95.8**	**92.9**	**100.0**	**96.6**

Table 1: **Task-suite comparison.** Both metrics use 71 matched cases (Functional/Geometric/Contextual: 28/14/29). Bold/pale fill marks maxima; underline marks runners-up below a unique maximum.

- **Geometric:** The placement depends directly on object contours, such as nesting an oval bowl inside a matching container or fitting a baking tray into a narrow box at an angle.
- **Contextual:** The placement takes its cues from the surrounding arrangement, such as completing a missing place setting or restoring a symmetric layout.

The 71 cases comprise 28 Functional, 14 Geometric, and 29 Contextual cases.

Scenes are assembled from generated 3D assets through randomized placement and manual refinement. Detailed instructions, object geometry, and scene context make the intended final arrangement well defined; we provide one manually annotated ground-truth pose per case. Construction and annotation details appear in Appendix §A.

Evaluation suite. We design an evaluation suite around two complementary requirements of object placement: realizing the intended spatial arrangement and maintaining physical plausibility. We report these as Intent Alignment and Physical Validity, using visual comparison with ground truth and geometric checks, respectively. Implementation details are provided in Appendix §B.

Intent Alignment measures perceptual similarity between renderings of the predicted and ground-truth placements. Within each case, we order methods by perceptual similarity to the ground-truth image and assign a score to every rank. Averaging these scores gives the *IA Rank Score* (0–100; higher is better), which measures how consistently a method outranks its competitors across tasks. Each case contributes equally, so a small number of large perceptual distances cannot dominate the aggregate. The score preserves the methods' relative order and discards distance magnitudes.

Physical Validity assesses whether the final placement has plausible static support and avoids unacceptable interpenetration with surrounding objects. We report the percentage of evaluated placements that pass these geometric checks.

Baselines. We compare VoxProg against three language-guided placement approaches. GenManip (Gao et al., 2025) constructs placements using predefined spatial relations between target and anchor objects. GenSim2-kPAM (Hua et al., 2025; Manuelli et al., 2022) specifies placement targets through predicted semantic keypoints together with geometric point and axis constraints. Direct Mesh Placement uses the Blender-MCP interface (Ahuja, 2025) to generate Blender Python code that computes and executes a placement. Input details appear in Appendix §A, and the compared methods are described in Appendix §C.

Model configuration. All compared methods use GPT-5.6-Sol with a shared model configuration for both language and vision processing.

4.2 Results on the task suite

Quantitative results. VoxProg achieves the highest IA Rank Score among the compared placement methods, both overall and in each of the three task groups (Table 1). Its overall score is **83.10**, compared with 48.59 for Direct Mesh, 42.72 for GenSim2-kPAM, and 25.59 for GenManip. Physical Validity reaches 95.8%, compared with 87.3% for Direct Mesh, 77.5% for GenSim2-kPAM, and 73.2% for GenManip. VoxProg also improves over Direct Mesh, which already generates task-specific code from explicit scene geometry. Its formulation organizes this computation around geometric references constructed for the instruction.

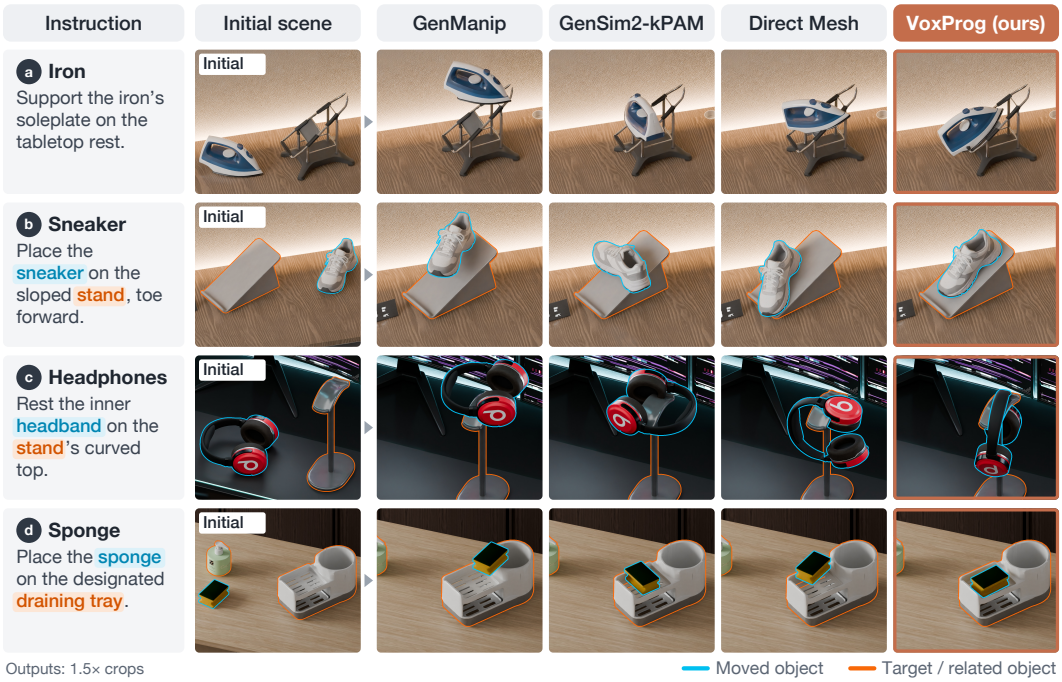

Figure 3: **Qualitative placement comparisons.** Rows (a–d) show iron, sneaker, headphone, and sponge placement, respectively, with their initial scenes in the first image column. All panels are cropped to a common aspect ratio; initial views retain both objects, and method outputs share a $1.5\times$ crop within each row. Cyan/orange mark moved/reference objects in (b–d); terracotta highlights VoxProg.

On Functional tasks, VoxProg obtains **86.90** against 53.57 for Direct Mesh and 34.52 for GenSim2-kPAM. These tasks express the goal through intended use, requiring the method to determine which object parts matter and how they should meet. The advantage on this group is consistent with the value of constructing geometry conditioned on the task. The strongest baseline changes on Contextual tasks, where GenSim2-kPAM outperforms Direct Mesh, while VoxProg retains the highest score at 77.01. On Geometric tasks, Direct Mesh and VoxProg score 44.05 and 88.10, respectively, yielding the largest separation from the strongest baseline. Across these different demands, VoxProg combines the ability to derive targets from scene relations with access to the local geometry needed to realize them through the moved object's position and orientation.

Qualitative results. Fig. 3 illustrates the part-level distinctions behind these placements. VoxProg supports the iron on its tabletop rest, seats the sneaker on the sloped stand with its toe facing forward, aligns the inner headband with the stand's curved top, and places the sponge within the designated tray region. Each instruction refers to a particular part or surface whose role must be interpreted in the scene. The results illustrate how task-dependent choices of region and orientation shape the final configuration, even when the relevant objects are readily identifiable. Additional task-suite comparisons are provided in Appendix §D.

4.3 Ablation and External Evaluation

Effect of voxel programming. We evaluate the contribution of voxel programming by replacing local voxel views with mesh-based geometry construction within the same staged pipeline (Table 2).

Direct Mesh generates a placement program over the original meshes through the Blender-MCP interface (Ahuja, 2025). Mesh Evidence instead constructs regions from mesh vertices and faces and passes them to VoxProg's pose stage. VoxProg uses the same staged formulation with local voxel views, with access to the corresponding source surfaces during pose computation for geometric measurements within each selected region.

Within this three-method comparison,IA Rank Score increases from 26.76 with Direct Mesh to 56.34 with Mesh Evidence and **66.90** with VoxProg.

Mesh Evidence and VoxProg both achieve 95.8% Physical Validity, compared with 87.3% for Direct Mesh. Both staged variants improve on Direct Mesh, supporting the use of explicit region construction before pose synthesis to expose the surfaces relevant to placement.

The additional IA gain, with Physical Validity unchanged, suggests that voxel views help specify the intended arrangement beyond satisfying support and interpenetration requirements.

Fig. 4 contrasts tablet placement on the same stand. VoxProg places the tablet in front with its screen facing forward, whereas Mesh Evidence places it behind the support structure.

Method	IA Rank Score ↑	Physical Validity (%) ↑
Direct Mesh	26.76	87.3
Mesh Evidence	56.34	**95.8**
VoxProg	**66.90**	**95.8**

Table 2: Ablation of voxel programming.



Figure 4: **Different placements on the same stand.** Tablet placements from the ablation variants, shown with the same view and crop.

External placement tasks. We evaluate on RoboCasa's pick-and-place and insertion tasks (Nasiriany et al., 2024) and a subset of Open6DOR v2 (Ding et al., 2024; Qi et al., 2025), adapting the source scenes and task goals into final-state object-placement problems (Fig. 5). All methods use the same evaluation cases. On RoboCasa, VoxProg obtains an IA Rank Score of **87.72**, compared with baseline scores ranging from 26.32 to 45.61. On Open6DOR v2, it scores **61.54**, compared with 57.69 for Direct Mesh. Physical Validity is 94.7% and 92.3%, respectively. The smaller IA margin may reflect this subset's broad, simple spatial goals, which leave more latitude in the final pose and less scope for detailed geometry construction to distinguish methods. Adaptation and evaluation details are provided in Appendix §E.

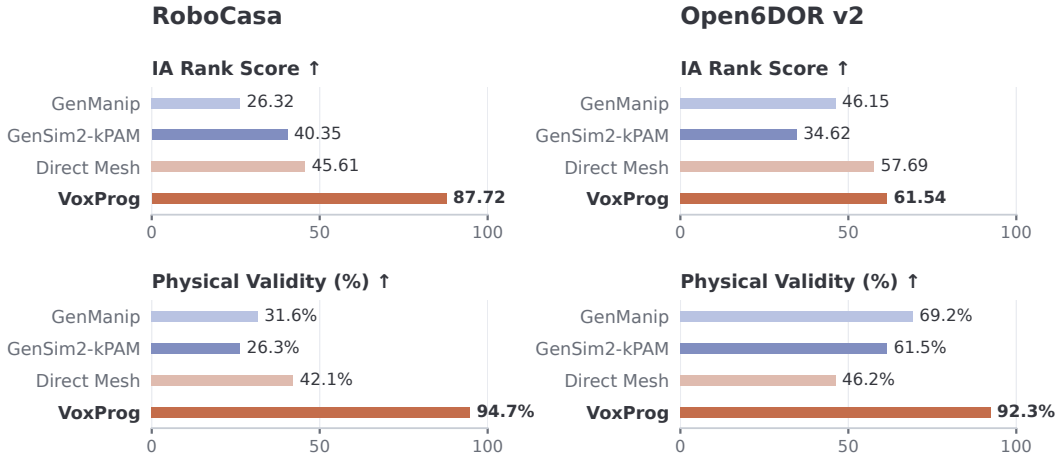

Figure 5: **External placement evaluation.** IA Rank Score (0–100; ↑) and Physical Validity (Valid %; ↑) on tasks adapted from RoboCasa and Open6DOR v2. All methods and both metrics use 19 RoboCasa and 13 Open6DOR v2 cases, with matched inputs across methods.

The qualitative results in Fig. 6 illustrate spatial references across these external tasks. VoxProg brings the mug beneath the coffee outlet, places the cereal box within the cabinet, and places the SD card on top of the USB drive. These tasks require identifying a functional location on an appliance, an interior storage region, and a supporting object surface, respectively.

4.4 Failure Analysis

Physical failure profiles. Support failures are prominent for GenManip, whereas interpenetration dominates GenSim2-kPAM (Fig. 7a). These categories distinguish contact with a supporting surface



Figure 6: **External placement comparisons.** The top two rows show mug and cereal placement from RoboCasa; the bottom row shows SD-card placement from Open6DOR v2. Initial scenes are cropped, and outputs share a crop within each row. Cyan marks the moved object and instruction terms; orange marks the goal reference; terracotta highlights the placements produced by VoxProg.

from clearance around the object's full extent. Direct Mesh exhibits both despite access to complete meshes. Both staged variants reduce physical rejections from nine to three, suggesting that making the relevant surfaces explicit helps pose computation account for these local geometric requirements.

Visually ambiguous contact. The pot-lid example in Fig. 7b reveals a support failure that is difficult to recognize from the original view. The lid is upright, its handle faces outward, and its rim projects onto the rack, all suggesting that the requested arrangement has been achieved. A side view, however, shows that the lid remains separated from both the tray and the inclined backing. The rack provides a strong cue to the intended destination while obscuring the region where support should be established. Such configurations separate the visible arrangement from the contact needed to realize the intended use. A visual check can therefore miss a physical failure even when the relevant objects and their intended relation are recognizable. Here, overlap in the image does not establish that the lid's rim actually meets a supporting surface in 3D. Geometric verification complements this visual evidence by checking the task-relevant contact surfaces, making the distinction between apparent seating and actual support explicit in the scene's three-dimensional geometry.

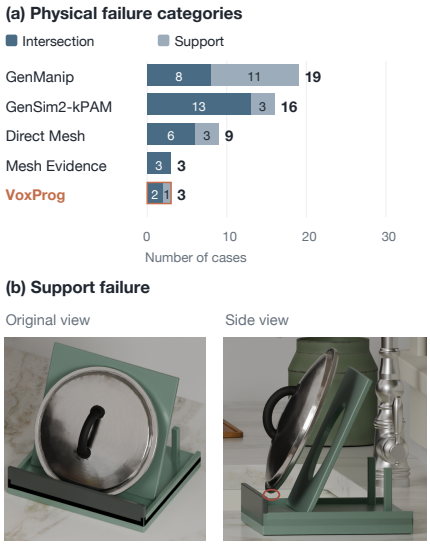

Figure 7: Physical failure profiles and a visually ambiguous support failure.

5 Conclusion

We presented VoxProg, a framework that expresses spatial task interpretations as programs for constructing geometry and computing object poses. Local voxel views organize scene surfaces into a regular working representation, allowing generated programs to derive the regions and boundaries that define a placement. This makes the geometric references of a task part of the computation itself. Across the functional, geometric, and contextual tasks in VoxProg-Bench, VoxProg outperforms the baselines in IA Rank Score. The ablation separates two effects: constructing regions before pose synthesis improves both evaluation metrics over Direct Mesh, while voxel views further improve overall Intent Alignment across the evaluated placement tasks.

AI USE STATEMENT

We used generative AI tools to create synthetic data and 3D assets for VoxProg-Bench: text-to-image generation produced concept images, and multiview image-to-3D generation reconstructed individual objects as meshes. Scene layouts were manually refined, and ground-truth poses were manually annotated, as described in Appendix §A. We also used generative AI tools to assist with writing and language polishing, LaTeX editing and document formatting, and figure preparation and visual refinement. The authors take responsibility for the final content of this paper, including its technical claims and reported results.

REPRODUCIBILITY STATEMENT

Section 3 describes the geometric representation and the evidence and pose programs. Appendix §A documents benchmark construction, inputs, and ground-truth annotations; Appendix §B describes the evaluation protocol and score aggregation. The compared methods and external task adaptations are described in Appendices §C and §E, respectively. Core prompt excerpts are provided in the supplementary document *VoxProg: Core Prompts*. Code will be released upon acceptance.

REFERENCES

Ahmed Abdelreheem, Filippo Aleotti, Jamie Watson, Zawar Qureshi, Abdelrahman Eldesokey, Peter Wonka, Gabriel Brostow, Sara Vicente, and Guillermo Garcia-Hernando. PlaceIt3D: Language-guided object placement in real 3D scenes. In *ICCV*, pp. 6645–6655, 2025.

Siddharth Ahuja. Blender-MCP. `https://github.com/ahujasid/mcp-for-blender`, 2025.

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics*, 37(4):43:1–43:12, 2018.

Haonan Chang, Kai Gao, Kowndinya Boyalakuntla, Alex Lee, Baichuan Huang, Jingjin Yu, and Abdeslam Boularias. LGMCTS: Language-guided monte-carlo tree search for executable semantic object rearrangement. In *IROS*, pp. 13607–13612, 2024.

Minglin Chen, Rongkun Yang, Qibin Hu, Kaiwen Xue, Shunbo Zhou, and Yulan Guo. Graph2Scene: Versatile 3D indoor scene generation with interaction-aware scene graph. In *IROS*, pp. 11313–11320, 2025.

Zeren Chen, Xiaoya Lu, Zhijie Zheng, Pengrui Li, Lehan He, Yijin Zhou, Jing Shao, Bohan Zhuang, and Lu Sheng. Geometrically-constrained agent for spatial reasoning. In *CVPR*, pp. 38689–38699, 2026.

Yufei Ding, Haoran Geng, Chaoyi Xu, Xiaomeng Fang, Jiazhao Zhang, Songlin Wei, Qiyu Dai, Zhizheng Zhang, and He Wang. Open6DOR: Benchmarking open-instruction 6-DoF object rearrangement and a VLM-based approach. In *IROS*, pp. 7359–7366, 2024.

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. In *NeurIPS*, pp. 18225–18250, 2023.

Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. DreamSim: Learning new dimensions of human visual similarity using synthetic data. In *NeurIPS*, pp. 50742–50768, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *ICML*, pp. 10764–10799, 2023.

Ning Gao, Yilun Chen, Shuai Yang, Xinyi Chen, Yang Tian, Hao Li, Haifeng Huang, Hanqing Wang, Tai Wang, and Jiangmiao Pang. GenManip: LLM-driven simulation for generalizable instruction-following manipulation. In *CVPR*, pp. 12187–12198, 2025.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *CVPR*, pp. 14953–14962, 2023.

Pu Hua, Minghuan Liu, Annabella Macaluso, Yunfeng Lin, Weinan Zhang, Huazhe Xu, and Lirui Wang. GenSim2: Scaling robot data generation with multi-modal and reasoning LLMs. In *CoRL*, pp. 5030–5066, 2025.

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 *CVPR*, pp. 13466–13476, 2025a.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *CoRL*, pp. 540–562, 2023.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *CoRL*, pp. 4573–4602, 2025b.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics*, 32(4):33:1–33:12, 2013.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *ICRA*, pp. 9493–9500, 2023.

Chenguo Lin and Yadong Mu. InstructScene: Instruction-driven 3D indoor scene synthesis with semantic graph prior. In *ICLR*, 2024.

Zefu Lin, Rongxu Cui, Hanning Chen, Xiangyu Wang, Junjia Xu, Xiaojuan Jin, Wenbo Chen, Hui Zhou, Lue Fan, Wenling Li, and Zhaoxiang Zhang. EmbodiedCoder: Parameterized embodied mobile manipulation via modern coding model. *arXiv preprint arXiv:2510.06207*, 2025.

Weiyu Liu, Chris Paxton, Tucker Hermans, and Dieter Fox. StructFormer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *ICRA*, pp. 6322–6329, 2022.

Weiyu Liu, Yilun Du, Tucker Hermans, Sonia Chernova, and Chris Paxton. StructDiffusion: Language-guided creation of physically-valid structures using unseen objects. In *RSS*, 2023.

Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kPAM: Keypoint affordances for category-level robotic manipulation. In *Robotics Research*, pp. 132–157. Springer, 2022.

Yao Mu, Junting Chen, Qing-Long Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, Peize Sun, Haibao Yu, Chao Yang, Wenqi Shao, Wenhai Wang, Jifeng Dai, Yu Qiao, Mingyu Ding, and Ping Luo. RoboCodeX: multimodal code generation for robotic behavior synthesis. In *ICML*, pp. 36434–36454, 2024.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of household tasks for generalist robots. In *RSS*, 2024.

Başak Melis Öcal, Maxim Tatarchenko, Sezer Karaoğlu, and Theo Gevers. SceneTeller: Language-to-3D scene generation. In *ECCV*, pp. 362–378, 2024.

Mingjie Pan, Jiyao Zhang, Tianshu Wu, Yinghao Zhao, Wenlong Gao, and Hao Dong. OmniManip: Towards general robotic manipulation via object-centric interaction primitives as spatial constraints. In *CVPR*, pp. 17359–17369, 2025.

Zekun Qi, Wenyao Zhang, Yufei Ding, Runpei Dong, Xinqiang Yu, Jingwen Li, Lingyun Xu, Baoyu Li, Xialin He, Guofan Fan, Jiazhao Zhang, Jiawei He, Jiayuan Gu, Xin Jin, Kaisheng Ma, Zhizheng Zhang, He Wang, and Li Yi. SoFar: Language-grounded orientation bridges spatial reasoning and object manipulation. In *NeurIPS*, pp. 85008–85053, 2025.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *ICRA*, pp. 11523–11530, 2023.

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 *CVPR*, pp. 29469–29478, 2025.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via Python execution for reasoning. In *ICCV*, pp. 11888–11898, 2023.

Weiliang Tang, Jia-Hui Pan, Yun-Hui Liu, Masayoshi Tomizuka, Li Erran Li, Chi-Wing Fu, and Mingyu Ding. GeoManip: Geometric constraints as general interfaces for robot manipulation. *arXiv preprint arXiv:2501.09783*, 2025.

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 *CVPR*, pp. 16227–16237, 2024.

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. In *NeurIPS*, pp. 30026–30038, 2023.

Zhihao Zhu, Yifan Zheng, Siyu Pan, Yaohui Jin, and Yao Mu. PASG: A closed-loop framework for automated geometric primitive extraction and semantic anchoring in robotic manipulation. In *ICCV*, pp. 8950–8960, 2025.

A Task Suite and Inputs

A.1 Task definition

Our task suite consists of language-guided placement cases. Each case provides a tabletop scene, a movable object, and an instruction specifying the desired arrangement. The output is a rigid final pose of the movable object. Evaluation focuses on the resulting object configuration.

The suite groups cases by their primary spatial requirement. **Functional** cases specify placement through intended use, such as returning headphones to a stand or positioning a cup to catch coffee grounds. **Geometric** cases require reasoning about shape, containment, or clearance, such as nesting oval bowls or fitting a stemmed bowl beneath a shelf. **Contextual** cases require inferring the target from surrounding arrangements, such as completing a place setting or restoring symmetry. These requirements can overlap; each case has one primary group for reporting. The full 71-case inventory contains 28 Functional, 14 Geometric, and 29 Contextual cases.

A.2 Benchmark construction

For each case, we first specify a placement instruction and the intended scene. A text-to-image model generates a concept image of the desired arrangement, from which individual objects are isolated and reconstructed as 3D meshes using multiview image-to-3D generation. We assemble the assets into a tabletop scene by randomly sampling initial positions and orientations, then manually refine the layout to keep objects recognizable and establish the scene context required by the instruction, including the surrounding objects that define the placement goal.

A.3 Inputs and ground-truth annotations

Scene meshes, initial poses, the instruction, and the mover identity define the placement problem. Object labels are available in the controlled benchmark. Each case has one manually annotated ground-truth pose, specifying the mover's target position and orientation in world coordinates. The ground-truth pose and its rendered image are used only for evaluation and are not provided to the placement methods when generating or refining a candidate pose.

B Evaluation Protocol

B.1 Intent Alignment

Intent Alignment uses the perceptual distance between the ground-truth image I^* and a method's final-state image I. We compute this distance with DreamSim (Fu et al., 2023), using the default pretrained ensemble in version 0.2.1 (official implementation). The scorer converts both images to RGB, resizes them to 512×512 with Lanczos resampling, applies the official preprocessing function, and computes feature embeddings $f(I)$ and $f(I^*)$. The distance is

$$d_{\mathrm{DS}}(I, I^*) = 1 - \frac{f(I)^\top f(I^*)}{\|f(I)\|_2 \, \|f(I^*)\|_2}. \tag{5}$$

Lower values in Eq. 5 indicate greater perceptual similarity to the ground-truth image. The scorer uses the full rendered image without an object mask or additional crop. Predicted and ground-truth scenes use matched cameras and lighting.

We report one evaluated run per method and case.

For reported Intent Alignment, we rank methods by increasing DreamSim distance separately within each case. With N evaluated cases and M compared methods, let $r_{n,m}$ denote the rank of method m on case n, using average ranks for ties. The reported score is

$$\text{IA Rank Score}(m) = \frac{1}{N} \sum_{n=1}^{N} 100 \frac{M - r_{n,m}}{M - 1}. \tag{6}$$

Each case has equal weight. The score lies between 0 and 100 and is equivalent to the average fraction of competing methods outperformed, counting ties as half a win. Main and external comparisons use four methods; the ablation uses Direct Mesh, Mesh Evidence, and VoxProg. The main

and ablation comparisons both use 71 cases for both IA and PV. Scores depend on the comparison set and are not directly comparable between these settings. Ranking preserves relative order but discards distance magnitudes between predicted and ground-truth images.

B.2 Physical Validity

Physical Validity evaluates whether a final placement has plausible support and avoids excessive interpenetration. We apply the predicted pose to the scene meshes and assess the moved object relative to the surrounding geometry. Surface samples and generalized winding numbers (Jacobson et al., 2013; Barill et al., 2018) identify penetration, while local intersection tests account for thin structures. Proximity and surface orientation identify candidate contacts. Their spatial distribution, together with the object's projected center, provides evidence of support; near-contact configurations are also assessed through the distance to the receiving surface.

Tolerances scale with object size and use the ground-truth placement as a baseline, allowing small mesh imperfections and contact gaps that also occur in the ground-truth placement. The resulting decision is Valid or Invalid. We report the percentage of valid placements among evaluated cases. The same support and interpenetration criteria apply to the task suite and external datasets.

C Compared Methods

GenManip (Gao et al., 2025) uses its relation-based placement component; GenSim2-kPAM (Hua et al., 2025; Manuelli et al., 2022) represents the target pose through semantic keypoints and point/axis constraints. Direct Mesh Placement generates code through the Blender-MCP interface (Ahuja, 2025) to query the mesh scene and modify the object transform. Mesh Evidence uses VoxProg's evidence and pose stages but selects witness regions through direct mesh operations. VoxProg constructs these regions through local voxel views and provides access to the source-mesh surfaces corresponding to the selected regions for subsequent pose computation.

Model configuration. All compared methods, including Mesh Evidence and VoxProg, use GPT-5.6-Sol with the same model configuration for language and vision processing. Mesh Evidence and VoxProg share the staged evidence and pose pipeline; their ablation changes how task-relevant regions are constructed, using direct mesh operations in Mesh Evidence and local voxel views in VoxProg.

Core prompt excerpts are provided in the supplementary document *VoxProg: Core Prompts* (`core-prompts.pdf`), covering the evidence and pose stages.

D Additional Qualitative Results

Figs. S1–S3 show 18 illustrative task-suite cases, with ground-truth placements and outputs from the four compared placement methods under the same camera view for each case. The supplementary material provides results for all 71 cases.

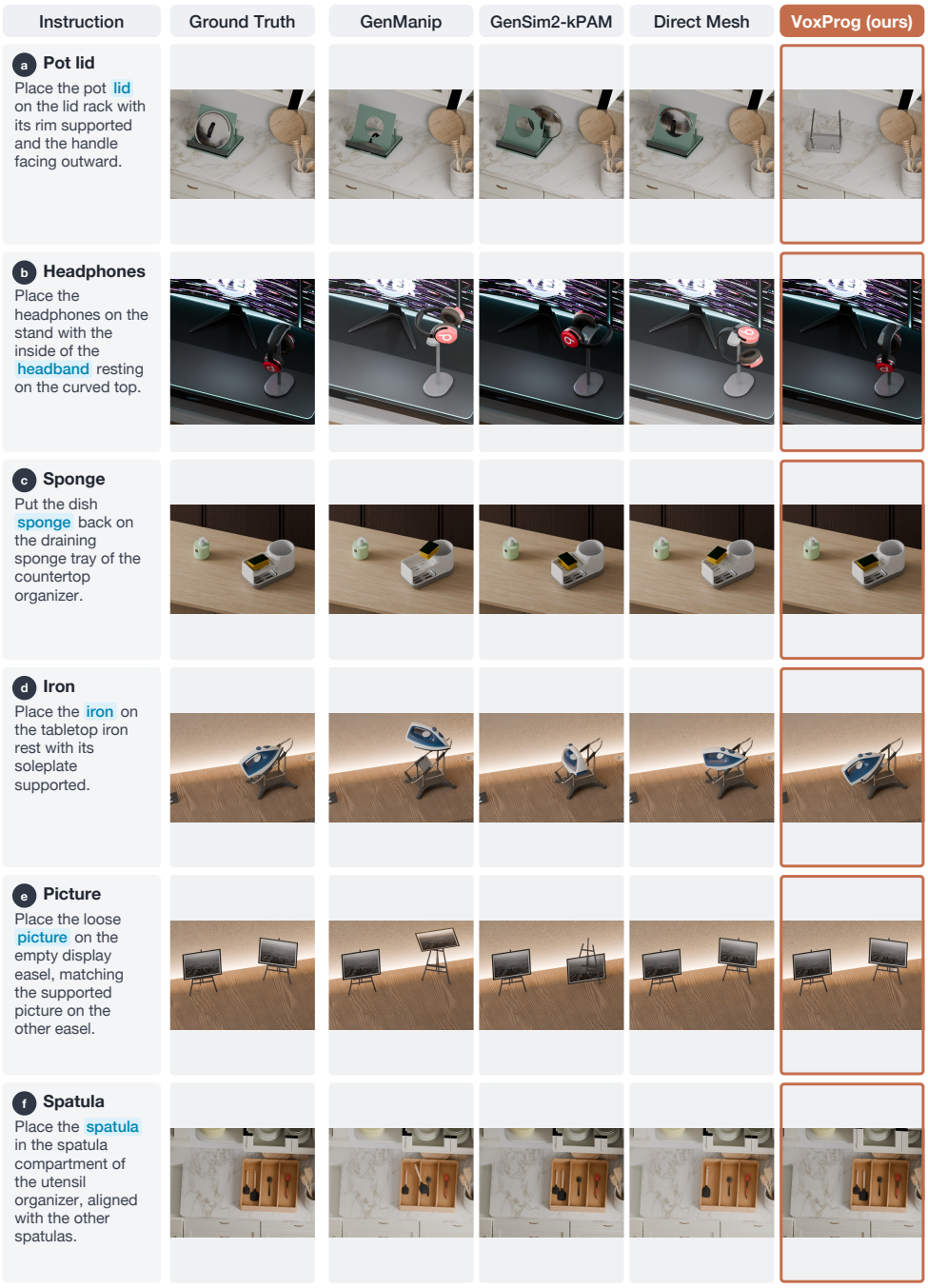

Figure S1: **Task-suite placement comparisons.** Each case shows its instruction, ground-truth placement, and predictions from GenManip, GenSim2-kPAM, Direct Mesh, and VoxProg. Terracotta frames highlight the placements produced by VoxProg for comparison with the other methods.

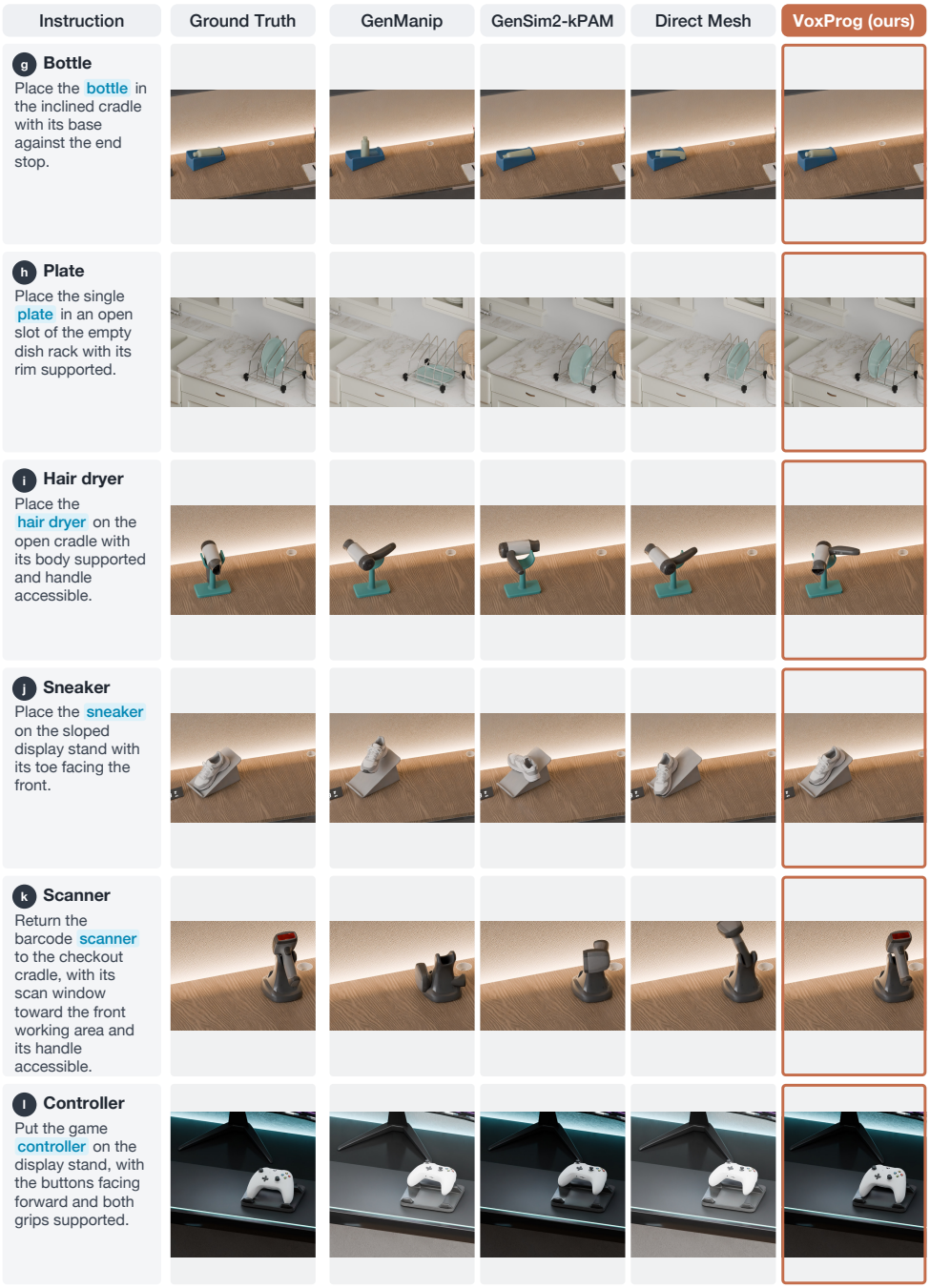

Figure S2: **Task-suite placement comparisons.** Each case shows its instruction, ground-truth placement, and predictions from GenManip, GenSim2-kPAM, Direct Mesh, and VoxProg. Terracotta frames highlight the placements produced by VoxProg for comparison with the other methods.

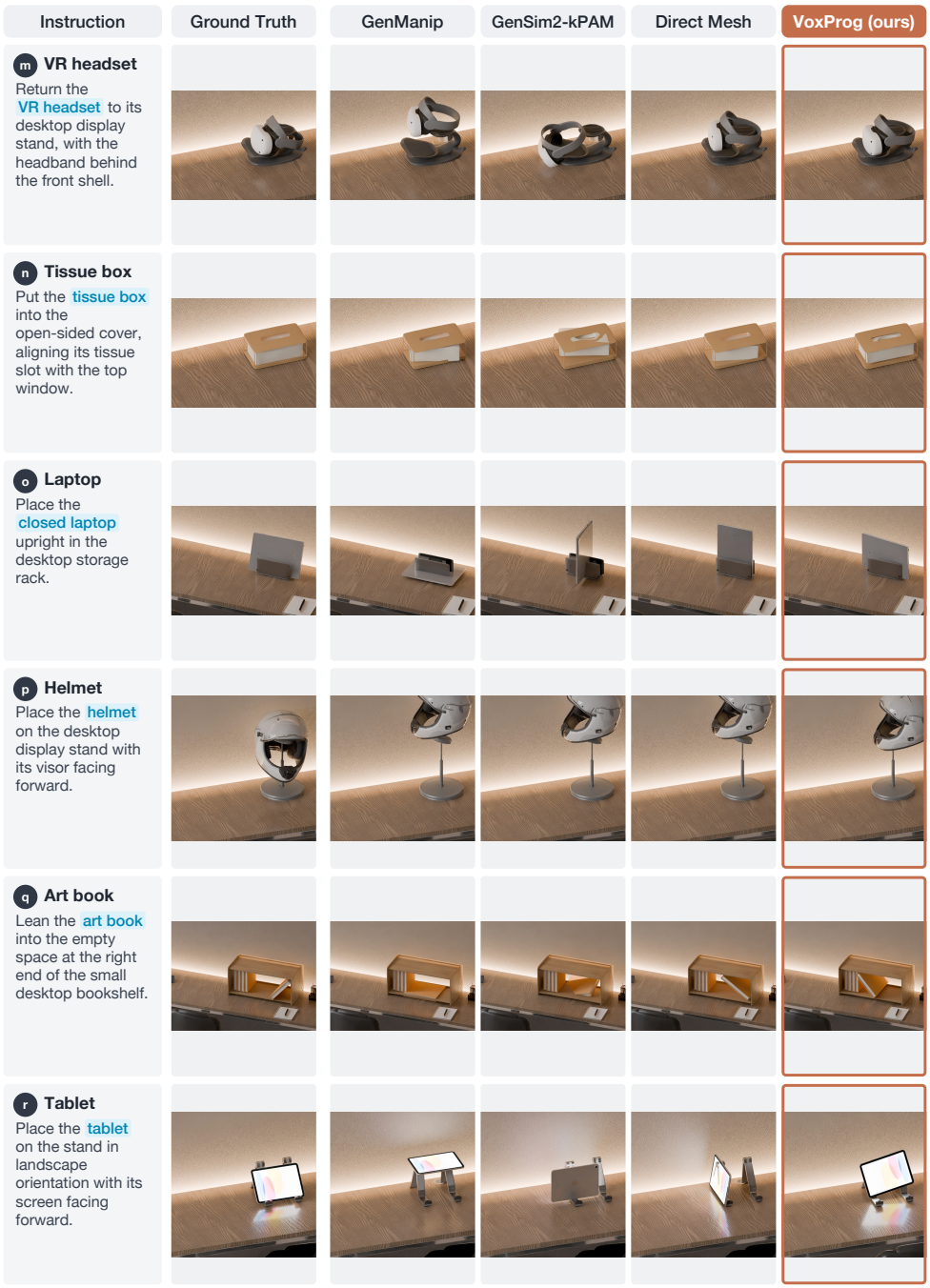

Figure S3: **Task-suite placement comparisons.** Each case shows its instruction, ground-truth placement, and predictions from GenManip, GenSim2-kPAM, Direct Mesh, and VoxProg. Terracotta frames highlight the placements produced by VoxProg for comparison with the other methods.

E External Task Sources and Adaptation

Our external tasks are adapted from placement and insertion cases in RoboCasa (Nasiriany et al., 2024) and Open6DOR v2 (Ding et al., 2024; Qi et al., 2025). We use their scene assets and task requirements to formulate final-state object-placement problems, evaluating the resulting placement rather than a robot's action trajectory. All compared methods receive the same task setup. Predicted and ground-truth end states are rendered with matched cameras and lighting, and both metrics use the same 19 RoboCasa and 13 Open6DOR v2 cases. Quantitative results are reported in Fig. 5. Qualitative examples from both datasets appear in Fig. 6.