

OrganizeIt: Compiling Generated Images into Tidy Tabletop Arrangements

Junshan Huang, Xiyu Ke, Jintong Li, Litao Liu, Duo Zhang, Jingjin Yu
Rutgers University

{junshan.huang, jintong.li, litao.liu, duo.zhang, jingjin.yu}@rutgers.edu
xk41@scarletmail.rutgers.edu

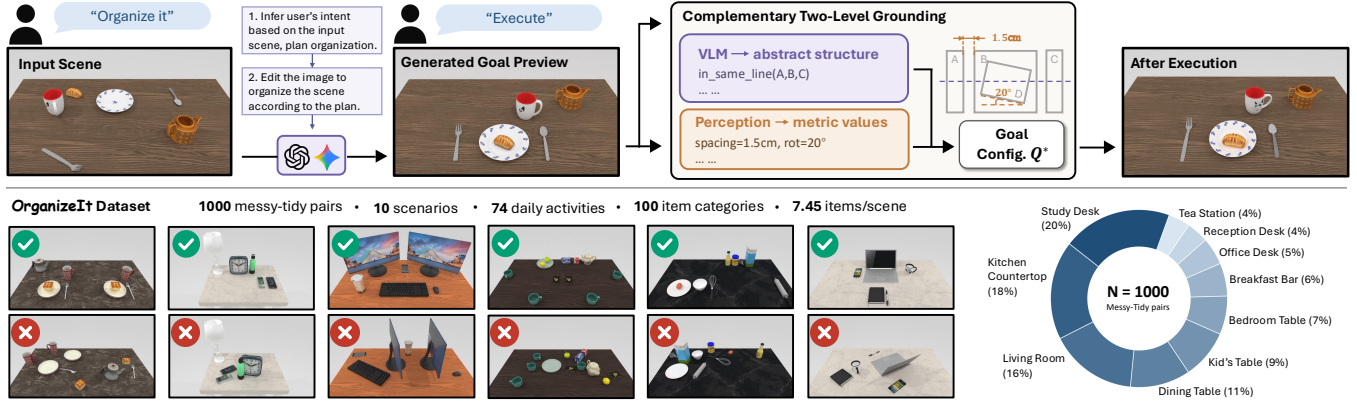


Figure 1: Upper: ORGANIZEIt turns a high-level request into robot execution by first generating a tidy goal image, then compiling it into an executable target configuration Q^* : a VLM reads relational structure from the goal image, while perception grounds its metric parameters against the real scene. Lower: the ORGANIZEIt dataset. 1,000 human-verified messy–tidy pairs spanning 10 scenario types, 74 daily activities, and 100 object categories, averaging 7.45 objects per scene. Each example pairs a tidy scene (green check) with a messy scene built from the same objects (red cross); the ring chart shows the scenario distribution.

Abstract

We present ORGANIZEIt, a framework harnessing broad visual and commonsense priors for open-ended organization, instantiated in tabletop settings. Given an RGB-D observation of a cluttered scene and a brief instruction (e.g., “organize it”), a vision–language model (VLM) infers the intent as an object-wise organization plan, which an image generator renders as a tidy preview; ORGANIZEIt then carries out the arrangement. The preview, however, commits in pixels rather than in directly usable poses. A key insight we establish is that tidiness is a joint function of object–object spatial relationships and each object’s individual placement and pose: registering objects independently recovers only the latter, and the uncoupled errors leave the depicted rows and alignments ragged. This insight motivates our core innovation: a dual-channel parsing of the preview. The VLM channel compiles the depicted relational structure into an arrangement program with open metric parameters; the perception channel fills them by registering reconstructed meshes against the preview, supplying the spacings, offsets, and off-axis orientations language cannot state precisely. Evaluating the completed program derives all target poses jointly, so placements and relations hold by construction, and the same program drives execution. For validation, we build the ORGANIZEIt dataset: 1,000 human-verified messy–tidy pairs spanning 74 daily activities, 100

object categories, and 766 assets, with an open-ended pairwise evaluation protocol. Three independent VLM judges all directionally prefer ORGANIZEIt over SG-Bot on its own 90-scene benchmark, by at least 17.8 tidiness points, and a human study confirms these verdicts track human judgment. Across 1,000 scenes, the full system’s tidiness win rate exceeds its loss rate against either single-channel ablation by at least 32.9 points, and real-robot trials in four tabletop domains demonstrate physical execution.

1 Introduction

Presented with a messy scene, a capable household robot should readily execute a request as simple as “organize it.” Yet such a request gives neither a target arrangement structure nor object-level placement constraints: what counts as acceptable depends on the activity, the objects, and the user’s intent, none of it stated verbatim. The robot must turn an underspecified instruction into a concrete multi-object goal and realize it precisely. Existing methods make goals executable through pre-specified relations, learned layout priors, or direct pose prediction (Liu et al. 2022b,a; Zhai et al. 2024; Xu et al. 2025). Open-ended requests add an additional challenge: a system must first decide what the target scene should

be. Image generation offers a natural bridge (Kapelyukh, Vosylius, and Johns 2023; Mendez et al. 2024; Chen et al. 2025a), rendering one concrete tidy alternative that commits to grouping, alignment, spacing, and orientation. Such an image, however, commits in pixels rather than poses and carries perspective and other inaccuracies: it shows what the result should look like, not where each real object should go.

The difficulty runs deeper than geometric imperfection. A key insight we establish is that *tidiness is a joint function of object–object spatial relationships and each object’s individual placement and pose*: a scene reads as tidy only when both hold at once, so neither suffices alone. If several pens are depicted as an evenly spaced row, registering each independently attends only to individual placement: every pen lands near its counterpart, yet the row comes out ragged because the errors are uncoupled. A VLM has the complementary limitation: it recognizes the row and its shared orientation, but not the exact spacing, offset, or off-axis angle execution requires. A generated goal thus holds two layers: *relational structure*, recovered by semantic reasoning, and *metric instantiation*, which requires geometric measurement; tidiness emerges only when the two are resolved together.

ORGANIZEIT treats a generated goal not as an image to imitate but as a program to compile (Figure 1). Given an RGB-D observation and an instruction, a VLM produces an object-wise organization plan that an image generator renders as a tidy preview. A dual-channel compiler then reads it: the VLM channel compiles the depicted relational structure into an arrangement program with metric parameters left open; the perception channel fills them by registering the real objects against the preview at true scale. Evaluating the completed program derives all target poses jointly, so the declared relations hold by construction rather than emerging from independent estimates. The same program then serves as a persistent contract for relation-preserving refinement, execution planning, and post-execution repair.

This paper makes three main contributions.

- We identify tidiness as a joint function of object–object spatial relations and individual object placement, and accordingly cast the grounding of generated visual goals as compilation, introducing a dual-channel compiler: a VLM extracts relational structure into an arrangement program, while perception resolves its open metric parameters, so evaluating it yields jointly constrained target poses.
- We develop ORGANIZEIT, an end-to-end system applied to zero-shot tabletop organization requiring no pre-existing CAD models: from a single RGB-D observation and an underspecified instruction, it generates an inspectable preview, compiles it into a relation-preserving arrangement, and executes that arrangement on a real robot. The persistent program supports metric grounding, refinement, dependency-aware execution, and repair.
- We introduce the ORGANIZEIT dataset and benchmark: 1,000 human-verified messy–tidy pairs across 74 daily activities, 100 object categories, and 766 assets, with a human-validated open-ended pairwise protocol.

Against SG-Bot (Zhai et al. 2024) on its own 90-scene benchmark, three independent VLM judges all directionally

prefer ORGANIZEIT on both tidiness and user intent, with tidiness margins of at least 17.8 points. On 1,000 benchmark scenes, its tidiness win rate exceeds its loss rate against either single-channel variant by at least 32.9 points. A human study validates the protocol: the primary VLM judge is statistically equivalent to a human rater, and all twelve judges, human and VLM, place ORGANIZEIT above every baseline and below the human-authored oracle. Real-robot trials across four tabletop domains confirm end-to-end execution.

2 Related Work

Generative models as goal specification Generative models increasingly specify robot goals in image or video space. Prior work recovers object-level targets from a generated goal image by matching, pose estimation, or 3D registration (Kapelyukh, Vosylius, and Johns 2023; Mendez et al. 2024; Jin et al. 2025; Chen et al. 2025a; Wang et al. 2026), renders and scores imagined arrangements in a reconstructed 3D scene (Kapelyukh et al. 2024), generates visual subgoals for goal-conditioned control (Black et al. 2024; Liu et al. 2026), or turns video plans into actions via inverse dynamics or policies (Du et al. 2023; Ajay et al. 2023; Du et al. 2024). These make generated states actionable, but their executable representations are local transformations or ordered plans, which leave the joint structure of a complete arrangement implicit. ORGANIZEIT instead reads one image as a specification of the whole arrangement, compiling its relational structure into a program that perception grounds.

Semantic rearrangement priors Semantic rearrangement needs priors to infer desirable configurations from underspecified goals. Prior work learns layout distributions from task-specific data (Liu et al. 2022b,a; Zeng et al. 2024; Wei et al. 2023), inherits commonsense priors from image generators (Kapelyukh, Vosylius, and Johns 2023), LLMs (Wu et al. 2023), and VLMs (Chen et al. 2025b), encodes structure through relation graphs or programs (Zhai et al. 2024; Xu et al. 2025; Ding et al. 2023), or searches arrangements under geometric or learned tidiness objectives (Chang et al. 2024; Kee et al. 2025). Embodied tidying benchmarks encode similar commonsense priors (Kant et al. 2022; Sarch et al. 2022), and LLMs and VLMs can write executable plans or programs directly (Ahn et al. 2022; Liang et al. 2023; Singh et al. 2023; Huang et al. 2023), though ORGANIZEIT’s program declares the goal arrangement rather than the action sequence. ORGANIZEIT combines two priors: a VLM resolves the instruction into an object-wise organization plan, supplying semantic structure, and an image generator instantiates that plan as a tidy arrangement, supplying fine-grained metric cues.

Evaluating tidiness Prior work evaluates tidiness using target-matching metrics (Zhai et al. 2024; Hu et al. 2025), predefined semantic or physical constraints (Liu et al. 2022b,a; Chang et al. 2024), human judgments and task-specific checks (Kapelyukh, Vosylius, and Johns 2023; Zeng et al. 2024; Xu et al. 2025), or learned discriminators (Kee et al. 2025). Automated measures scale but encode restricted notions of organization; human judgments admit multiple valid organizations at higher cost. VLMs now serve as scalable judges (Zhang et al. 2023), and pairwise comparison

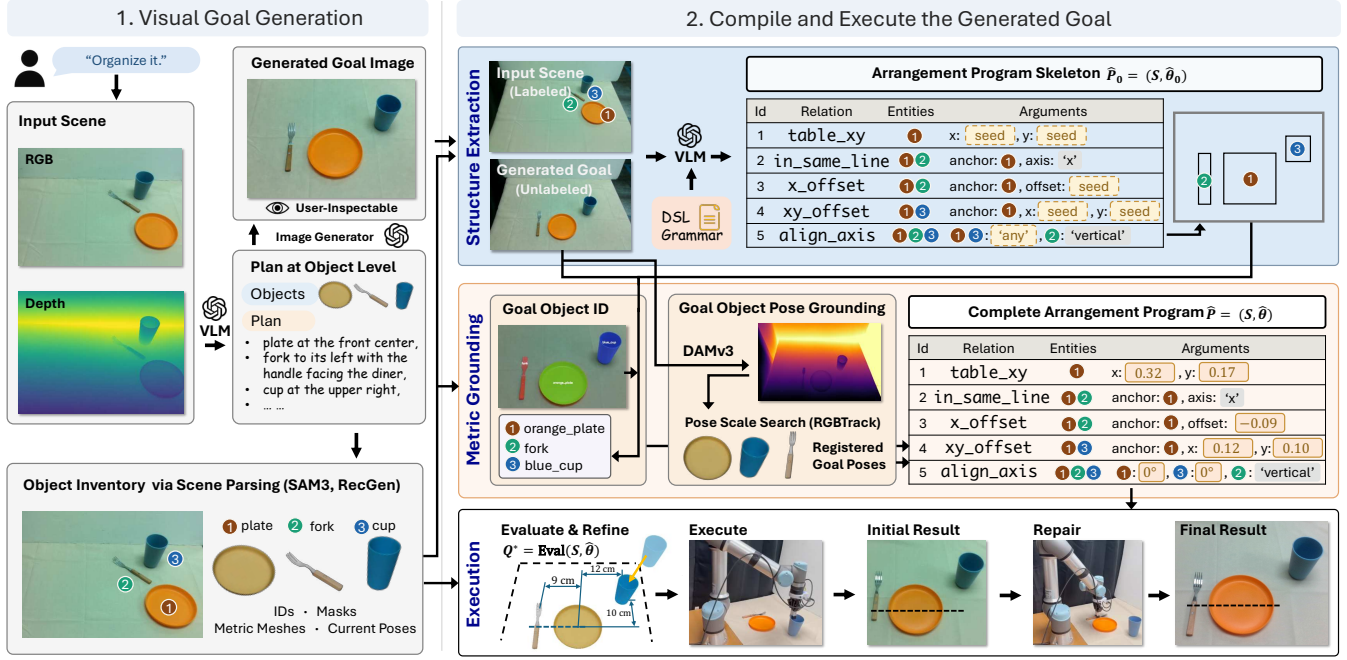


Figure 2: Overview of ORGANIZEIt. Given a cluttered RGB-D observation and a high-level instruction, a VLM produces an object-wise organization plan, which an image generator renders as a tidy preview. The real scene provides an authoritative inventory of object identities and geometry. A dual-channel compiler extracts relational structure with a VLM and grounds metric parameters through perception; the completed program is refined into collision-free targets that drive dependency-aware execution and post-execution repair.

suits open-ended organization, needing neither a reference nor an absolute scale, though such judges can be order-sensitive (Chen et al. 2025c; Zheng et al. 2023; Wang et al. 2024). We therefore collect bidirectional pairwise VLM judgments on post-execution images for tidiness and user-intent alignment, validated against human raters.

3 Method

3.1 Overview and Problem Setup

We study messy scene organization from a single calibrated RGB-D observation and a high-level instruction. The scene contains objects $\mathcal{O} = \{o_1, \dots, o_N\}$, and the system must ground a target configuration $Q^* = (q_1^*, \dots, q_N^*)$, then find a feasible pick-and-place sequence realizing it. An ordinary placement leaves an object upright on a support surface, so we parameterize it by a SE(2) pose $q_i = (x_i, y_i, \psi_i)$ with ψ_i the yaw in the table frame; height is fixed by the table or a declared support relation together with the reconstructed geometry. Perception estimates full 6-DoF poses, so a depicted non-planar reorientation retains its complete orientation.

Figure 2 summarizes the system. Stage 1 converts the observation and instruction into an inspectable tidy preview, the *goal image* (§3.2). Stage 2 is a dual-channel compiler: a VLM compiles the depicted relational structure into an arrangement program with metric values left open (§3.3), and perception grounds them from the goal image and real object geometry (§3.4). Registered poses are therefore measurements, not independent targets: the program decides which

enter the arrangement and derives the coupled poses. The result is refined for feasibility, ordered by a dependency-aware planner, and executed (§3.5).

The real scene is parsed once: we segment the observation (Carion et al. 2025), register every detected object as a persistent entity, and reconstruct a complete mesh for each (Zadaianchuk et al. 2026), which the calibrated depth places at metric scale and current pose. This parse supplies the inventory and geometry used downstream. All system-internal VLM calls use GPT-5.5, and all goal images are generated by GPT-Image-2.

3.2 Scene-Aware Goal Generation

The instruction ranges from a generic “organize it” to a task-specific description supplying additional context, e.g., which side of the table the user will sit at. It never specifies object-level poses or placement constraints.

A single VLM call infers the scene context and drafts an object-wise organization plan assigning each object a functional role and a coarse placement in natural language, for example “place the teapot at the upper right, with its spout toward the mug.” Passing the observation and this plan to an image generator yields a tidied version of the same scene. The plan separates semantic organization from rendering: the VLM decides what the arrangement accomplishes, the generator commits it to a concrete layout.

In our evaluation the first generated image is used directly; interactively, the user may request edits until satisfied. The

accepted image is the sole representation of the tidying intent consumed by Stage 2.

3.3 Arrangement Structure Extraction

The language. We introduce a domain-specific language (DSL) for the depicted structure. An arrangement program is $P = (S, \theta)$: the skeleton S is an ordered list of relation declarations, each naming a relation type, its entities, an optional anchor, and discrete arguments; θ holds the continuous fields they expose. Extraction fixes S and leaves θ open.

Declaration order is part of the semantics: any anchor a relation reads must have been written earlier, inducing a define-before-use dependency. Each relation declares its write-set over $(o_i.x, o_i.y, o_i.\psi)$: `in_holder` writes all three, whereas `on_top_of` writes only the support relation that fixes height, leaving a stacked object’s planar variables to other declarations. Every scalar pose variable is written exactly once, so a pose may be assembled from several relations but never overwritten, making inventory coverage, define-before-use, and write conflicts statically checkable. A deterministic evaluator $\text{Eval}(S, \theta)$ unrolls the declarations in order into a complete target configuration. Table 1 summarizes the minimal relation vocabulary.

Program synthesis. A VLM synthesizes the program in one pass from two images: the real scene with numbered segmentation masks, which binds program variables to entities, and the unlabeled goal image. The prompt supplies the grammar and an inventory contract: a well-formed program over exactly the registered inventory. Hallucinated objects are ignored; objects missing from the goal image are completed from the depicted structure or commonsense, so a three-pen row may extend to all six registered pens. Because many programs evaluate to similar poses, we bias synthesis toward higher-order relations rather than equivalent chains of pairwise offsets: a shared-line and even-spacing declaration keeps a row a row when its values change, whereas an offset chain does not. The VLM may also emit provisional numeric seeds, which serve only the layout render of §3.4 and are later overwritten by grounding.

3.4 Metric Grounding: The Program Parameters

Reference-mask assignment. Segmenting the goal image yields candidate masks, possibly duplicated, merged, distorted, or hallucinated. To guide assignment we produce a top-down layout render of the program, with every mesh at its seeded pose and scale, so the render carries the authoritative inventory while roughly following the depicted layout. A VLM then assigns one reference mask per entity using layout and category cues; entities may share a mask, unused masks are discarded, and the program rather than the registration separates entities sharing one. The mask defines the pose-detection region; the VLM supplies no metric value.

Parameter filling. The goal image has no measured depth. We estimate monocular depth (Lin et al. 2025) and rectify it against the world table plane, giving a pseudo-RGB-D observation in the table frame. Each mesh is registered to it within its reference mask; because that depth is only approximately

Relation	Writes	Arguments
<code>table_x/y</code>	$o.x$ and/or $o.y$	position
<code>x/y_offset_from</code>	$o.x$ and/or $o.y$	offset
<code>in_same_line</code>	$o.x$ or $o.y$	axis
<code>evenly_spaced</code>	$o.x$ or $o.y$	axis, order, spacing, mode
<code>in_holder</code>	$o.x, o.y, o.\psi$	—
<code>align_axis</code>	$o.\psi$	hori., vert., custom, any
<code>on_top_of</code>	support relation	—

Table 1: The relation vocabulary. o denotes a constrained object. The third column lists each relation’s arguments; discrete arguments belong to S , continuous ones to θ . Declared write-sets make single assignment statically checkable. `evenly_spaced` abbreviates `evenly_spaced_from_anchor`. Full semantics are given in Appendix B.

metric and generation can distort apparent scale, registration searches jointly over scale and pose (Guo and Yu 2025; Wen et al. 2024), refined by render-and-compare.

A registration is a measurement, not a target: traversing the ordered program, perception fills only its open fields, and each declaration either measures a value from the relevant registered pose or derives what it writes from an earlier anchor. If A is anchored on the table and B shares A ’s y , the shared-line declaration sets $B.y = A.y$ and only $B.x$ comes from B ’s registration; offsets, spacings, and custom (off-axis) orientations follow analogously (Appendix C). Writing $\hat{\theta}$ for the filled parameters, $\text{Eval}(S, \hat{\theta})$ then produces Q^* : the evaluator never corrects poses independently, so every declared relation holds exactly.

3.5 Refinement and Execution

The evaluated configuration may still collide or extend beyond the table, since the generated viewpoint and geometry are imperfect. We resolve these by refining only θ while keeping S fixed, so the declared relations are preserved. Collision checks use two representations: reconstructed meshes, complete but approximate, and observed partial point clouds transformed to the candidate targets, sensor-grounded but incomplete. The final layout must be collision-free under both and within the table boundary, yielding one rigid transformation per entity (Appendix D).

Following prior work (Zhang, Huang, and Yu 2026), we order the moves with a dependency graph in which an edge (i, j) means i cannot move before j is handled: j rests on or overhangs i , occupies its goal region, or is its goal support. Depth-first search with backtracking finds an executable sequence of goal and buffer moves (Appendix E): objects with no outgoing edge move directly, and a cycle is broken by relocating one movable blocker to a buffer pose.

Planar targets are executed by top-down transport, while non-planar reorientation and `in_holder` insertion are routed to dedicated manipulation skills (Appendix F). The program remains available afterwards: we re-perceive the scene, evaluate each declared relation, and re-place affected

objects when a residual exceeds its threshold, so the program that derived the targets also specifies verification and repair.

4 ORGANIZEIT Dataset

To evaluate tabletop organization under natural user instructions, we construct the ORGANIZEIT dataset (Figure 1): 1,000 human-verified messy–tidy scene pairs grounded in 74 daily activities across 10 scenario types (e.g., office desk, dining table, kitchen countertop), with 4 to 13 objects per scene (7.45 on average). Each sample provides a physically stable messy scene, a human-authored tidy goal configuration, and a task prompt that conveys the task context and the user’s seating position but never specifies where objects go. With the protocol of §5.1, it is the benchmark for most of our experiments; its per-object annotations also support training.

We construct it in four stages: VLM-based proposal of activities and templates, human curation with DSL annotation, scene instantiation, and verification.

The library is seeded with household and tabletop objects from RoboTwin (Mu et al. 2025), Google Scanned Objects (Downs et al. 2022), and SG-Bot (Zhai et al. 2024), and extended with assets from Objaverse (Deitke et al. 2023), Lightwheel (Lightwheel 2025), NutritionVerse-3D (Tai et al. 2023), and Sketchfab when an activity needs new categories. Its 766 assets are organized manually into 100 everyday object categories whose members share function and placement semantics and are interchangeable during instantiation.

The dataset is organized hierarchically. A scenario is a common household surface such as a nightstand; it contains daily activities, concrete situations with a short narrative such as taking medicine before sleep; and each activity is realized by templates that pair a situational prompt with an object list fixing the categories and their counts. From 15 candidate scenarios, a VLM proposes 8 activities per scenario and 8 templates per activity, yielding 960 candidate templates.

We curate these manually, retaining the 100 templates (covering 74 activities) that plausibly occur in daily life and merging overlapping or rare scenarios into 10 scenario types. Each retained template is annotated with the DSL of §3.3 to form a tidy template; these annotations serve only instantiation and evaluation, and no layout information is exposed to any evaluated method. We also verify manually that no prompt specifies placement, while every prompt still provides enough context to infer a reasonable arrangement.

Each tidy template is instantiated into 10 tidy scenes, 1,000 in total, differing in object appearance, as assets are drawn interchangeably within a category; in table texture; and in placement, by perturbing the DSL parameters within admissible ranges. From each we derive a messy scene with the same objects: each is dropped from above onto a random unoccupied table region and settles under simulated physics, producing physically plausible clutter.

Every scene must pass verification. Automatic verification requires each object stable before and after simulator reload and at least 60% visible to the camera. A manual pass discards implausible poses and confirms each object is identifiable from its visible portion. Failed scenes are resampled.

5 Experiments

Our experiments address five questions. Does ORGANIZEIT beat a strong rearrangement baseline under that baseline’s own conditions (§5.2)? Is the dual-channel reading of the goal image necessary (§5.3)? Does explicit planning improve goal generation over directly prompting the image-editing model (§5.4)? Can VLM judging be trusted here (§5.5)? And does the system transfer to a real robot (§5.6)?

5.1 Evaluation Protocol

Tidiness admits no unique ground truth: a cluttered table can be organized into many equally tidy configurations. Prior work often designates one reference arrangement and reports per-object displacement error against it (Zhai et al. 2024; Liu et al. 2022b), which penalizes every valid alternative and measures agreement with one annotator rather than quality. We therefore evaluate open-endedly, comparing two systems on one scene without a privileged reference.

Each comparison presents one image of each system’s final post-execution scene, rendered in the same simulator from the same camera. The judge picks the better result or declares a tie on two criteria: *tidiness*, which arrangement is tidier, and *user intent*, which better serves the task context in the prompt. Every pair is judged twice, the second round swapping the presentation order. A win or loss is counted when both rounds agree, or when one returns that verdict and the other a tie; verdicts that flip are reported as inconsistent. This removes positional bias and exposes judge uncertainty rather than hiding it; inconsistent verdicts concentrate on pairs of similar quality, and we report them in full.

The judges are VLMs: §5.2 uses three VLM judges—GPT-5.5, Claude Opus 4.8, and Gemini 3.1 Pro—whereas §5.3 uses GPT-5.5 alone. Although GPT-5.5 is also used within our system, it judges only the post-execution observations, rendered identically for all methods. The other two judges agree with it on the direction of every comparison, and §5.5 further shows that the VLM-judging protocol closely agrees with human raters.

5.2 Head-to-Head Comparison with SG-Bot

We first compare against the available strong baseline, SG-Bot (Zhai et al. 2024), on its own benchmark of 90 scenes (baseline selection: Appendix A.1). SG-Bot takes a scene relation graph as its native input, and our system receives the same graph verbatim as its instruction; the graph specifies placement-level structure ours normally operates without, so supplying it lets SG-Bot compete with its full native input. Both configurations are executed by SG-Bot’s own teleport procedure and rendered by the same pipeline, so preference can only come from the arrangement itself.

Our system wins under every judge and on both criteria (Table 2, Figure 3): tidiness win rates run 47.8% to 52.2% against losses of 28.9% to 34.4%, and on user intent, the largest gap is GPT-5.5’s 57.8% versus 13.3%. All three judges agree on the direction of every comparison, making the conclusion robust to the choice of judge.

Judge	Tidiness				User Intent			
	Win	Loss	Tie	Inc.	Win	Loss	Tie	Inc.
GPT-5.5	52.2	31.1	0.0	16.7	57.8	13.3	0.0	28.9
Claude Opus 4.8	47.8	28.9	5.6	17.8	46.7	27.8	12.2	13.3
Gemini 3.1 Pro	52.2	34.4	10.0	3.3	45.6	34.4	1.1	18.9

Table 2: Head-to-head comparison with SG-Bot on its own benchmark (90 scenes). Win/Loss are from our system’s perspective. Inc. denotes inconsistent verdicts, which flip under A/B order swap. All values are percentages.

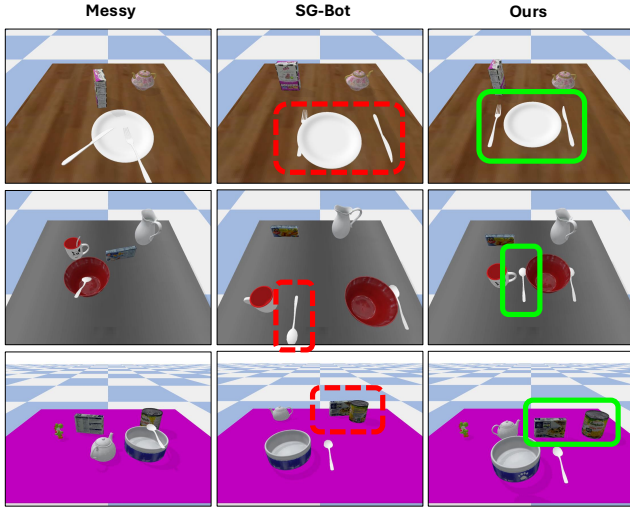


Figure 3: Qualitative comparison with SG-Bot. Red dashed boxes mark imprecise placements by SG-Bot, and green solid boxes the corresponding regions in our results.

5.3 Main Results on ORGANIZEIt

We evaluate on all 1,000 scenes of ORGANIZEIt. Each system receives the messy observation and the task prompt; an idealized pick-and-place executor realizes its target configuration step by step in simulation. We compare ours with three ablations and an oracle. *Perception-only* removes VLM structure extraction, registering each object against the goal image independently with no program constraining the targets, though it retains the reference-mask association of §3.4 for fairness. *VLM w/ goal* removes the perception channel. *VLM w/o goal* removes the goal image too, writing the program from the messy observation and the organization plan. Both VLM variants share an enhanced DSL prompt supplying true object dimensions and a numeric delta-angle field, since they must otherwise state every metric value themselves. *Reference goal*, the human-authored tidy configuration from the ORGANIZEIt dataset, serves as an oracle. One goal image is generated per scene and shared across variants.

The three ablations correspond to the dual-channel claim of §3.1, and ours beats each on both criteria (Table 3; examples in Appendix A.2). *Perception-only* independently grounds each object from the same goal image and is beaten 56.6% to 17.7% on tidiness. Its individually small but un-

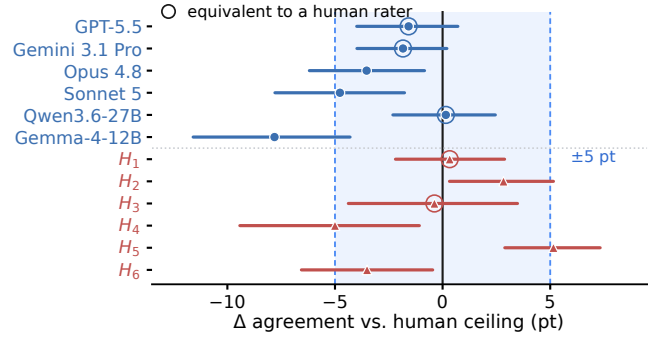


Figure 4: Each judge’s agreement with our 6 human raters minus the raters’ agreement among themselves (90% bootstrap CIs, 100 scenes). A judge is equivalent when its interval lies inside the ± 5 pt margin (circled): GPT-5.5, Gemini 3.1 Pro, and Qwen3.6-27B qualify; four of six humans do not.

Comparison	Tidiness				User Intent			
	Win	Loss	Tie	Inc.	Win	Loss	Tie	Inc.
vs. Perception-only	56.6	17.7	6.0	19.7	43.5	22.1	11.8	22.6
vs. VLM w/ goal	52.9	20.0	1.6	25.5	47.0	21.4	1.8	29.8
vs. VLM w/o goal	59.8	20.1	0.5	19.6	47.8	25.1	0.4	26.7
vs. Reference goal	23.6	60.4	0.2	15.8	32.5	46.6	0.5	20.4
VLM w/ goal vs.								
VLM w/o goal	42.7	33.0	1.6	22.7	30.5	37.6	4.0	27.9

Table 3: Pairwise VLM-judge results on 1,000 scenes. Rows 1–4 compare ours with each ablated baseline; the last compares VLMs with and without the goal image. Win/Loss refer to the first-named system; Inc. denotes inconsistency. All values are percentages.

coupled errors degrade relational tidiness as predicted; its tie rates are the table’s highest (6.0%, 11.8%) as independent grounding often lands near tidy. *VLM w/ goal* instantiates the same program class by estimation but is beaten 52.9% to 20.0%, showing metric values must come from perception. *VLM w/o goal* never imagines a goal and is beaten by the largest margin, 59.8% to 20.1%. User intent shows the same pattern: wins at 43.5%–47.8% versus losses at 21.4%–25.1% across all three baselines.

The last row examines the goal image in isolation. *VLM w/ goal* beats *VLM w/o goal* on tidiness (42.7% versus 33.0%) but loses on user intent (30.5% versus 37.6%): naive conditioning recovers little of the image’s value and can even hurt. Read through our two channels, the same image beats *VLM w/o goal* by 39.7 points on tidiness, so extraction, not the image alone, is what turns that information into results.

The oracle row calibrates the remaining headroom. Against the human-authored reference we win 23.6% and lose 60.4% on tidiness, so human layouts remain clearly better, while on user intent the gap narrows to 32.5% versus 46.6%. The system trails human taste in geometric polish but is competitive in reading what a scene is for.



Figure 5: Direct generation (w/o *Plan*) vs. our plan-then-edit pipeline (w/ *Plan*): each row shows an initial cluttered scene (left) and the goal image each variant generates.

5.4 Plan-before-Edit Ablation

In Stage 1 a VLM produces an object-wise organization plan that an image-editing model (GPT-Image-2) renders as the goal image. To test whether this step is necessary, we prompt the same model directly to tidy the scene, controlling the prompts so both variants receive the same information. Under the protocol of §5.1, planning wins decisively (51% vs. 22% on tidiness, 85% vs. 5% on user intent; Appendix A.3). Figure 5 suggests why: without a plan the editing model must resolve intent, spatial reasoning, and rendering in one pass, and it sacrifices the first two for the last, dropping or deforming objects (collapsed laptop, row 1) and drifting toward local adjustments rather than intent-consistent regrouping (fruits left off the plate, row 3). Decoupling *what to do* from *how it looks* lets each model work within its competence.

5.5 Human Validation of the Protocol

We validate the VLM verdicts against human judgment on 100 representative main-experiment scenes (of 1,000). Six humans and six VLM judges rate the same pairs double-blind under the bidirectional tidiness protocol.

On comparisons where both raters commit to a winner, agreement is 83.0% across the 15 human-human pairs and 79.9% across the 36 human-VLM pairs ($\kappa=0.64/0.57$). We treat the human-human rate as a ceiling and measure each judge’s shortfall, with confidence intervals from a scene-level bootstrap. Our least-consistent human falls 5 pt short, so we test equivalence at a ± 5 pt margin. GPT-5.5, the judge behind our main results, passes ($\Delta = -1.6$, 90% CI $[-4.0, +0.7]$; TOST, $\alpha=.05$), as do Gemini 3.1 Pro and Qwen3.6-27B. Four of six humans fail (Figure 4). All twelve judges rank ours below the oracle and above every baseline, so Tables 2 and 3 track human judgment in direction (Appendix A.4).

5.6 Real-Robot Demonstrations

We deploy the full system on a UR5e arm with a wrist-mounted RGB-D camera, synthesizing grasps with AnyGrasp (Fang et al. 2023). Trials span four everyday domains: a dining table, a kitchen countertop, a study desk, and a

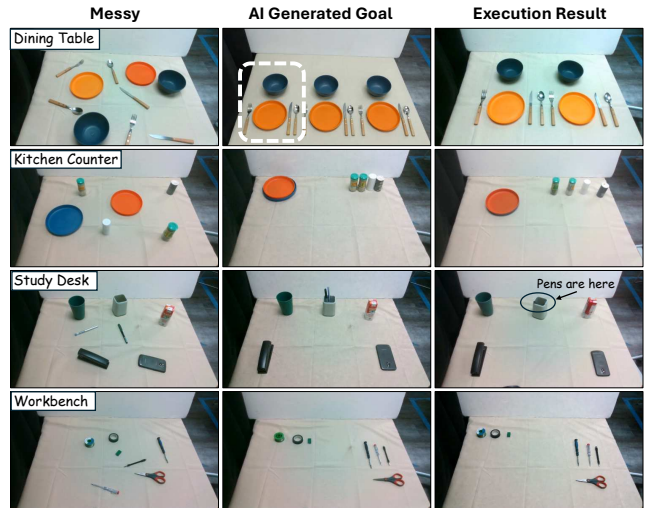


Figure 6: Real-robot demonstrations across four tabletop domains: initial scene, generated goal, and execution result. White dashed boxes mark duplicated items introduced by the image generator.

workbench. Each receives only the cluttered observation and the instruction “Organize it.”, from which the system generates the goal image, grounds the program, and executes the resulting transformations (Figure 6). The robot realizes the depicted groupings, alignments, spacings, and orientations. Some goals contain duplicated objects hallucinated by the generator, marked by white dashed boxes; the system discards them and succeeds regardless. We further re-shuffle each domain into a fresh clutter five times, and on all 20 scenes the pipeline returned a feasible tidy configuration, a 100% success rate. For additional trials and full rollouts, see the supplementary materials.

6 Conclusion

We presented ORGANIZEIt, a framework for zero-shot, CAD-free tabletop organization from a single RGB-D observation and an underspecified instruction. It renders the instruction as an inspectable tidy preview, then compiles that image into a relation-preserving, executable target arrangement. At its core is a persistent arrangement program: a VLM extracts the goal image’s relational structure, while perception grounds its metric parameters. The completed program determines object targets and drives relation-preserving refinement, dependency-aware execution, and post-execution repair. Across the 1,000-scene ORGANIZEIt dataset, the full system is consistently preferred over variants removing either structure extraction or metric grounding. Human evaluation validates the bidirectional VLM-judging protocol, and real-robot trials confirm end-to-end execution. Generated images can thus provide practical robot goals when semantic structure and metric geometry are grounded jointly. The ORGANIZEIt dataset and code will be released upon publication.

References

- Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. 2022. Do As I Can, Not As I Say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*.
- Ajay, A.; Han, S.; Du, Y.; Li, S.; Gupta, A.; Jaakkola, T.; Tenenbaum, J.; Kaelbling, L.; Srivastava, A.; and Agrawal, P. 2023. Compositional foundation models for hierarchical planning. *Advances in Neural Information Processing Systems*, 36: 22304–22325.
- Black, K.; Nakamoto, M.; Atreya, P.; Walke, H.; Finn, C.; Kumar, A.; and Levine, S. 2024. Zero-shot robotic manipulation with pre-trained image-editing diffusion models. In *International Conference on Learning Representations*.
- Carion, N.; Gustafson, L.; Hu, Y.-T.; Debnath, S.; Hu, R.; Suris, D.; Ryali, C.; Alwala, K. V.; Khedr, H.; Huang, A.; et al. 2025. SAM 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719*.
- Chang, H.; Gao, K.; Boyalakuntla, K.; Lee, A.; Huang, B.; Yu, J.; and Boularias, A. 2024. LGMCTS: Language-guided monte-carlo tree search for executable semantic object rearrangement. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 13607–13612. IEEE.
- Chen, H.; Guo, J.; Wang, B.; Zhang, T.; Huang, X.; Zheng, B.; Hou, Y.; Tie, C.; Deng, J.; and Shao, L. 2025a. Goal-VLA: Image-Generative VLMs as Object-Centric World Models Empowering Zero-shot Robot Manipulation. *arXiv preprint arXiv:2506.23919*.
- Chen, W.; Xiao, C.; Gao, G.; Sun, F.; Zhang, C.; and Zhang, J. 2025b. DreamArrangement: Learning language-conditioned robotic rearrangement of objects via denoising diffusion and VLM planner. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.
- Chen, Z.; Wen, Z.; Du, Y.; Zhou, Y.; Cui, C.; Han, S.; Weng, J.; Wang, C.; Tong, Z.; Huang, L.; et al. 2025c. MJ-Bench: Is your multimodal reward model really a good judge for text-to-image generation? *Advances in Neural Information Processing Systems*, 38.
- Deitke, M.; Schwenk, D.; Salvador, J.; Weihs, L.; Michel, O.; VanderBilt, E.; Schmidt, L.; Ehsani, K.; Kembhavi, A.; and Farhadi, A. 2023. Objaverse: A universe of annotated 3D objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13142–13153.
- Ding, Y.; Zhang, X.; Paxton, C.; and Zhang, S. 2023. Task and motion planning with large language models for object rearrangement. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Downs, L.; Francis, A.; Koenig, N.; Kinman, B.; Hickman, R.; Reymann, K.; McHugh, T. B.; and Vanhoucke, V. 2022. Google scanned objects: A high-quality dataset of 3D scanned household items. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2553–2560. IEEE.
- Du, Y.; Yang, S.; Dai, B.; Dai, H.; Nachum, O.; Tenenbaum, J.; Schuurmans, D.; and Abbeel, P. 2023. Learning universal policies via text-guided video generation. *Advances in Neural Information Processing Systems*, 36: 9156–9172.
- Du, Y.; Yang, S.; Florence, P.; Xia, F.; Wahid, A.; Sermanet, P.; Yu, T.; Abbeel, P.; Tenenbaum, J. B.; Kaelbling, L.; et al. 2024. Video language planning. In *International Conference on Learning Representations*.
- Fang, H.-S.; Wang, C.; Fang, H.; Gou, M.; Liu, J.; Yan, H.; Liu, W.; Xie, Y.; and Lu, C. 2023. AnyGrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*, 39(5): 3929–3945.
- Guo, T.; and Yu, J. 2025. RGBTrack: Fast, Robust Depth-Free 6D Pose Estimation and Tracking. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 18872–18879. IEEE.
- Hu, Y.; Goldfeder, J.; Zhang, Z.; Zhu, X.; Liu, R.; Wyder, P. M.; Lin, J.; and Lipson, H. 2025. Knolling Bot: Teaching Robots the Human Notion of Tidiness. In *Advances in Neural Information Processing Systems (Creative AI Track)*.
- Huang, W.; Wang, C.; Zhang, R.; Li, Y.; Wu, J.; and Fei-Fei, L. 2023. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *Conference on Robot Learning (CoRL)*.
- Jin, S.; Wang, R.; Chen, K.; and Pokorny, F. T. 2025. PACA: Perspective-aware cross-attention representation for zero-shot scene rearrangement. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 6559–6569. IEEE.
- Kant, Y.; Ramachandran, A.; Yenamandra, S.; Gilitschenski, I.; Batra, D.; Szot, A.; and Agrawal, H. 2022. Housekeep: Tidying virtual households using commonsense reasoning. In *European Conference on Computer Vision (ECCV)*.
- Kapelyukh, I.; Ren, Y.; Alzugaray, I.; and Johns, E. 2024. Dream2Real: Zero-shot 3D object rearrangement with vision-language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 4796–4803. IEEE.
- Kapelyukh, I.; Vosylius, V.; and Johns, E. 2023. DALL-E-Bot: Introducing web-scale diffusion models to robotics. *IEEE Robotics and Automation Letters*, 8(7): 3956–3963.
- Kee, H.; Oh, W.; Kang, M.; Ahn, H.; and Oh, S. 2025. Tidiness Score-Guided Monte Carlo Tree Search for Visual Tabletop Rearrangement. *arXiv preprint arXiv:2502.17235*.
- Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Lightwheel. 2025. Lightwheel Sim-Ready Assets: High-Quality USD Assets for NVIDIA Isaac Sim. GitHub repository, <https://github.com/LightwheelAI/Lightwheel-simready-asset>. 259 robotics simulation assets under CC BY-NC 4.0.
- Lin, H.; Chen, S.; Liew, J.; Chen, D. Y.; Li, Z.; Shi, G.; Feng, J.; and Kang, B. 2025. Depth Anything 3: Recovering the Visual Space from Any Views. *arXiv preprint arXiv:2511.10647*.

- Liu, L.; Wang, W.; Han, Y.; Xie, Z.; Yi, P.; Li, J.; and Lian, W. 2026. FoAM: Foresight-augmented multi-task imitation policy for robotic manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, 18460–18468.
- Liu, W.; Hermans, T.; Chernova, S.; and Paxton, C. 2022a. StructDiffusion: Object-centric diffusion for semantic rearrangement of novel objects. In *Workshop on Language and Robotics at CoRL 2022*.
- Liu, W.; Paxton, C.; Hermans, T.; and Fox, D. 2022b. StructFormer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 6322–6329. IEEE.
- Mendez, A.; Prados, A.; Menendez, E.; and Barber, R. 2024. Everyday objects rearrangement in a human-like manner via robotic imagination and learning from demonstration. *IEEE Access*, 12: 92098–92119.
- Mu, Y.; Chen, T.; Chen, Z.; Peng, S.; Lan, Z.; Gao, Z.; Liang, Z.; Yu, Q.; Zou, Y.; Xu, M.; et al. 2025. RoboTwin: Dual-arm robot benchmark with generative digital twins. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 27649–27660.
- OpenAI. 2026. Introducing ChatGPT Images 2.0. <https://openai.com/index/introducing-chatgpt-images-2-0/>. Accessed: 2026-06-21.
- Sarch, G.; Fang, Z.; Harley, A. W.; Schydlo, P.; Tarr, M. J.; Gupta, S.; and Fragkiadaki, K. 2022. TIDEE: Tidying up novel rooms using visuo-semantic commonsense priors. In *European Conference on Computer Vision (ECCV)*.
- Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. ProgPrompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Tai, C.-e. A.; Keller, M.; Kerrigan, M.; Chen, Y.; Nair, S.; Xi, P.; and Wong, A. 2023. NutritionVerse-3D: A 3D food model dataset for nutritional intake estimation. *arXiv preprint arXiv:2304.05619*.
- Wang, J.; Hong, Z.; Bao, C.; Zhu, Y.; Sun, J.; and Zhang, G. 2026. LAMP: Lift Image-Editing as General 3D Priors for Open-world Manipulation. *arXiv preprint arXiv:2604.08475*.
- Wang, P.; Li, L.; Chen, L.; Cai, Z.; Zhu, D.; Lin, B.; Cao, Y.; Kong, L.; Liu, Q.; Liu, T.; and Sui, Z. 2024. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.
- Wei, Q. A.; Ding, S.; Park, J. J.; Sajjani, R.; Poulenard, A.; Sridhar, S.; and Guibas, L. 2023. LEGO-Net: Learning regular rearrangements of objects in rooms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 19037–19047.
- Wen, B.; Yang, W.; Kautz, J.; and Birchfield, S. 2024. FoundationPose: Unified 6D pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 17868–17879.
- Wu, J.; Antonova, R.; Kan, A.; Lepert, M.; Zeng, A.; Song, S.; Bohg, J.; Rusinkiewicz, S.; and Funkhouser, T. 2023. TidyBot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8): 1087–1102.
- Xiang, F.; Qin, Y.; Mo, K.; Xia, Y.; Zhu, H.; Liu, F.; Liu, M.; Jiang, H.; Yuan, Y.; Wang, H.; et al. 2020. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11097–11107.
- Xu, Y.; Mao, J.; Li, L.; Du, Y.; Lozano-Pérez, T.; Kaelbling, L. P.; and Hsu, D. 2025. “Set it up”: Functional object arrangement with compositional generative models. *The International Journal of Robotics Research*, 02783649251378198.
- Zadaianchuk, A.; Barcellona, L.; Schuenemann, L.; Gumbusch, C.; Wang, Z.; Irshad, M. Z.; Despinoy, F.; Aljundi, R.; Gavves, S.; and Zakharov, S. 2026. Reconstruction by Generation: 3D Multi-Object Scene Reconstruction from Sparse Observations. *arXiv preprint arXiv:2604.27106*.
- Zeng, Y.; Wu, M.; Yang, L.; Zhang, J.; Ding, H.; Cheng, H.; and Dong, H. 2024. LVDiffusor: Distilling functional rearrangement priors from large models into diffusor. *IEEE Robotics and Automation Letters*, 9(10): 8258–8265.
- Zhai, G.; Cai, X.; Huang, D.; Di, Y.; Manhardt, F.; Tombari, F.; Navab, N.; and Busam, B. 2024. SG-Bot: Object rearrangement via coarse-to-fine robotic imagination on scene graphs. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 4303–4310. IEEE.
- Zhang, D.; Huang, J.; and Yu, J. 2026. High-Performance Dual-Arm Task and Motion Planning for Tabletop Rearrangement. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Zhang, X.; Lu, Y.; Wang, W.; Yan, A.; Yan, J.; Qin, L.; Wang, H.; Yan, X.; Wang, W. Y.; and Petzold, L. R. 2023. GPT-4V(ision) as a generalist evaluator for vision-language tasks. *arXiv preprint arXiv:2311.01361*.
- Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems*, 36.

Supplementary Materials

A Additional Experimental Results

A.1 Baseline Selection

The comparison venue. A prior system’s ability to technically process a scene is not, by itself, sufficient for a meaningful comparison. Open-ended organization admits no privileged reference configuration (§5.1), so our protocol compares two systems by executing both on the same scene and judging the rendered post-execution results pairwise. A quantitative comparison therefore needs a common venue: each system must receive its required input without retraining or a method-specific adapter, and both outputs must be executed, rendered from the same camera, and judged by the same pipeline. Numbers produced under different simulators, success predicates, oracle inputs, and sampling budgets are not comparable, so we do not juxtapose our results with figures published elsewhere. Two venues are possible: our benchmark, or a prior one.

On our benchmark. No released prior system consumes our input contract — a single RGB-D observation with an underspecified instruction — and returns a complete arrangement over an open object vocabulary. Structure-conditioned methods (Liu et al. 2022b,a; Chang et al. 2024; Xu et al. 2025) require the target structure to be named, as a structure family, a program sketch, or explicit geometric relations; supplying that information would mean authoring by hand the intent our setting asks the system to infer, and withholding it takes these methods out of their task. Learned layout priors (Wei et al. 2023; Zeng et al. 2024; Chen et al. 2025b) are trained for restricted structure, asset, or activity distributions and provide no model transferable to our open inventory and prompts. Receptacle-assignment and geometric-tidiness methods (Wu et al. 2023; Kee et al. 2025) optimize object destinations or unconditioned regularity rather than a prompt-conditioned configuration of every object, so they cannot be judged on user-intent compliance under the same input contract. The one paradigm whose input contract matches ours is generated-goal matching (Kapelyukh, Vosylius, and Johns 2023; Jin et al. 2025), which registers each object independently against a generated image. Neither system is publicly released, and reproducing one would entangle the paradigm’s defining assumption with implementation choices we would be making on its behalf; for the earlier system these are also dated ones — a closed-set detector and the image generator of its time. We therefore test that assumption in controlled form: the *Perception-only* baseline of §5.3 instantiates independent per-object grounding while sharing our goal generation, segmentation, registration, executor, and judges, so the comparison isolates exactly the design choice the paradigm asserts. It is a strengthened representative of the family — it inherits the plan-then-edit goal generation that §5.4 shows outperforms direct generation — and is evaluated on the same footing as every other variant.

No retraining on ORGANIZEIT. We also do not retrain task-specific learned baselines, including the layout priors above, on our benchmark. The 1,000 scenes constitute an evaluation benchmark rather than a train/test corpus: training

on them would expose the evaluated activities and layouts, conflict with our zero-shot setting, and make the outcome depend on an additional data-generation and supervision budget.

On a prior benchmark: why SG-Bot. Adapting to a prior benchmark requires two things. The benchmark must be a self-contained executable artifact — assets, scenes, and simulator in a runnable state — so that both systems’ outputs can be executed and rendered from one camera; and it must define a native goal specification that both systems can consume verbatim. Among prior systems, SG-Bot (Zhai et al. 2024) satisfies both. Its complete 90-scene held-out benchmark ships with a runnable environment and its own teleport executor, and its native scene graph can be handed to our system unchanged as instruction text. We give SG-Bot the full graph and give our system the same graph verbatim, deliberately supplying the placement-level structure our system normally has to infer, so the baseline is neither deprived of its native information nor moved out of distribution. Both predicted configurations are then executed by SG-Bot’s teleport procedure and rendered from the same camera: the object inventory, goal specification, executor, rendering, and judges are shared, and preference can only come from the arrangement itself. Because this benchmark holds 90 scenes, we read the comparison as a directional result and treat the 1,000-scene evaluation of §5.3 as the statistically powered evidence (Appendix A.6).

Why not StructDiffusion. StructDiffusion (Liu et al. 2022a) is the nearest alternative, and the venue failure is not at the input: its released held-out data ship calibrated RGB-D observations with segmentation — the observation type our first stage consumes. What the release does not ship is the environment: the scene definition and simulation harness that would instantiate a configuration are absent, and while object models are separately downloadable, no released component assembles them into a renderable scene. No target configuration, theirs or ours, can therefore be executed and rendered without reconstructing that environment, and the pairwise protocol has nothing to judge. Its native goal specification further supplies an oracle partition of target, distractor, and anchor objects together with a parametric structure specification, information our input contract omits by design; and the physics-verification procedure behind its published success rates is not part of the release, so any evaluator we rebuilt would define a new criterion and produce numbers that are no longer the ones its paper reports.

Scope of the comparison. We do not claim that SG-Bot subsumes prior rearrangement systems, nor that this experiment establishes superiority in their native task-specific settings. The SG-Bot comparison isolates one controlled question: given the same explicit relational goal, which system produces the better complete metric arrangement. The 1,000-scene experiments separately test what the open-ended setting requires — intent inference, generated-goal use, relational compilation, and metric grounding — and our claims and experimental wording are restricted accordingly.

A.2 Qualitative Results on ORGANIZEIt

Figure 7 shows why the aggregate margins of Table 3 take the shape they do. The ablations rarely fail by putting an object somewhere absurd. They fail by getting each object approximately right and the *relation between* objects wrong, which is exactly the failure the dual-channel design is meant to remove.

The third row is the cleanest instance. All four systems place a phone, a fruit plate and a remote across the front of the table, and all four choose essentially the same three positions. Only ours keeps the three centers on one line, which is what the program declares; *perception-only* grounds each object independently against the same goal image and its three small, uncorrelated errors leave the row visibly ragged, while the two VLM variants push one item out of the group altogether. Nothing here is a large error — the displacements are centimeters — but tidiness is a joint property, and it is lost.

The first row makes the same point about orientation and spacing. The instruction is to gather the dishes for collection, and every system gathers the cutlery. Ours returns a bundle whose pieces stay parallel and evenly spaced, because the program declares a shared line and an even-spacing group over them and refinement is not free to break either (Appendix D). The ablations return the same four implements at inconsistent angles and uneven gaps: gathered, but not tidy.

The remaining rows show two other recurring modes. In the fourth row the two VLM variants topple the mirror — an out-of-plane pose that the SE(2)/SE(3) gate of §C.2 exists to catch — while *perception-only* splits the bottles into two clusters instead of one row. In the fifth row the ablations crowd the notebook, pen and calculator into overlapping positions or strand the calculator at the table edge, whereas the declared offsets keep the working area laid out for the stated task. The second row is a case where a single ablation happens to succeed: *VLM w/ goal* produces a reasonable nightstand, which is consistent with it being the strongest of the three in Table 3 and with the 25.5% of tidiness comparisons against it that the judges return as inconsistent.

A.3 Plan-before-Edit Ablation

Perspective	Win	Loss	Tie	Inconsistent
Tidiness	51	22	1	26
User Intent	85	5	0	10

Table 4: Ablation on planning: plan-then-edit vs. direct generation with an information-equivalent prompt, judged by GPT-5.5, on the 100-scene representative set. All values are percentages, so they are also counts.

A.4 Human Validation of the Judging Protocol

§5.5 rests on the claim that our VLM judges track human judgment closely enough for a 1,000-scene VLM-only evaluation to be trustworthy. This subsection gives the study behind that claim.

Raters. Six raters were recruited, and are referred to as H_1 through H_6 throughout. H_1 – H_5 are computer-science students aged 18–25; H_6 has no technical background and is aged 50. None of the six had seen any part of this project before taking part — no scenes, no goal images, no results, and no description of the method — so no rater could recognize our output by its style rather than judging it on the merits. The VLM judges are labeled $V_1, V_2, \dots$ under the same convention, and their roster is given with the results below.

Task and protocol. Each rater completes the same set of comparisons: the 100 representative scenes of §A.6, each compared against five opponents, in both presentation orders. That is $100 \times 5 \times 2 = 1,000$ items per rater. The five opponents are the three ablations of §5.3 (*perception-only*, *VLM w/ goal*, *VLM w/o goal*), the human-authored *reference goal* that serves as the oracle, and the *messy* scene as it was before any tidying.

The study is double-blind. A rater is never told which image comes from which system, and the images are served without any method name attached, so nothing in the interface identifies a side. Which of the two appears on the left is randomized per item, and the full list of 1,000 items is randomly permuted independently for each rater.

Criterion. Humans are asked about tidiness only, which keeps the workload per rater to one question per item. A rater sees the scene’s situational prompt and picks the arrangement they consider tidier, or declares the two roughly equal, judging by their own preference rather than against a rubric we supply. Ties carry no directional information and are treated as abstentions in the agreement computation, exactly as for the VLM judges. Those judges are then run on the identical items under the identical protocol, so the human and VLM verdicts compared in §5.5 differ only in who produced them.

Judge roster. The six VLM judges are V_1 GPT-5.5, V_2 Gemini 3.1 Pro, V_3 Claude Opus 4.8, V_4 Claude Sonnet 5, V_5 Qwen3.6-27B, and V_6 Gemma-4-12B. The first four are the current frontier releases of the three major labs; the last two are open-weight models served locally, included to test whether the protocol’s conclusions survive a much weaker judge. Every judge, human or model, sees the identical 1,000 items.

Reducing an item pair to one verdict. A rater answers each scene–opponent pair twice, once in each presentation order. We fold the two answers into a single verdict per pair: *ours* or *opponent* when the two orders agree, or when one order is a tie and the other is not; *tie* when both orders are ties; and *order-inconsistent* when the two orders name opposite winners. This is the same folding used for the win/loss/tie counts elsewhere in the paper. With 100 scenes and, excluding the *messy* floor, four opponents, each judge therefore contributes 400 folded verdicts.

Preference margins. Figure 8 and Table 5 give each judge’s margin per opponent, the percentage of the 100 scenes it awards to us minus the percentage it awards to

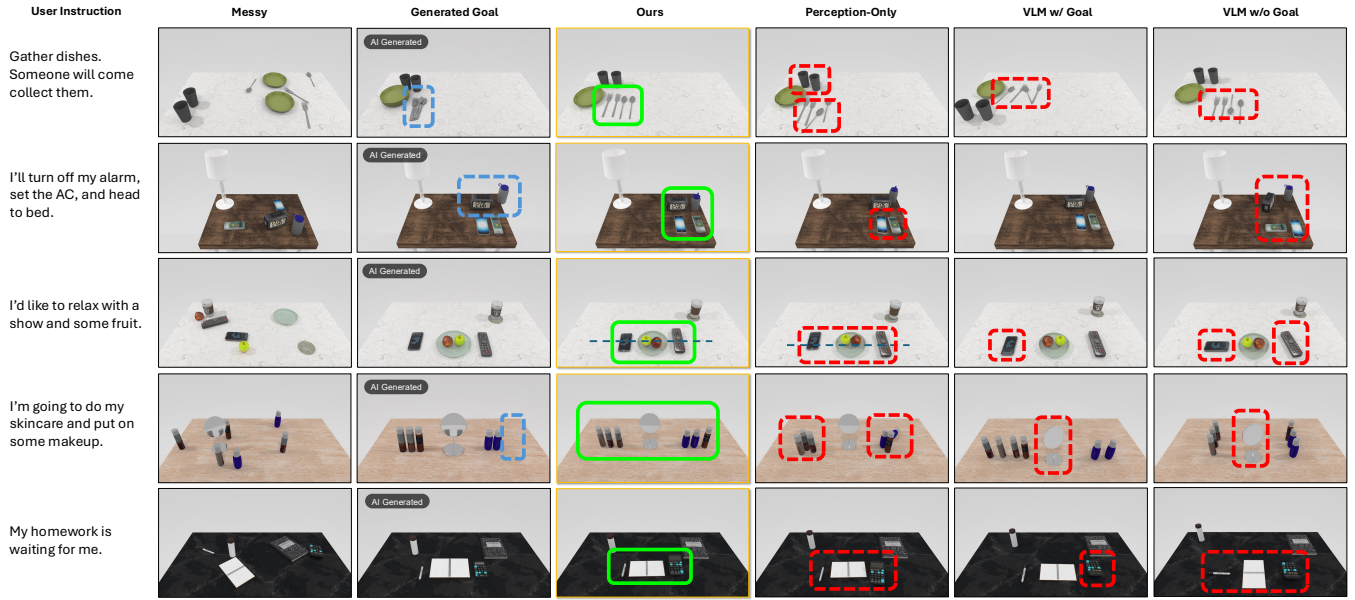


Figure 7: Qualitative comparison on ORGANIZEIt. Each row is one scene: the instruction, the messy observation, the generated goal image, our post-execution result, and the three ablations of §5.3. Blue dashed boxes mark the region of the goal image the row is about; green solid boxes mark the structure our system realizes; red dashed boxes mark where an ablation loses it. The dashed blue line in the third row is the shared line the program declares. All results are rendered from the same camera by the same executor, so only the arrangement differs.

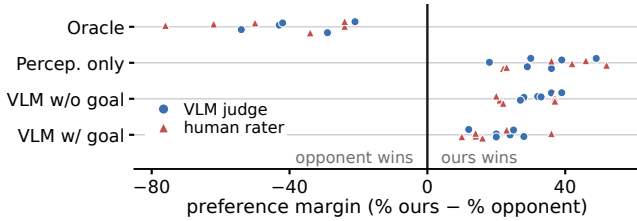


Figure 8: Preference margin of every judge against every opponent. Each row is one opponent and each mark one of the twelve judges; a mark right of zero means that judge preferred our result. Every row lies entirely on one side, so all twelve judges name the same winner in all four comparisons.

the opponent. All twelve judges place our result below the human-authored oracle and above all three ablations: 48 of 48 judge–opponent cells agree in sign. This is the strongest form of the agreement claim, because it involves no threshold and no choice of agreement metric — it only asks who each judge declared the winner.

Agreement metric. Two raters are compared on the 400 folded verdicts. We report agreement only over the pairs on which *both* raters committed to a winner, discarding any pair where either declared a tie or contradicted itself across orders. This is deliberately the conservative choice: the looser convention, in which a tie counts as not objecting to the other rater, cannot distinguish an abstention from a match and inflates every rate by roughly ten points. Restricting to committed pairs also makes the comparison binary, which removes

the class imbalance that depresses κ under the four-way coding: κ rises from 0.24–0.32 to 0.57–0.69. For completeness, under the looser convention the three rates below are 92.8%, 90.9% and 92.7%; no conclusion in this subsection changes between the two conventions.

Pooling all pairs within a group, agreement is 83.0% over the 15 human–human pairs ($\kappa = 0.638$), 79.9% over the 36 human–VLM pairs ($\kappa = 0.570$), and 85.6% over the 15 VLM–VLM pairs ($\kappa = 0.690$).

Equivalence to a human rater. An absolute agreement rate is not interpretable on its own — there is no threshold at which a judge becomes acceptable. We therefore calibrate every judge against the agreement humans reach among themselves. For a judge X we compute Δ_X , the agreement between X and the human raters other than X , minus the agreement those same humans reach with each other. A human rater is scored by exactly the same leave-one-out construction as a model, so the two are directly comparable. Confidence intervals come from resampling the 100 scenes with replacement (4,000 draws), which respects the fact that the 400 verdicts of a judge are clustered within scenes.

A conventional test here could only fail to reject $\Delta_X = 0$, which is weak evidence: with 100 scenes, a null result is as consistent with low power as with genuine equivalence. We instead test equivalence directly, with two one-sided tests at $\alpha = .05$ against a margin of ± 5 pt. The margin is not chosen for convenience: 5 pt is the deviation of our least-consistent human rater, so the test asks whether a judge is at least as close to the human consensus as the weakest human we actually recruited. Operationally, a judge passes when its

Judge	Reference (oracle)	VLM w/o goal	VLM w/ goal	Percep. only
V_1 GPT-5.5	-43	+28	+24	+49
V_2 Gemini 3.1 Pro	-54	+27	+25	+36
V_3 Opus 4.8	-29	+39	+12	+39
V_4 Sonnet 5	-29	+36	+28	+29
V_5 Qwen3.6-27B	-42	+32	+20	+30
V_6 Gemma-4-12B	-21	+33	+20	+18
H_1	-76	+37	+36	+52
H_2	-62	+21	+10	+42
H_3	-24	+22	+14	+22
H_4	-24	+20	+23	+23
H_5	-50	+37	+16	+46
H_6	-34	+37	+14	+36
Sign	all -	all +	all +	all +

Table 5: Preference margin (% ours - % opponent) over the 100 representative scenes. Negative means the judge prefers the opponent. Every judge ranks ours below the oracle and above every ablation.

90% interval lies entirely inside $(-5, +5)$.

Table 6 reports the outcome. GPT-5.5 — the judge that produces the 1,000-scene results of §5.3 — and Gemini 3.1 Pro both pass, as does Qwen3.6-27B. Four of the six human raters do not. The comparison is therefore not that these models are tolerably close to human judgment, but that they sit closer to the human consensus than most individual humans do. Widening the margin to ± 6 pt, an obvious sensitivity check, changes no VLM verdict and admits one further human (H_2). The margin is anchored in the data rather than prespecified, so we also report the stricter direction: at ± 4 pt both frontier judges sit exactly on the boundary (each CI lower bound is -4.0) and only Qwen3.6-27B and H_1 still pass, likewise at ± 3 pt. The anchoring rater does not pass the margin they define (H_4 , $\Delta = -5.0$, CI $[-8.9, -1.1]$).

What the metric does not capture. The commit column of Table 6 is reported because the strict metric is scored only on pairs a judge commits to, and judges differ sharply in how much they commit and why. Human abstentions are almost entirely declared ties — honest statements that two arrangements are comparable — with order-inconsistency accounting for 4–7% of the abstentions of H_3 , H_4 and H_5 . Several models abstain the opposite way: 98% of Qwen3.6-27B’s abstentions and 91% of GPT-5.5’s are self-contradictions rather than declared ties. Discarding those pairs lets a judge be scored on a subset it selected, and the selection favors the pairs it found easy.

Qwen3.6-27B is the clearest case, and we report it as a caution rather than a result: it attains the highest agreement of any VLM (83.1%) while committing on only 66% of pairs and contradicting itself on a third of them. Charging each self-contradiction as a disagreement instead of discarding it — keeping declared ties exempt, since those are informative — moves Qwen from $+0.1$ to -14.2 and Gemma from -7.8 to -22.7 , while the human ceiling falls only from 83.0% to

Judge	Cmt.	vs. H	Δ	90% CI	Eq.
V_1 GPT-5.5	78.0	81.4	-1.6	$[-4.0, +0.7]$	yes
V_2 Gemini 3.1 Pro	79.5	81.1	-1.8	$[-4.0, +0.3]$	yes
V_3 Opus 4.8	69.8	79.4	-3.5	$[-6.3, -0.8]$	-
V_4 Sonnet 5	66.0	78.2	-4.8	$[-7.8, -1.9]$	-
V_5 Qwen3.6-27B	66.0	83.1	+0.1	$[-2.3, +2.4]$	yes
V_6 Gemma-4-12B	56.0	75.1	-7.8	$[-11.5, -4.2]$	-
H_1	81.8	83.2	+0.3	$[-2.2, +2.9]$	yes
H_2	64.8	84.8	+2.8	$[+0.3, +5.1]$	-
H_3	41.5	82.7	-0.4	$[-4.4, +3.5]$	yes
H_4	50.5	79.4	-5.0	$[-8.9, -1.1]$	-
H_5	63.8	86.3	+5.2	$[+3.0, +7.3]$	-
H_6	63.8	80.6	-3.5	$[-6.7, -0.5]$	-

Table 6: Equivalence to a human rater. **Cmt.** is the percentage of the 400 folded verdicts on which the judge names a winner rather than abstaining; **vs. H** is agreement with the other human raters over pairs both committed on; Δ is that value minus the agreement those humans reach among themselves. **Eq.** is yes when the 90% interval lies inside the ± 5 pt equivalence margin (TOST, $\alpha = .05$). All values are percentages.

70.0% and Gemini, the most decisive judge, becomes the best model at -4.4 . We nevertheless report the discarding convention as primary because it matches how the protocol actually consumes verdicts: order-inconsistent comparisons are excluded from every win rate in this paper, so a judge’s inconsistency costs effective sample size rather than introducing wrong answers. The conclusion this qualifies is a narrow one: the equivalence result for GPT-5.5 and Gemini 3.1 Pro is a statement about the verdicts they commit to, and both commit to roughly four pairs in five.

Limitation. GPT-5.5, the judge behind the 1,000-scene results, is also the system’s internal VLM, and the judging rubric names the same properties the program enforces. The human study above and the three-judge direction agreement of Table 2 bound this risk for directional conclusions, and what the judge is shown bounds it further: it receives only post-execution observations, rendered from the same camera by the same executor for every method (§5.1), and never the goal image, the program, or any other internal artifact. Its own contribution to a result has passed through grounding, refinement, execution, and rendering before it sees anything, so there is no channel through which it could recognize its own output rather than judge the arrangement. Magnitudes under an equally capable independent judge remain to be reported.

A.5 Real-Robot Experiment Details

Protocol. The real-robot trials of §5.6 cover four representative tabletop domains — a dining table, a kitchen counter-top, a study desk, and a workbench. Each domain is run five times, and between runs its objects are re-shuffled into a fresh cluttered configuration rather than reset to the previous start state, so the five runs of a domain differ in what is on the table and where. That gives 20 trials in total. Every trial receives a single RGB-D observation and the instruction “Organize it.”,

and is then carried out end to end without human intervention: goal generation, program synthesis, metric grounding, refinement, execution planning, and physical execution.

The 20 shuffled trials. Figure 9 shows the observation, the generated goal, and the derived target configuration for every trial. The layout row is rendered from RecGen meshes reconstructed from a single view, so it carries reconstruction artifacts that the physical scene does not: an object’s mesh may be coarse, mis-scaled, or missing surface detail. These artifacts belong to the render, not to the program, which places entities by the pose and extent grounding assigns them.

Three cases in the figure are worth reading closely, because each is a defect that reaches the program’s input and is nonetheless absorbed.

The study-desk pens are the clearest. Pose estimation on a pen standing in a holder is unreliable: the object is thin, largely occluded, and its visible extent barely constrains its axis. The program does not depend on that estimate. Its declaration is `in_holder`, so the pen is routed to the insertion skill of §F.3, which places it by the holder’s opening rather than by the pen’s own recovered pose. A relation the VLM reads correctly thus substitutes for a measurement perception reads poorly.

Two study-desk goals contain hallucinated objects, items the generator drew that are not on the table. They never enter the arrangement: program synthesis is constrained to the inventory registered from the real observation (§3.3), so an entity with no real counterpart has nothing to bind to and is dropped.

One kitchen-counter layout appears to be missing an object. It is not. RecGen returned a mesh for the white shaker at an anomalous scale, which the renderer draws too small to see. The system assigned that entity a target position as usual, and the program declared its shared line with the remaining shakers, so the arrangement is complete even though the render does not show it. The failure is in the reconstruction that draws the figure, not in the configuration the figure is meant to depict.

Pipeline reliability. The pipeline completed in all 20 trials. Every trial produced a goal image, a program that passed the static checks of §B.4, a grounded and refined layout, an executable move order, and a physical execution; none required a restart, a hand-edited intermediate artifact, or a fallback to a different scene. We state this as a statement about the *pipeline*, not about the arrangements: it means no stage failed to produce a usable output for the next one, not that every object finished within tolerance of its target.

Where the residual error comes from. The stages that derive the target configuration Q^* — goal generation, program synthesis, metric grounding, refinement, and move ordering — were reliable across the trials: each produced a usable result every time, and none was the cause of a visible defect in a finished scene. The error that remains is concentrated in the last step, physically carrying the plan out, and it has two sources.

The first is grasping. The gripper can close off-center on an object, an object can slip or rotate while it is being carried,

and the arm can brush a neighbor on its way. Each of these costs placement accuracy. Because execution is open loop (Appendix F.3), an object that gets nudged is not noticed at the time, and the moves that follow are not re-planned around its new position.

The second is the gap between the real scene and the observation of it. Every stage after Stage 1 works on reconstructed geometry: a goal image whose depth is estimated and rectified, a mesh completed from a single viewpoint, and an extrinsic calibration. Each is close but not exact, so the pose the arm is commanded to reach differs slightly from the pose the program intends. The discrepancy is largest for small and thin objects, whose reconstruction is the least constrained by the observation. This is the error the post-execution repair stage of §F.3 exists to absorb, and it is why that stage grades the declared relations against a fresh observation instead of trusting the commanded poses.

Neither source is specific to our system — both are properties of open-loop pick-and-place under imperfect perception — but together they set what the real-robot trials establish: that the arrangement a generated image specifies can be recovered and physically realized, not that it is realized to simulator precision.

Wall-clock cost. We timed the same 20 runs from the RGB-D observation to an executable target configuration, that is, goal generation, program synthesis, metric grounding, refinement, and move ordering. Robot execution is not included, nor is the simulated teleport-and-refine stage, which is skipped on real scenes. A run takes 208.0 s on average (median 203.8, s.d. 31.5, range 168.0–288.6), or roughly three to four minutes. Per domain the means are 228.7 s for the dining table, 215.3 s for the kitchen counter, 196.6 s for the study desk, and 191.4 s for the workbench.

Two conditions qualify these numbers. Runs were strictly serial, one scene at a time, so each had the whole reconstruction fleet to itself; scenes run concurrently would queue behind one another and no single wall clock would be representative. And the model services were already warm, so these are not cold-start figures.

The cost is dominated by the two VLM calls rather than by geometry. Goal image generation and program synthesis together account for 64.3% of the wall clock on average, and object count acts almost entirely through program synthesis: it averages 86.8 s on the ten-object dining table against 50.4–67.6 s elsewhere, while the geometric stages barely move. The geometry tail is also the stable part — layout preview, goal depth matching, and layout optimization took the same 1, 1, and 5 s in all 20 runs. Fleet fan-out is what keeps reconstruction off the critical path: per-object RecGen averages 15–25 s and per-object RGBTrack 18–28 s, yet reconstructing all six to ten objects of a scene costs 20–71 s and pose estimation 24–48 s. The one mechanism that pushes a run past four minutes is not inference but goal segmentation: in 2 of 20 runs, ambiguous masks in the goal image made reference-mask assignment and its SE(3) check cost about 120 s more than usual.

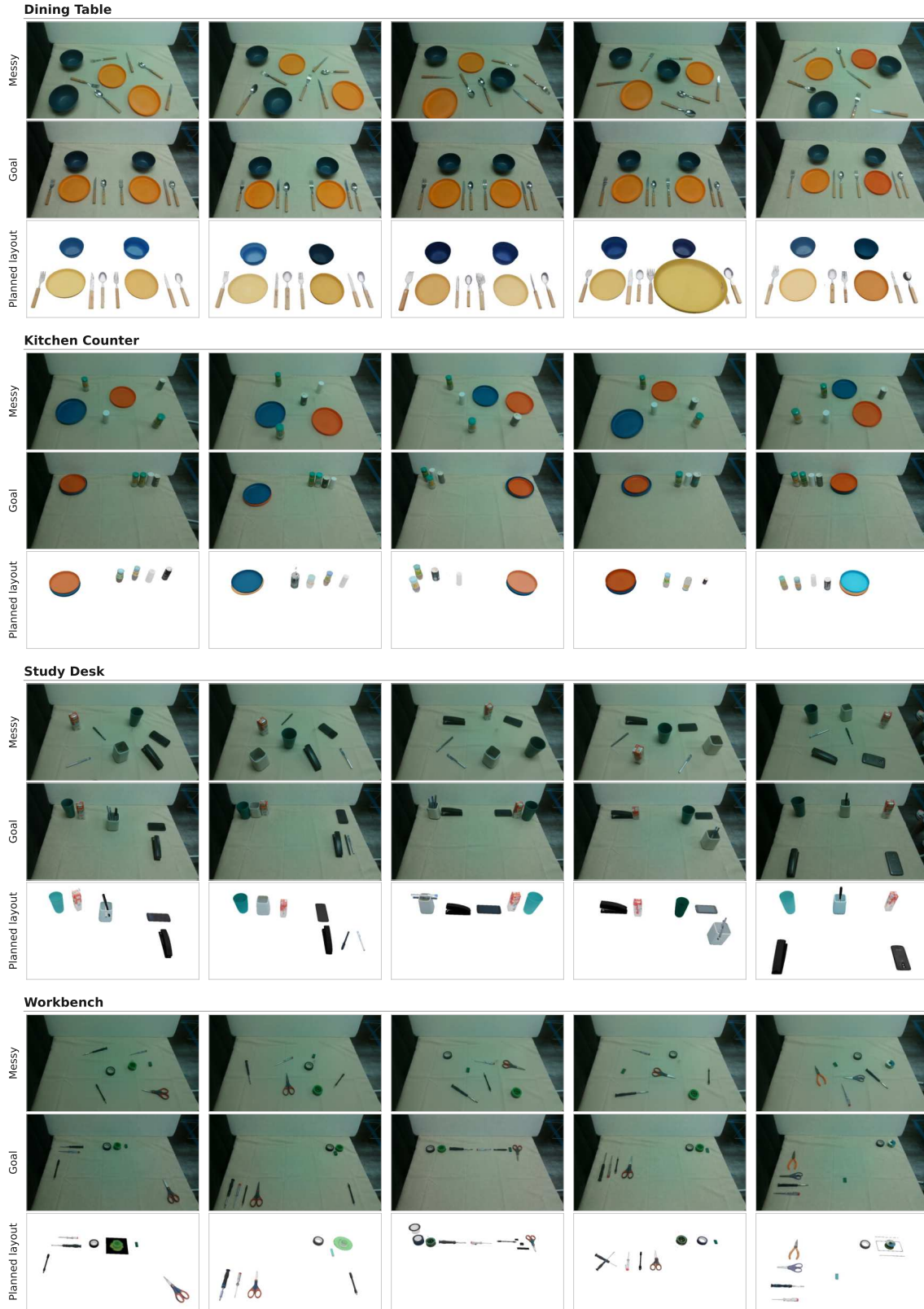


Figure 9: All 20 real-robot trials. Each block is one domain and each column one trial; within a block the rows are the messy RGB-D observation the system receives, the goal image it generates, and the target configuration its program derives, rendered from the reconstructed meshes. The layout row is a render of Q^* before execution, so it shows what the system decided rather than what the arm achieved.

A.6 Statistical Significance

Procedure. Every comparison in this paper is a paired preference: the same scene is shown to the same judge under both presentation orders, and the outcome is a win, a loss, a tie, or an order-inconsistent verdict. Ties and inconsistent verdicts carry no directional information, so we restrict each test to the *decided* pairs, $n_{\text{dec}} = \text{wins} + \text{losses}$, and apply a two-sided exact binomial (sign) test of $H_0 : \Pr(\text{win}) = 0.5$. Interval estimates are Clopper–Pearson exact 95% confidence intervals on the win rate among decided pairs. Because each table reports several comparisons, we also give Holm-corrected p -values within each table; the correction changes no conclusion in Table 7. All p -values are computed exactly from the binomial coefficients rather than by normal approximation. These tests treat scenes as independent; Appendix G repeats them with the 100 dataset templates as clusters.

Main results. Table 7 tests the ten comparisons of Table 3. ORGANIZEIt is preferred over every ablation at $p \leq 2 \times 10^{-16}$, with confidence intervals whose lower bounds are at least 62%, and the oracle is in turn preferred over ours ($p \leq 1.8 \times 10^{-6}$) — the benchmark separates all five systems decisively at this sample size.

Sensitivity to inconsistent verdicts. The tests above exclude order-inconsistent verdicts, so we bound their possible influence using the rates of Table 3. Splitting each comparison’s inconsistent share equally between win and loss leaves every direction unchanged. Under the adversarial extreme that counts every inconsistent verdict against us, the three tidiness comparisons still stand (56.6 vs. 37.4, 52.9 vs. 45.5, 59.8 vs. 39.7), while the three user-intent comparisons narrow past the boundary (43.5 vs. 44.7, 47.0 vs. 51.2, 47.8 vs. 51.8). The tidiness conclusions are therefore robust to any resolution of the inconsistent verdicts; the user-intent margins hold under the even split but not under the adversarial extreme, and should be read with that caveat.

Power on the SG-Bot benchmark. Table 8 repeats the analysis for Table 2. Our system wins on every judge and both criteria by win rate, but only GPT-5.5 on tidiness ($p = 0.037$), GPT-5.5 on user intent ($p < 10^{-6}$), and Claude Opus 4.8 on user intent ($p = 0.050$) reach $\alpha = 0.05$ unadjusted. This reflects the size of the benchmark rather than the absence of an effect: SG-Bot’s benchmark contains 90 scenes, which leaves only 64–78 decided pairs after ties and inconsistent verdicts are removed. At the observed win rates, a two-sided exact test on that many pairs has a power of only 20% to 46% in the non-significant cells, and would need 123 to 411 decided pairs to reach the conventional 80%. We therefore read Table 2 as a consistent directional result on a small established benchmark, and treat the 1,000-scene evaluation of Table 3, where every comparison is significant by several orders of magnitude, as the statistically powered evidence.

Representative set. Two analyses are run on a 100-scene representative subset rather than the full benchmark: the plan-before-edit ablation (Table 4) and the human validation of the judging protocol (§5.5). The subset is built by sampling one scene from each of the 100 templates, so it covers every

template exactly once and contains 100 distinct scenes. Both analyses use the same subset.

Plan-before-edit ablation. Applying the same test to Table 4, planning wins on tidiness with 51 wins against 22 losses ($n_{\text{dec}} = 73$, 69.9% [58.0, 80.1], $p = 9.1 \times 10^{-4}$) and on user intent with 85 against 5 ($n_{\text{dec}} = 90$, 94.4% [87.5, 98.2], $p = 7.5 \times 10^{-20}$). Both are significant at $\alpha = 0.05$ despite the smaller subset, because the effect is far larger than the one separating our system from SG-Bot.

A.7 What the Benchmark Can and Cannot Favor

Our tidy templates are annotated with the same DSL that §3.3 uses as a target language, which invites the question of whether the benchmark rewards DSL-shaped arrangements and therefore flatters our system. This subsection traces where the annotation actually enters the evaluation.

The inventory is fixed before any DSL is written. The order in which the dataset is built already rules out the most direct form of the concern. A template’s content — the activity, the situational prompt, and the object list fixing which categories appear and how many of each — is proposed by a VLM from the scenario alone and then curated by hand for everyday plausibility (§4). Only afterwards do annotators author the tidy configuration for that already-fixed inventory, and the DSL is the notation they write it in. No object is admitted because it is convenient to express, and none is dropped because it is not: the annotation is fitted to the inventory rather than the inventory to the annotation. The DSL is thus confined to a single step of construction, one that changes where the given objects go and never what they are. The curated templates carry a median of 7 categories under a median of 15 constraints each, drawn from the same open inventory of 766 assets in 100 categories that every evaluated system must handle.

Where the DSL enters, and where it does not. The DSL annotation is used for exactly two things: instantiating the tidy scene of each dataset entry, and defining the *Reference goal* oracle of §5.3. It does not touch the input any system sees. The evaluated observation is the *messy* scene, which carries no DSL constraint at all: every object is dropped from above onto a random unoccupied table region and settles under physics (§4), so its layout is a physical outcome rather than an annotated one. Measured across the benchmark, the within-template standard deviation of object xy position in the messy scenes is 19.3 cm against 21.6 cm across the whole dataset — the input a system is asked to organize retains 89% of the dataset-wide spatial spread even among scenes built from the same template. No evaluated system receives the template, the constraint file, the DSL program, or the tidy configuration; each receives the messy RGB-D observation and the task prompt, and our own program is extracted from a generated goal image rather than read off the annotation. The judge, in turn, sees two rendered post-execution images and the task prompt, with system identities withheld and presentation order swapped (§5.1); it is never shown the annotation, the reference configuration, or which arrangement came from a program.

Comparison	Crit.	W	L	n_{dec}	Win% [95% CI]	p (Holm)
vs. Perception-only	Tid.	566	177	743	76.2 [72.9, 79.2]	3×10^{-47}
vs. VLM w/ goal	Tid.	529	200	729	72.6 [69.2, 75.8]	3×10^{-34}
vs. VLM w/o goal	Tid.	598	201	799	74.8 [71.7, 77.8]	1×10^{-45}
vs. Reference goal	Tid.	236	604	840	28.1 [25.1, 31.3]	5×10^{-37}
VLM w/ vs. w/o goal	Tid.	427	330	757	56.4 [52.8, 60.0]	9.5×10^{-4}
vs. Perception-only	UI	435	221	656	66.3 [62.6, 69.9]	2×10^{-16}
vs. VLM w/ goal	UI	470	214	684	68.7 [65.1, 72.2]	3×10^{-22}
vs. VLM w/o goal	UI	478	251	729	65.6 [62.0, 69.0]	2×10^{-16}
vs. Reference goal	UI	325	466	791	41.1 [37.6, 44.6]	1.8×10^{-6}
VLM w/ vs. w/o goal	UI	305	376	681	44.8 [41.0, 48.6]	7.3×10^{-3}

Table 7: Exact binomial tests for Table 3 (1,000 scenes). n_{dec} is the number of decided pairs; win% and the confidence interval are computed among those. Holm correction is applied across the ten tests.

Judge	Crit.	W	L	n_{dec}	Win% [95% CI]	p (Holm)	Pow.
GPT-5.5	Tid.	47	28	75	62.7 [50.7, 73.6]	0.037 (0.18)	55%
Opus 4.8	Tid.	43	26	69	62.3 [49.8, 73.7]	0.053 (0.20)	46%
Gemini 3.1 Pro	Tid.	47	31	78	60.3 [48.5, 71.2]	0.089 (0.20)	37%
GPT-5.5	UI	52	12	64	81.2 [69.5, 89.9]	4.6×10^{-7}	100%
Opus 4.8	UI	42	25	67	62.7 [50.0, 74.2]	0.050 (0.20)	55%
Gemini 3.1 Pro	UI	41	31	72	56.9 [44.7, 68.6]	0.289 (0.29)	20%

Table 8: Exact binomial tests for Table 2 (90 scenes). Unadjusted p ; Holm-corrected values appear in parentheses. Power is the probability that a two-sided exact test at $\alpha = 0.05$ would detect an effect of the observed size at this n_{dec} .

The comparison is internal to one pipeline. Rows 1–3 of Table 3 compare ORGANIZEIT against ablations of ORGANIZEIT. Every arm shares the goal image, segmentation, registration, refinement, executor, renderer, camera, and judges, and differs only in the channel under test (§5.3). A benchmark-level preference for DSL-shaped layouts, if one existed, would act identically on all four arms and so could not open a margin between them. The margins are also not ordered by whether an arm writes a program: *Perception-only* uses no program at all and loses on tidiness by 38.9 points, which falls between the two program-writing arms (32.9 and 39.7). If the evaluation were rewarding programs as such, the one arm without a program would be the outlier, and it is not.

The oracle row runs the other way. The single place the annotation appears as a competitor is the *Reference goal* row, and there it beats us: 60.4% to 23.6% on tidiness. A benchmark tilted toward our compiled output would not rank that output below the annotated configuration by the largest margin in the table. What the row shows instead is a judge that prefers the human-curated layout to the machine-produced one, which is the direction an unbiased evaluation of a system still short of human quality should produce. That the judge’s preferences are human-like rather than annotation-shaped is established independently in §5.5: on the 100-scene subset, VLM verdicts agree with human raters at a rate statistically indistinguishable from human–human agreement, and those raters saw only rendered images and never any part of the DSL.

B Arrangement DSL: Syntax and Semantics

This appendix gives the full syntax and semantics of the arrangement DSL summarized in Table 1, the static checks that decide whether a synthesized program is well formed, and the evaluator that turns a program into a target configuration. It closes with statistics over the 16,865 relation declarations that the system wrote across the 1,000 scenes of §5.3.

B.1 Design Rationale

The language exists to make one distinction operational: which properties of a depicted arrangement are *structural* and must survive metric estimation, and which are *measurements* that estimation is supposed to supply. Three commitments follow.

Relations own the structure; perception owns the numbers. Every relation separates a discrete part, fixed by the VLM, from a continuous part, left open for perception. A row of three items is declared once as a shared line plus an even-spacing group; what the spacing *is* never appears in the VLM’s output. This is what lets the two channels of §3.3 and §3.4 disagree about millimeters without disagreeing about whether the row is a row.

Single assignment over pose variables. Each object exposes exactly three scalar variables, $o.x$, $o.y$, and $o.\psi$; every declaration publishes the write-set it claims, and each variable must be claimed exactly once. A pose may therefore be assembled from several relations — a keyboard’s x from a shared column with the monitor, its y from an absolute placement — while remaining free of the silent overwrites

that make a chain of pairwise offsets ambiguous. Height is deliberately not a variable: z follows from the table plane or from a declared support relation together with the reconstructed geometry, so no declaration can put an object at an unsupported height.

Well-formedness is decidable without geometry. Because write-sets and read-sets are declared rather than inferred, inventory coverage, define-before-use, and write conflicts are checked symbolically, before any mesh is touched. A malformed program is rejected with the two declarations that collide, which turns program synthesis into a loop the VLM can close by itself (§B.4).

B.2 Program Form and Coordinate Conventions

A program is a JSON object with a single key, `constraints`, holding an ordered list of relation declarations; nothing else is accepted at the top level. Object references are the `object_id` strings of the registered inventory, and a declaration naming an unregistered identifier is rejected. Writing S for the list of declarations stripped of its continuous arguments and θ for those arguments, a program is the pair $P = (S, \theta)$ of §3.3.

Positions live in the table frame, seen from the user’s side of the table: x increases to the right, y increases away from the user, and the table center is the origin. Yaw ψ is the rotation about the table normal, in degrees. Extraction emits θ in table-normalized units, where the table edges sit near ± 1 on each axis, because the VLM reads the goal image and not a ruler; grounding replaces those values with meters in the same frame (§3.4). The skeleton S carries over unchanged, so the normalized program and the grounded one are the same program under two unit systems.

B.3 The Relation Vocabulary

Table 9 lists the twelve relation types with their read-sets, write-sets, and the split of their arguments between S and θ . Table 1 in the main text groups them by family: `table_x/y` covers the three absolute placements, `x/y_offset_from` the three offsets, and `in_same_line` the two shared-coordinate relations.

Two entries deserve comment. `on_top_of` writes no pose variable at all: it declares that one object rests on another, which fixes the supported object’s height and assigns it to a stacking layer, while its planar variables must still be written by other declarations — concentric stacking is expressed as `on_top_of` plus an offset of zero. `in_holder`, by contrast, writes all three variables at once, because inserting a pen into a cup determines position and yaw together; an object placed this way therefore admits no further declaration, not even `align_axis`.

Orientation. `align_axis` takes one of four discrete values. `horizontal` and `vertical` put the object’s long axis along x and along y respectively; `any` marks an orientation that carries no meaning, as for a plate or a bowl, and leaves the downstream stages free; `custom` marks a visibly off-axis placement whose angle the VLM does not attempt to state, and is the one case in which `align_axis` contributes

a continuous field for perception to fill. This is the mechanism behind the claim of §3.4 that off-axis orientations are measured rather than named.

B.4 Static Well-Formedness

A candidate program is accepted only if it passes four checks, all of them symbolic. **Completeness:** every registered object has x , y , and ψ written. **Single assignment:** no variable is written twice, within a declaration or across declarations. **Define-before-use:** every variable a declaration reads was written by an earlier one, which makes the anchor structure a directed acyclic order rather than a system to be solved. **Layer consistency:** an even-spacing group may not contain both objects of an `on_top_of` pair, since spacing an object against the very object it rests on is not realizable; shared-line relations carry no such restriction and may cross layers, so a mouse on a mousepad can still share a column with the monitor.

Schema-level conditions are checked alongside: each declaration carries exactly the keys its type defines, the anchor belongs to the group it anchors, `order` is a permutation of `objects`, `spacing` is non-negative, and an object is never its own anchor.

Rejection is reported as a machine-readable code together with the offending declaration index; an over-definition additionally names the earlier declaration that claimed the variable, so the report identifies both ends of the conflict. That report, the rejected program, and the original prompt are returned to the VLM, which may retry, to a maximum of three attempts.

Across the 1,000 scenes of §5.3, 994 programs (99.4%) were accepted on the first attempt. All 6 rejections were the same check firing — an even-spacing group that mixed a stacked object with its support — and all 6 were accepted on the second attempt, so one round of feedback repaired every rejected program. No program needed a third attempt, and every program in the benchmark was ultimately accepted. The check that fires is also the one the language cannot make unnecessary: completeness, single assignment, and define-before-use are properties of the program text that the prompt can and does teach, whereas layer consistency couples two declarations through a stacking relation that may be several lines apart, which is the one condition the VLM occasionally misses on a first pass.

B.5 Evaluation Semantics

$\text{Eval}(S, \theta)$ maps a well-formed program to a complete configuration in two passes.

Pass 1: orientation. Every `align_axis` declaration is resolved first, independently of the others, since it reads nothing. Let w and h be the object’s footprint extents along the two table axes in its local frame. Then `any` yields 0° ; `horizontal` yields 0° if $w \geq h$ and 90° otherwise; `vertical` yields the complement; and `custom` takes the angle supplied in θ , for which the ungrounded layout render substitutes a fixed 45° placeholder. Yaw is resolved before position because footprint spacing depends on how much room a rotated object occupies along the spacing axis.

Relation	Reads	Writes	Discrete arguments (S)	Continuous arguments (θ)
table_x	—	$o.x$	object	x
table_y	—	$o.y$	object	y
table_xy	—	$o.x, o.y$	object	x, y
x_offset_from	$a.x$	$o.x$	object, anchor	dx
y_offset_from	$a.y$	$o.y$	object, anchor	dy
xy_offset_from	$a.x, a.y$	$o.x, o.y$	object, anchor	dx, dy
in_same_vertical_line	$a.x$	$o.x$ for each $o \neq a$	objects, anchor	—
in_same_horizontal_line	$a.y$	$o.y$ for each $o \neq a$	objects, anchor	—
evenly_spaced_from_anchor	$a.u$	$o.u$ for each $o \neq a$	objects, anchor, order, axis: $x \mid y$, mode: $obj_center \mid footprint$	spacing
align_axis	—	$o.\psi$	object, axis: $horizontal \mid vertical \mid custom \mid any$	yaw, when axis is custom
on_top_of	—	— (support only)	object, anchor	—
in_holder	$h.x, h.y$	$o.x, o.y, o.\psi$	object, holder	—

Table 9: Complete relation vocabulary. o is the constrained object, a the anchor, h the holder, and $u \in \{x, y\}$ the declared axis. Reads must have been written by an earlier declaration; writes must be unclaimed. Arguments in the S column are discrete and fixed by the VLM; those in the θ column are continuous and filled by perception.

Pass 2: position. The declarations are then swept once in order, each writing the variables it claims:

- `table_x`, `table_y`, `table_xy` write their literal arguments.
- `x_offset_from` writes $o.x = a.x + dx$, and `y_offset_from` and `xy_offset_from` analogously.
- `in_same_vertical_line` writes $o.x = a.x$ for every member other than the anchor; `in_same_horizontal_line` does the same for y .
- `evenly_spaced_from_anchor` propagates outward from the anchor in both directions along $order$. The step between consecutive members a and b is $\delta(a, b) = s$ in `obj_center` mode and $\delta(a, b) = s + (e_u(a) + e_u(b)) / 2$ in `footprint` mode, where s is the declared spacing and $e_u(o)$ is the extent of o along the spacing axis u at the yaw resolved in pass 1. Footprint mode therefore declares an edge-to-edge gap, which stays collision-free when objects of different size share a row.
- `on_top_of` has no numeric effect.
- `in_holder` copies the holder’s x, y , and yaw.

Because single assignment and define-before-use hold, the sweep needs no iteration, no ordering heuristic, and no conflict resolution: each variable is written once, at the moment its declaration is reached, from values already present. Eval is thus a deterministic function of (S, θ) , and the configuration it returns satisfies every declaration by construction rather than by optimization.

Grounded evaluation. The grounded pass calls the same evaluator on $(S, \hat{\theta})$, with two differences. Extents are metric rather than table-normalized, and yaw is supplied directly from the registered goal-side pose of each object rather than derived in pass 1 — so `align_axis` contributes no number to the grounded configuration, and its discrete value acts ear-

lier, as a constraint on the render-and-compare pose search. The result is Q^* .

B.6 A Worked Example

Scene `oi_0003` holds a plate, a slice of bread, a coffee cup, a phone, and a carton of milk. The synthesized program is thirteen declarations:

Listing 1: Extracted program for `oi_0003`, with θ in table-normalized units. Eight declarations fix structure; five carry the numbers that grounding overwrites.

```
{ "type": "table_xy", "object": "plate", "x": -0.10, "y": -0.35 }
{ "type": "align_axis", "object": "plate", "axis": "any" }
{ "type": "on_top_of", "object": "bread", "anchor": "plate" }
{ "type": "xy_offset_from", "object": "bread", "anchor": "plate", "dx": 0.0, "dy": 0.0 }
{ "type": "align_axis", "object": "bread", "axis": "horizontal" }
{ "type": "in_same_horizontal_line", "objects": [ "plate", "coffee_cup" ], "anchor": "plate" }
{ "type": "x_offset_from", "object": "coffee_cup", "anchor": "plate", "dx": 0.42 }
{ "type": "align_axis", "object": "coffee_cup", "axis": "any" }
{ "type": "xy_offset_from", "object": "phone", "anchor": "plate", "dx": -0.48, "dy": -0.12 }
{ "type": "align_axis", "object": "phone", "axis": "vertical" }
{ "type": "in_same_vertical_line", "objects": [ "phone", "milk" ], "anchor": "phone" }
{ "type": "table_y", "object": "milk", "y": 0.55 }
{ "type": "align_axis", "object": "milk", "axis": "vertical" }
```

The program exercises most of the vocabulary. The plate anchors the scene absolutely. The bread is stacked on it and placed concentrically, which is why `on_top_of` appears alongside a zero offset rather than instead of one. The cup shares the plate’s row, so only its x is free and a single-axis offset supplies it. The milk shares the phone’s column, so only its y is free and `table_y`, not `table_xy`, supplies it — the two-axis form would claim x a second time and be rejected. Each of the five objects has x , y , and ψ written exactly once.

Grounding leaves all thirteen declarations untouched and rewrites the eight numbers they carry, as Table 10 shows.

Declaration	Field	Extracted	Grounded (m)
<code>table_xy(plate)</code>	x	−0.10	−0.063
	y	−0.35	−0.156
<code>xy_offset_from(bread)</code>	dx	0.00	0.000
	dy	0.00	0.000
<code>x_offset_from(coffee_cup)</code>	dx	0.42	0.197
<code>xy_offset_from(phone)</code>	dx	−0.48	−0.278
	dy	−0.12	+0.001
<code>table_y(milk)</code>	y	0.55	0.283

Table 10: Continuous parameters of Listing 1 before and after grounding. The VLM’s seeds serve only the layout render of §3.4; the phone’s y offset shows why, moving from a guessed −0.12 to a measured +1 mm, which is the goal image saying that the phone in fact sits in the plate’s row.

Refinement (Appendix D) then moves the milk 1 cm toward the user to clear the table margin, by changing the one number in θ that no other declaration reads. Every other parameter is left alone, and in the executed layout the cup’s y equals the plate’s y and the milk’s x equals the phone’s x exactly, as declared.

B.7 Corpus Statistics

Across the 1,000 scenes of §5.3 the system wrote 16,865 declarations, 16.9 per program on average (median 16, range 8 to 39) over 7.45 objects. Table 11 gives the breakdown.

Relation	Count	Share
<code>align_axis</code>	7,325	43.4%
<code>table_xy</code>	2,124	12.6%
<code>in_same_horizontal_line</code>	1,639	9.7%
<code>x_offset_from</code>	1,542	9.1%
<code>xy_offset_from</code>	1,534	9.1%
<code>on_top_of</code>	731	4.3%
<code>evenly_spaced_from_anchor</code>	625	3.7%
<code>in_same_vertical_line</code>	546	3.2%
<code>y_offset_from</code>	331	2.0%
<code>table_y</code>	233	1.4%
<code>in_holder</code>	125	0.7%
<code>table_x</code>	110	0.7%

Table 11: Relation usage over the 1,000 extracted programs.

Every object receives exactly one orientation declaration unless `in_holder` places it, which writes yaw itself. The

two counts close exactly — 7,325 `align_axis` plus 125 `in_holder` declarations account for all 7,450 objects in the benchmark — which is an independent check that the single assignment invariant held on every program the system wrote. This is also why `align_axis` accounts for 43.4% of all declarations, and why its distribution is informative on its own: 45.5% of objects are *any*, 26.4% *vertical*, 24.5% *horizontal*, and 3.6% *custom*. That last figure is small but not negligible — 261 objects across the benchmark are placed at an angle the VLM declines to name and perception measures instead.

Structural relations outnumber absolute placements. Shared lines, even-spacing groups, and offsets account for 36.9% of declarations against 14.6% for the three `table_*` forms, so the typical object is positioned relative to another object rather than to the table, which is the behavior the synthesis prompt of §3.3 asks for. Among even-spacing groups, footprint mode is chosen 605 times against 20 for `obj_center`.

Because parameter-free relations expose nothing for grounding or refinement to adjust, they are a direct test of the claim that the program survives the metric stages. Over the 2,185 shared-line declarations in the executed layouts, the residual — the spread of the shared coordinate within a declared line — is exactly zero in every case.

Expressivity and the benchmark. The benchmark’s tidy scenes are themselves authored in this DSL during dataset construction, so the evaluation cannot probe arrangements outside the vocabulary: diagonal rows and radial layouts have no dedicated relation and would fall back to chains of offsets. Off-axis structure is not absent, however — the 261 *custom* orientations above are exactly the depicted angles that fall outside the axis-aligned vocabulary and are measured rather than named. Extending the relation set, and evaluating on tidy scenes authored outside it, are left to future work.

B.8 The Relaxed Variant Used by the Ablations

The *VLM w/ goal* and *VLM w/o goal* ablations of §5.3 must state every metric value themselves, since they have no perception channel to fill θ . Two provisions keep that comparison fair rather than merely harder. Their prompt supplies each object’s true mesh dimensions, so a spacing can be chosen against real sizes instead of guessed ones. And `align_axis` exposes a fifth axis value, `delta`, which takes a numeric `delta_deg` argument and rotates the object by that angle from its *current* yaw in the observation. This replaces the `custom` value, which is unavailable to the ablations: where our system marks an off-axis orientation for measurement, the ablations state the angle. The rest of the language, the static checks, and the evaluator are unchanged, so all three systems produce programs in the same class and differ only in where the numbers come from.

C Metric Grounding from Generated Goals

Extraction (§3.3) fixes the skeleton S and leaves θ open. This appendix describes how the perception channel closes θ from the goal image. Three problems have to be solved in order: which pixels of the generated image belong to which

registered entity (§C.1), what metric geometry those pixels imply once the image is given a depth and each mesh a pose (§C.2), and how each open field of each declaration is computed from that geometry (§C.3).

The governing constraint throughout is that perception is never allowed to decide the arrangement. It measures quantities; which quantities matter, and how they compose into poses, is fixed by S . A consequence worth stating early is that a failure of association or registration degrades a value rather than the structure: a mis-measured spacing leaves a row a row.

C.1 Entity–Mask Association

Goal-side segmentation. The goal image is segmented by the same tabletop perception stack used on the real observation, prompted with the names of the registered inventory, and yields a set of candidate item masks together with a goal-side table mask. At this point these are regions, not entities. They carry no identity, and a generated image gives no guarantee that they correspond one-to-one with the real inventory: the generator duplicates objects, merges neighboring ones into a single blob, distorts silhouettes, and occasionally invents an object that was never on the table.

The layout render. Association is made tractable by giving the VLM a second view of the same arrangement whose inventory is authoritative. The extracted program is evaluated with the VLM’s own numeric seeds, and every reconstructed mesh is drawn top-down at the resulting pose and scale, together with each object’s metric bounding box in the table frame. The render is therefore approximately faithful to the depicted layout — it is the same program — while containing exactly the registered objects, each already named. It is a bridge from the symbolic program to something the VLM can compare against pixels.

Assignment. A single VLM call receives the layout render, the program, the goal image with its candidate masks numbered, and the real observation with its own numbered masks, and returns one goal mask per entity. The response is validated against the inventory before it is accepted, and the assignment is written back onto each entity as its reference mask together with the identifier of the raw mask it came from. That identifier, not the mask pixels, is what later stages use to test whether two entities were given the same region.

Many-to-one is allowed, and one-to-many is not. The assignment is deliberately not a matching. Several entities may be assigned the same mask, which is what happens when the generator draws a stack as one silhouette or merges two adjacent objects; masks that no entity claims are simply discarded, which is how a hallucinated object leaves the pipeline without ever reaching the program. What cannot happen is an entity receiving no mask, since the inventory contract requires every registered object to be assigned.

Across the 1,000 scenes of §5.3 the 7,450 entities were assigned 7,310 distinct goal masks. Sharing is real but uncommon: 262 entities (3.5%) share their mask with at least one other, concentrated in 110 scenes (11.0%), and the shared groups are small — 107 pairs, 12 triples, and 3 groups of

four. Sharing is not treated as a failure; §C.3 shows how the program supplies what a shared footprint cannot.

C.2 Goal-Side Geometry and Pose Candidates

Depth, and why it must be rectified. The goal image has no measured depth, so a monocular estimate is computed at the goal image’s intrinsics. A monocular estimate is only approximately metric, and its error is dominated by an unknown scale and offset rather than by shape. We therefore fit a single affine correction, $d' = ad + b$, defined by the requirement that the *visible table surface* land on the world table plane $z = 0$.

The fitting region is the set of pixels that are bare table in *both* images: the goal-side table mask minus every goal-side item mask, intersected with the real table mask minus every real item mask. Restricting to the intersection matters, because a pixel that is table in the generated image but occupied in the real one is not evidence about the plane. At least 200 such pixels are required. Under the camera model each pixel’s world height is affine in its depth, so imposing $z = 0$ over the region gives an overdetermined linear system in (a, b) ; it is solved by least squares, then refit once on the residual inliers, keeping pixels within 3.5 median absolute deviations so that a few mis-segmented pixels cannot drag the plane. The corrected depth is clipped to $[50, 5000]$ mm, and the residual table height before and after correction is recorded for inspection.

The appeal of this construction is what it does *not* need. It requires no correspondence between real and generated objects, no known object scale, and no assumption that the generator preserved the viewpoint exactly — only that some bare table is visible in both images. Together with the goal RGB, the result is a pseudo-RGB-D observation expressed in the real table frame.

Registration. Each entity’s reconstructed mesh is registered against that pseudo-RGB-D observation within its reference mask. Because the depth is only approximately metric and generation can alter apparent size, the search is over scale as well as pose. The registration returns a per-object pre-pose in the camera frame, which is transformed into the world frame.

A pre-pose whose translation disagrees with the image is not trusted: the registered translation is compared against the world point obtained by averaging the depth over the entity’s reference mask, and if the two differ by more than 10 cm the mask-depth point is used instead. This catches the case where registration locks onto the wrong depth layer, which the subsequent orientation search cannot recover from on its own because it holds translation fixed.

Deciding SE(2) or SE(3). Most tidying leaves objects upright, and searching the full orientation space for those objects invites a physically absurd but photometrically plausible answer — a plate scored as if standing on edge. Before the orientation search, a VLM is therefore shown the paired real and goal views and asked, per object, whether the depicted current-to-goal transform is a planar reorientation, SE(2), or a genuine reorientation out of the plane, SE(3). The verdict gates the candidate set rather than the result, and the default

when the verdict is missing or unparseable is SE(3), so the default widens the search instead of silently constraining it.

Candidate set. Orientation is chosen by render-and-compare over a discrete set. The full set is the six canonical base orientations — which face of the object’s bounding box points down — each swept over 12 yaws at 30°, plus the registration pre-pose. The set is then pruned:

- An SE(2) object keeps only the single base orientation closest to its current gravity-stable face, swept over yaw, and *drops* the pre-pose, since a planar move is precisely the case in which the tracked 3D rotation should not be trusted.
- An SE(3) object keeps the whole set, with the pre-pose serving as the candidate that no discretization can express.
- A `horizontal` or `vertical` declaration in the program then keeps only candidates whose projected long axis falls within 30° of the corresponding table axis. If no candidate satisfies the constraint the filter is dropped rather than the object, so an unsatisfiable declaration degrades the search instead of failing the scene.

This is where a discrete argument of S acts on perception: the axis value chosen by the VLM does not supply a number, it removes candidates.

Scoring. Each candidate is rendered at the fixed translation and scored against the goal image inside the reference mask by

$$C = C_{\text{mask}} + C_{\text{depth}} + 0.2 C_{\text{color}},$$

lowest total winning. $C_{\text{mask}} = 1 - \text{IoU}$ between the rendered silhouette and the reference mask. C_{depth} is the median absolute depth difference over their overlap, in millimeters, scaled by 10^{-3} and capped at 1. C_{color} compares *spatial color layout* rather than color: the object’s bounding box is divided into a 2×2 and a 3×3 grid, the mean LAB color of each cell is taken, each signature is centered by subtracting its own mean, and the two signatures are compared cell-wise; the two grid resolutions are averaged. Centering is what makes this usable on generated pixels — the generator shifts an object’s overall color and lighting freely, but it preserves which end of a mug is darker, and that is the part that disambiguates a 180° yaw when silhouette and depth cannot.

The winning candidate supplies each entity’s goal orientation. Its rotation is the measured yaw that Appendix B refers to as overriding `align_axis` in the grounded evaluation, and its center is the entity’s goal-side position.

C.3 Filling Program Parameters

Footprints. The corrected goal depth is back-projected into world points, and each entity’s points are those falling inside its reference mask. Masks on generated images bleed onto whatever lies behind the object, so the points are cut on the far side first: the masked depths are histogrammed in 1 cm bins, bins holding at least 10% of the peak count are treated as occupied, and everything beyond the first gap of two or more empty bins past the dominant cluster is discarded as background. The axis-aligned bounding box of the surviving points gives the entity’s footprint — a center and a range on

each table axis. All of these cross into the DSL frame at this point and once only, so every relation below composes in DSL coordinates rather than mixing frames.

Concentric groups. An object resting concentrically on another is declared as `on_top_of` plus a zero offset, and its footprint is the less reliable of the two — a slice of bread on a plate occupies few pixels and sits at a depth barely distinguishable from the plate’s. Entities linked by a zero offset (below 1.1 cm) are therefore pooled, and the whole group adopts the footprint center of whichever member has the largest reference mask. The declared concentricity is thus realized by construction rather than measured twice and reconciled.

Per-relation filling. The program is then traversed and each open field is written:

- `table_x`, `table_y`, `table_xy` take the corresponding component of the entity’s footprint center.
- The offsets take the difference of the two entities’ footprint centers along the relevant axes. An offset the VLM authored as zero is kept at exactly zero, so concentricity survives as an equality rather than as a small measured residual.
- `evenly_spaced_from_anchor` has its mode set to `footprint` and its spacing measured edge-to-edge. Group members are first deduplicated by reference mask, so a merged silhouette contributes one entity rather than several, then sorted along the declared axis; the spacing is the *minimum* gap between consecutive entities. Taking the minimum rather than the mean matters because the declaration will re-space the whole group uniformly: the mean would let one loosely drawn pair inflate the row until it runs off the table, whereas the tightest observed pair is by construction achievable. A group that collapses to a single entity falls back to 1 cm.
- `align_axis` with axis `custom` is filled by the orientation chosen in §C.2. The remaining relations — `shared_lines`, `on_top_of`, `in_holder` — expose no continuous field and are passed through untouched.

Entities that share a footprint. If an entity and its anchor were assigned the same reference mask, their measured centers are identical and the difference between them is zero, which would collapse the two objects onto each other. There is nothing to measure in this case, and the system does not pretend otherwise: the offset instead keeps the value the VLM authored, converted from the normalized DSL scale to meters against the table’s extent on that axis. This is the concrete mechanism behind the claim of §3.4 that the program, not the registration, separates entities sharing a mask — the 262 entities of §C.1 are placed relative to one another by the depicted structure, while their common footprint still fixes where the group as a whole sits on the table.

Output. The result is the same skeleton with θ replaced by the measured values $\hat{\theta}$, in meters. It is written out before any feasibility reasoning has been applied, and it is this unrefined program that Appendix D takes as input.

D Relation-Preserving Layout Refinement

Evaluating the grounded program gives Q^* , the configuration the goal image depicts as measured through real geometry. It need not be physically realizable. Reconstructed meshes differ in size from the assets they stand for, the monocular depth of Appendix C is only approximately metric, and the generator freely draws objects overlapping or half off the table. The refinement stage removes those defects.

The difficulty is that the obvious formulation is the wrong one. If the declared relations are turned into penalty terms and minimized against a collision term, then relations and feasibility compete, and the solver is free to buy a millimeter of clearance by bending a row — exactly the failure the arrangement program exists to prevent. Our solver is therefore built on a different division: the relations that carry no continuous parameter are made *unbreakable by construction*, and only the parameterized ones are allowed to absorb the correction. No force in the system can break the former, because the displacement that would break them is not representable.

The variables are relation groups, not objects. The solver does not optimize per-object positions. For each table axis independently, objects are merged by union-find according to the program: `on_top_of` and `in_holder` merge their two objects on *both* axes, so a stack is a single rigid body in the plane; `in_same_vertical_line` merges its members on *x* only, and `in_same_horizontal_line` on *y* only. Each resulting group is one scalar variable per axis, and every object in it moves by exactly the same amount along that axis.

This is what makes the refinement relation-preserving. A shared line cannot bend because its members do not have independent coordinates on the shared axis; a book cannot float off the plate it rests on because the two share both coordinates. These are precisely the relations that Appendix B lists as exposing no field in θ — there is nothing for refinement to adjust, and correspondingly no way for it to interfere. Across the 1,000 scenes of §5.3 all 856 stacked or inserted pairs received bit-identical displacements, and all 3,664 shared-line member pairs received bit-identical displacements on their fused axis.

The same fusion has a consequence for collisions. If two objects are *directly* related by a line or a stack, a separating force between them along the fused axis would cancel inside their shared variable and merely waste a step, so that axis is frozen for that pair. The test is a declaration naming both objects, not transitive membership in the same group: two rows that happen to share an anchor do not freeze each other's members.

Four forces. Everything else is expressed as a force on these variables. The forces are not all gradients of a single energy, which is deliberate — two of them are more naturally written as forces than as potentials — and it is why the layout is relaxed by explicit stepping rather than minimized by a quasi-Newton method.

- **Springs.** Every offset relation, and every consecutive link of an even-spacing group, becomes a spring whose rest length is the value measured in Appendix C.3. These are

the only relations permitted to stretch, and stretching them is how the configuration makes room.

- **Collision.** Contacts are evaluated on two representations at once: the reconstructed mesh at its goal pose, complete but approximate, and the observed partial point cloud carried to the same goal pose, sensor-grounded but incomplete. Both are rasterized to 5 mm cells. For a pair, the clearance is a *swept* quantity — bin both objects by cell row along the perpendicular axis and take the tightest facing-edge distance over the rows they share — and the force is proportional to how far that clearance falls short of the target, applied along the separating axis. The swept clearance is monotonic in separation, whereas a sum of per-cell penetrations is not; monotonicity is what keeps explicit stepping from settling into a limit cycle under deep overlap. The target is aimed 1 mm beyond the clearance the layout is actually judged on, so that the residual deficit inherent to a force balance lands inside that margin and the true clearance still clears.
- **Table boundary.** A soft wall acts at a 3 cm inset from the table edge, with a *floor* on its magnitude: an object past the margin feels at least a fixed inward force however hard its neighbors pull, so internal springs cannot wedge it permanently off the table. It is not a hard projection. An object is only ever dragged back *to* the margin line, so a scene that genuinely does not fit settles into a bounded oscillation around that line rather than being reported as solved.
- **Even spacing.** The consecutive springs already make a row approximately even. A further force makes it enforced: the group's gaps are measured edge-to-edge, members are ordered by their *current* positions rather than the declared order, since relaxation may have reshuffled them, and each gap is pulled toward the group's current mean gap by an equal and opposite pair of forces, so the row equalizes without its center drifting.

Relaxation. Integration is overdamped — no mass, no velocity. Each step evaluates all four forces at the current configuration, sums each object's force onto the group variables it belongs to, and displaces every variable by a fixed multiple of that force, clamped to 2 mm per axis. The clamp keeps a deep initial overlap, whose force is enormous, from teleporting an object across the table, and it makes the dynamics local: moving one object only changes its neighbor's spring force at the *next* evaluation, so a disturbance ripples outward one hop per step and the layout is watched relaxing rather than solved in one shot.

The step budget is a ceiling of 500, not a fixed cost. The loop exits the step the layout is clean at the true clearance, which is the only usable termination test: the balance between springs and contacts jitters at sub-millimeter amplitude indefinitely, so a small-force criterion would never fire.

Verdict and outputs. The relaxed configuration is then re-checked at the true thresholds with no solve margin, and is reported as residual if any contact or boundary violation survives. A second, independent check re-derives the layout from the written-out program and re-tests it, together with

the blocking predicate the execution planner of Appendix E will later use; disagreement marks the scene degraded rather than aborting it, leaving the decision to the offline verifier.

Refinement writes the layout in two forms. One is absolute, consumed by execution. The other re-expresses the relaxed layout as an arrangement program: the topology is untouched — every relation, anchor and axis is exactly as extracted — while the metric values are re-measured from the relaxed poses, with each absolute anchor and each offset taking its achieved value and each even-spacing group taking its relaxed mean gap. This second form is what makes the claim of §3.5 checkable after the fact: it is the same S with a different $\hat{\theta}$.

Behavior on the benchmark. Refinement is cheap and usually gentle. The median scene relaxes in 14 steps and the ninetieth percentile in 70; 13 of the 1,000 scenes run to the 500-step ceiling. 970 scenes pass both checks, and 30 are marked degraded. 5,068 of the 7,450 objects (68.0%) move at all, and those that do move by a median of 15.9 mm, a ninetieth percentile of 82 mm, and at most 317 mm.

The division of labour between S and θ is visible in the same numbers. Of the 4,941 continuous offset fields in the benchmark, 60.9% come through refinement unchanged to within 0.1 mm and the median change is zero, but the ninetieth percentile is 31.7 mm and the ninety-ninth is 116 mm: the springs do stretch, and stretch a long way when a scene is crowded. Meanwhile the relations with no parameter to stretch — every stack, every shared line — are preserved exactly, in every scene. Feasibility is bought entirely out of θ .

E Dependency-Aware Execution Planning

A refined layout says where every object ends up, not in what order to put it there. Order matters because a tabletop is its own workspace: an object cannot be lifted while something rests on it, and cannot be set down where something still stands. Some scenes admit no ordering at all without temporary displacement — two objects that must swap places are the smallest example — so the planner must be able to move an object somewhere that is neither its start nor its goal.

E.1 Dependency Graph Construction

Each object is represented on two substrates at once: the reconstructed mesh, complete but only as good as the reconstruction, and the observed partial point cloud, sensor-grounded but incomplete. Both are carried at the object’s current pose and at its goal pose, so every test below can be asked of either geometry.

We write an edge (i, j) to mean that i cannot be handled until j has been. Edges come in three kinds, produced by five tests.

Departure edges. i cannot leave where it is. An object that something rests on cannot depart first, so the current support graph contributes an edge from the lower object to each object on it. Independently, an object may be pinned by an *overhang* that rests on nothing: for each 5 mm cell of i ’s footprint we ask whether another object occupies the 10 cm

slab directly above i ’s surface there, and a blocker that does so in even one cell prevents i from being lifted straight out. This is what stops the planner from sliding a keyboard out from under a monitor arm.

Arrival edges. i cannot be placed yet. Anything currently standing in the column above i ’s goal region blocks it, because the executor brings objects down from above; the test runs over the full column rather than a fixed slab, so a bottle is not planned into the space beneath a lampshade. Separately, a goal that is within 4 cm of an object that has not yet moved is refused, on the grounds that placing i there would consume the access the gripper will need for its neighbor later. Finally, if i ’s goal rests on another object, that support must arrive first — the only edge kind that records a positive prerequisite rather than an obstruction.

Both substrates, then merged. Every geometric test is run twice, once on the mesh and once on the observed cloud, and the two edge sets are merged and deduplicated. The two disagree in opposite directions and that is the point: a reconstruction failure invents blockers the real scene does not have, while the partial cloud misses blockers it never observed. Taking the union is conservative, and when the mesh proves untrustworthy the planner can fall back to the cloud alone (§E.3).

One consistency pass runs before the search. Two objects can each appear to block the other’s departure, which is geometrically impossible for a genuine stack; when the current support graph names one as resting on the other, the contradicting edge is dropped, and when it names neither, both are, since the pair is an artifact of overlapping reconstructions.

Relations that the geometry cannot see. Insertion is invisible to occupancy reasoning, so two facts are injected from the program. Each `in_holder` declaration becomes a goal support edge, because the holder must be standing at its goal before anything is dropped into it. And a holder and its contents are exempted from each other’s goal-occupancy test, in both directions: without the reverse exemption a pen cup is reported as blocked by the very pens that can only arrive by being placed inside it, a dependency nothing can discharge.

E.2 DFS with Buffer Moves

The graph is a function of the current arrangement, not a fixed input. Every move changes what occupies which region, so the search rebuilds it from scratch at each node; Algorithm 1 is stated in those terms.

Which moves are offered. Objects with no outgoing edge are ready and are offered as direct goal moves. If none exists the arrangement contains a cycle, and the cycle is broken by displacing a member to a temporary pose. Two restrictions apply to that choice. A member that is itself blocked from departing cannot be the one to move. And a buffer move is only offered against a dependency it can actually discharge: displacing an object never satisfies an arrival edge, since that edge means another object is stacked on this one and needs it *at its goal*, and offering it anyway makes the search shuffle the same object indefinitely while the object waiting on it never proceeds.

Buffer moves are enumerated after the direct goal moves, so the search reaches them only by backtracking and prefers plans without detours. They are nonetheless enumerated from the very first node rather than only once nothing can move directly. Deferring them is a trap: by the time no direct move remains, the objects that have already gone to their goals have taken the free table space with them, and the scene no longer has room for the detour it needed at the start.

Choosing a buffer pose. A buffer pose is found by sweeping the table in $SE(2)$ — 2 cm cells and 90° yaw steps — for a placement that clears every other object’s current footprint by 4 cm and also clears the goal region the move is meant to release, preferring poses near the object’s current position so the detour stays short. The chosen pose is then dropped vertically to rest on the table.

Search control. Distinct move orders reach identical arrangements, so a plain depth-first search re-derives the same dead end once per permutation that reaches it, which is exponential on a scene with a few stacks. The search therefore memoizes states it has proved hopeless, keyed by the arrived flags and the object poses rounded to the millimeter. The key records the depth at which the state failed, because a state that failed with few moves left may still succeed with more. A branch is capped at 64 moves, and the whole search is given a 10 s wall clock, on the argument that a scene that does not solve quickly does not solve at all.

Last resort. A cycle made purely of departure edges cannot occur among convex objects; it requires a member that is simultaneously the top and the bottom of the ring, which happens when a non-convex object’s canopy reads as resting on its neighbor. In that case one edge — the spurious block the canopy imposes on the object directly beneath its apex — is released, the analysis is re-run normally so the search still decides for itself what the freed object does, and the resulting step is flagged for manual review rather than presented as verified.

E.3 Composite Moves, Fallbacks, and Behavior

Stacks that travel together. Treating every object independently makes the planner disassemble arrangements that need not be disturbed. A plate of bread whose contents sit on it both before and after tidying would be unloaded item by item to buffers, the bare plate moved, and the contents replaced — many moves, heavy buffer pressure, and not how a person moves a plate of food. Such a stack is instead solved as two nested problems. In the outer problem the base and the children that rest on it in *both* arrangements collapse into one rigid composite that travels to the base’s goal in a single move, carrying its children at their current relative poses. In the inner problem, with the base standing at its goal, the children are shuffled into their own goal poses, buffering across the whole table if they block each other. Only children present on the base in both arrangements ride along; one that joins or leaves the stack is planned normally, and stacks deeper than two levels fall back to flat planning.

Fallback ladder. The planner is run as a ladder, each rung tried only if the previous returned no plan: the full graph

Algorithm 1: DFS execution planning

Require: current and goal geometry on both substrates

Ensure: ordered plan of {goal, buffer} moves, or failure

```

1: Solve(state)
2: if all objects arrived then
3:   return []
4: end if
5:  $G \leftarrow \text{BUILDGRAPH}(\text{state})$  {3 edge types, both substrates}
6:  $R \leftarrow$  objects with no outgoing edge in  $G$ 
7: if  $R \neq \emptyset$  then
8:    $A \leftarrow \{\text{GOALMOVE}(o) : o \in R\}$ 
9: else
10:   $A \leftarrow \{\text{BUFFERMOVE}(b) : b \text{ in a cycle of } G, b \text{ free to leave}\}$ 
11: end if
12: for all  $a \in A$  do
13:    $\text{plan} \leftarrow \text{SOLVE}(\text{APPLY}(\text{state}, a))$ 
14:   if  $\text{plan} \neq \text{failure}$  then
15:     return  $[a] + \text{plan}$ 
16:   end if
17: end for
18: return failure

```

from mesh and cloud together; then the cloud alone, which is the right answer when a reconstruction is bad enough to invent blockers; then the deadlock release described above; and finally the same search with the access-clearance rules disabled, which trades the gripper-access margin for a plan.

On the benchmark. Every one of the 1,000 scenes of §5.3 yields a plan. Plans contain 8.07 moves on average (median 7, ninetieth percentile 11, longest 65), comprising 7,334 direct goal moves, 600 buffer moves, and 134 composite moves. Detours are common but small: 378 scenes (37.8%) need at least one buffer move, and those that do average 1.59 of them, though one crowded scene needs 57. Composite moves fire in 102 scenes (10.2%). No plan in the benchmark contains a step produced by the deadlock release, so that rung of the ladder, which is the only one whose output is not fully verified, never determined a result we report.

F Implementation Details and Prompts

F.1 Computing Infrastructure

Hardware and software. Experiments run on a workstation with an NVIDIA GeForce RTX 5090 GPU, an Intel Core Ultra 9 285K CPU, 64 GB of system memory, and Ubuntu 24.04 LTS. Mesh reconstruction (RecGen) and pose estimation (RGBTrack) are the exception: rather than running inline they are reached as inference services, whose worker fleet spans two hosts, this workstation and a second machine with an NVIDIA RTX Pro 6000. The timings of §A.5 are therefore not those of a single-GPU deployment. The software stack is Python 3.11 with PyTorch 2.8.0+cu128, torchvision 0.23.0+cu128, torchaudio 2.8.0+cu128, Triton 3.4.0, and CUDA runtime 12.8.90.

Simulation. Two simulators are used, each matched to the setting it evaluates. The head-to-head comparison with SG-Bot (§5.2) runs inside SG-Bot’s own PyBullet 3.2.7 environment, so that both systems are executed and rendered by the baseline’s native pipeline. All remaining simulated experiments run in RoboTwin (Mu et al. 2025), built on SAPIEN 3.0.0b1 (Xiang et al. 2020); for the record, the copy we used corresponds to commit 668952b (11 February 2026).

Perception components. We use the public releases of SAM 3 (Carion et al. 2025), Depth Anything 3 (Lin et al. 2025), RecGen (Zadaianchuk et al. 2026), RGBTrack (Guo and Yu 2025), and AnyGrasp (Fang et al. 2023), each taken as distributed by its authors during the experiment window below. Monocular depth uses the DA3METRIC-LARGE checkpoint, the variant fine-tuned for metric monocular depth. We did not pin specific commits for these components, and no component was retrained or fine-tuned for this work.

Hosted models. The system calls hosted models rather than locally trained networks, and we report model names and access dates rather than API version strings, since the deployments were not pinned to dated snapshots. All experiments were run between May and July 2026, using each model as deployed after its public release. All OpenAI models — GPT-5.5 for reasoning and GPT-Image-2 (OpenAI 2026) for image generation — are accessed through CliProxyAPI with Codex; Gemini 3.1 Pro, Claude Opus 4.8, and Claude Sonnet 5 are accessed through their respective vendor APIs.

Sampling configuration. All calls use each provider’s default sampling parameters; we set no temperature, top- p , or seed. GPT calls use medium reasoning effort, except for latency-sensitive modules, which use low; Claude and Gemini calls use their default reasoning effort. Because these models are sampled rather than run deterministically, repeated runs need not reproduce identical programs. Randomness in our pipeline therefore originates in provider-side sampling rather than in any seed we control, which is why we report the configuration above in place of random seeds. To keep comparisons paired, one goal image is generated per scene and shared across all compared variants (§5.3).

F.2 Scene Parsing and Geometry

The scene is parsed once, before any reasoning about arrangement. The stage must deliver a persistent inventory: every tabletop object as a named entity carrying a mask, an observed partial point cloud in the table frame, a complete mesh at metric scale, and a current 6-DoF pose. Everything downstream refers to these entities — they are the `object_id` values the program is written over, the meshes the layout render draws and registration aligns, and the two geometric substrates that refinement and execution planning reason with.

Open-vocabulary segmentation. One routine segments a tabletop image into a table mask and a set of item masks, and it is deliberately limited to that: it performs no matching, no depth estimation, no reconstruction, and no pose reasoning.

The same routine is applied to the real observation here and to the generated goal image in Appendix C.1.

A VLM first proposes English segmentation prompts from the image itself, seeded with the scene’s object list and asked for synonyms and more specific variants, since an open-vocabulary segmenter responds better to “milk carton” than to “container”. The proposals are merged with the seed and filtered against a stop list that removes both uselessly generic nouns and the parts of the scene that are not tabletop objects — table, desk, background, shadow, robot, gripper, hand. At most 32 prompts survive, and the generic prompt `object` is appended afterwards, so the catch-all enters exactly once, under our control, rather than crowding out specific prompts during expansion. SAM 3 (Carion et al. 2025) then segments the image against that vocabulary.

Three cleanup passes follow, in order. An over-segmentation merge shows the labeled overlay to a VLM and asks which instance ids belong to one physical object, which repairs the common failure of a mug and its handle, or a box and its lid, arriving as separate instances. An overlap pass then merges pairs whose masks coincide by at least 70% of the smaller mask, iterating until no such pair remains, which removes duplicates left by two prompts naming the same thing. Finally a VLM filter drops instances clearly not on the tabletop. That last pass is conservative by construction: a single candidate, a model error, or an unparseable reply all remove nothing, since dropping a real object is much worse here than keeping a spurious one — an object absent from the inventory can never be arranged, whereas a spurious one is constrained by the program and usually discarded downstream.

Observed geometry. The calibrated depth map is back-projected with the camera intrinsics and extrinsics into the world frame, and each entity keeps the points falling inside its mask. This partial cloud is what the object was actually seen to be. It is incomplete — a mug is observed from one side only — but it is not a guess, which is why the collision test of Appendix D and the dependency tests of Appendix E consult it alongside the reconstruction rather than trusting either alone.

Completion. Each entity’s masked crop and partial cloud are completed by RecGen (Zadaianchuk et al. 2026) into a closed mesh, returned already at metric scale together with the object’s current 6-DoF pose in the world frame. Meshes are simplified to at most 12,000 faces, which bounds the cost of the render-and-compare search of Appendix C.2 and of the per-object rasterization that refinement and planning perform at every step. No CAD model, category prior, or object-specific training is used at any point: the mesh a downstream stage manipulates is reconstructed from the single observation of that scene.

Segmentation on the simulated benchmark. On the 1,000 simulated scenes of §5.3 the item masks for the *messy observation* are read from the simulator’s instance buffer instead of being predicted, and the pipeline continues unchanged from that point. This fixes one inventory for our system and for every ablation and the oracle, so a difference

between two rows of Table 3 cannot be an artifact of one run happening to segment the scene differently. The generated goal image has no ground truth by construction and is therefore always segmented by the predicted path described above, in every experiment; the real-robot trials of §5.6 use the predicted path on both sides.

F.3 Robot Execution and Repair

The program specifies target poses, and because it survives execution it also specifies what to check afterwards. Two executors implement this: the simulated one used for the benchmark and the real arm of §5.6. They share the move order, the goal transform, and the constraint families they grade; they differ in what a correction physically is.

What drives a move. Three inputs are used for exactly one thing each. The planner of Appendix E supplies the order, buffer detours included, so the executor never derives an order of its own. The motion is the goal transform of the object’s *observed partial cloud* rather than of its mesh, since the cloud is the physical thing being moved. The route is decided by the SE(2)/SE(3) verdict of §C.2 together with the program’s `in_holder` declarations.

Execution branches. Each step is routed by a fixed priority. *Insertion* comes first: if the program declares the object `in_holder` something, the step goes to a dedicated skill, and this outranks the SE(2)/SE(3) label because placing an item into a container is a different motion — stand it upright, line it up over the opening, lower it in — rather than a harder rotation. Insertion needs the opening’s rim height and center, which can only be measured where the holder actually is; a holder that is itself a plan step is therefore measured immediately after it is placed, and that measurement is cached rather than refreshed, since re-measuring once an item is in would read the item protruding from the opening as the rim. Every remaining step is planar and is executed directly by top-down transport, translating the object and turning it by the declared yaw. In practice this covers all of them: the SE(3) verdict is rare and almost always lands on an object the program also declares `in_holder`, so the non-planar cases are precisely the ones insertion already handles.

Grasping. On the robot, AnyGrasp (Fang et al. 2023) is run in its ordinary 6-DoF mode, but only its *position* is kept: a strict top-down orientation is synthesized around it whose yaw is chosen geometrically so the fingers straddle the object’s point cloud. Candidates are ranked by AnyGrasp’s own score together with top-down alignment and proximity to the object center, and filtered for gripper–scene collision. The planar goal transform is then composed onto the selected grasp, so the release height is the grasp height plus the object’s vertical displacement, never below the grasp height. Placement is open loop: nothing is re-perceived between the pick and the release.

Re-perception and grading. After the last placement the scene is re-perceived and the program is re-evaluated against what is observed — never against simulator state. In simulation this uses the RGB-D and per-object masks captured during placement; on the robot the scene is re-observed with

the same camera and segmenter, and masks are bound to objects by where each was released. Object positions and orientations are read from the observed masked depth exactly as in any other perception stage, with orientation taken from the min-area 2D box’s long edge.

Four relation families are graded: a shared horizontal line by the spread of its members’ centers along y , a shared vertical line by the spread along x , `align_axis` by the angle between the object’s long edge and the target axis, and `evenly_spaced_from_anchor` by the non-uniformity of its consecutive gaps. Two exclusions apply. Objects stacked on another are not graded, since their placement is dictated by their base and their masks are usually occluded. Objects whose min-area box is nearly square (aspect ratio below 1.15) are not graded for orientation, because their long axis is not meaningful.

Correction. Corrections are not capped by a budget: asking for a line means the whole line should line up, so every offending object is corrected. In simulation a member more than 5 mm off its anchor’s line is nudged back onto it and an orientation error above 5° is corrected in place, and a correction is then discarded if the observed point clouds say it would *newly* overlap a neighbor — objects already touching after placement are not the correction’s doing, so only a newly introduced overlap vetoes a move.

On the robot a correction is a nudge rather than a re-place: the object rises 5 mm and slides sideways, deliberately staying below transport height, because lifting it clear would make the move trivially safe at the cost of the placement accuracy the repair exists to recover. Ordering therefore becomes the whole problem, since the object and the fingers sweep across the table at working height: the nudges are topologically sorted so that none crosses a neighbor that has not yet moved away or lands where one is about to arrive. Two exemptions keep that test honest rather than merely conservative. A pair is never rejected for its starting configuration, which the executor demonstrably achieved, only for becoming worse than it; and a neighbor the move recedes from cannot become a problem.

Effect on the benchmark. Of the 994 simulated scenes with a repair record, 898 (90.3%) received at least one correction, 2,390 objects were corrected in total, and a corrected scene received 2.66 corrections on average. Flagged constraints fall from 1,037 to 419, a 59.6% reduction, and the share of scenes with nothing flagged rises from 34.5% to 66.7%. Restricted to the 651 scenes that had something to fix, 497 (76.3%) improve and 152 are unchanged. Four scenes end worse than they started, two of which had no flagged constraint to begin with; the collision veto keeps a correction from being attempted, but it does not guarantee that every attempted one lands.

F.4 Prompts and Key Parameters

Free parameters. The pipeline exposes few constants of its own, since its reasoning components are prompted rather than trained and its metric components are third-party models used at their defaults. The ones that matter for reproducing our results are the following: all imaging, both observed and

rendered, runs at 1280×720 ; render-and-compare searches the six canonical orientations at a 30° angular resolution; layout refinement targets a 2cm minimum clearance and keeps the layout 3cm inside the table edge; and the post-execution repair stage corrects a member more than 5mm off its declared line, or more than 5° off its declared axis (Appendix F.3).

The clearance is defined on the whole configuration rather than pairwise in isolation: with every object placed at its target pose, we ask that the closest-point distance between any two objects be at least 2cm, evaluated on both the reconstructed meshes and the observed point clouds so that neither representation alone decides feasibility. Because refinement is iterative and adjusts θ while holding the declared relations fixed, this clearance is a best-effort objective rather than a hard constraint: the solver drives the configuration toward it but does not certify it, and a returned layout may leave some pair below the target.

Scale and pose search is not ours to parameterize: we delegate it to RGBTrack (Guo and Yu 2025) and adopt its published defaults without modification, so no search range is set on our side. None of the values above was tuned: we ran no hyperparameter search, and each was fixed before the evaluations reported in this paper.

Listing 2: Organization-plan prompt (Stage 1, §3.2).

```
You are a careful table-arrangement
  ↳ planner. The image shows a messy table
  ↳ with several items on it. Your job is
  ↳ to write a concrete plan for
  ↳ rearranging the items so the table
  ↳ matches what the user is likely about
  ↳ to do.
```

```
USER'S STATED DIRECTION (may be "(not
  ↳ specified)"):
{user_intent}
```

```
REFERENCE STYLE (optional, may be "(none
  ↳ provided)"):
{reference_style_summary}
```

```
Step A -- Determine the WORKING INTENT
  ↳ (one short sentence):
```

- If the user's stated direction is
 - ↳ concrete and actionable (e.g. "dining
 - ↳ setup", "free up workspace", "meeting
 - ↳ setup", "ready to read"), restate it as
 - ↳ the working intent and move on.
 - Otherwise (stated direction is "(not
 - ↳ specified)" or too vague like "tidy up"
 - ↳ / "organize this"), OBSERVE the scene
 - ↳ and INFER a meaningful working intent:
 - * Look at which item types are actually
 - ↳ present on the table and ask: what
 - ↳ activity do these items collectively
 - ↳ suggest the user is about to do RIGHT
 - ↳ NOW?
 - * Examples of plausible inferences:
 - plates / bowls / cutlery / cups +
 - ↳ food items present -> "single-person

- ↳ dining; the user is about to eat at
 - ↳ this table".
 - plates / bowls / cookware plus
 - ↳ condiment bottles / jars / seasoning
 - ↳ containers, with no laid-out place
 - ↳ setting (no cutlery arranged for a
 - ↳ diner, no ready-to-eat food) ->
 - ↳ "kitchen counter cleanup; the user is
 - ↳ clearing and resetting a kitchen work
 - ↳ surface: dishware nested and grouped,
 - ↳ condiments gathered together, the
 - ↳ working area left clear".
 - laptop / notebook / pens / papers
 - ↳ present -> "workspace / note-taking;
 - ↳ the user is about to work or take notes
 - ↳ at this table".
 - books / reading lamp / mug present
 - ↳ -> "reading nook; the user is about to
 - ↳ read at this table".
 - mostly drinkware (bottles / cans /
 - ↳ glasses) and snacks -> "casual drinks;
 - ↳ the user is about to have a drink at
 - ↳ this table".
 - mixed clutter with no dominant
 - ↳ activity signal -> "general tidy: group
 - ↳ same-class items, no special zoning,
 - ↳ just clean and aligned".
 - * Pick the SINGLE most plausible intent
 - ↳ given the dominant items; do not list
 - ↳ multiple options. Do NOT default to
 - ↳ dining unless dining items are actually
 - ↳ present.
 - If the user's stated direction is
 - ↳ concrete but the scene clearly cannot
 - ↳ support it (e.g. user says "dining
 - ↳ setup" but the table only has a laptop
 - ↳ and no dining items), still follow the
 - ↳ user's stated direction -- the user's
 - ↳ direction always wins over what the
 - ↳ scene suggests.

```
Step B -- Plan the layout based on the
  ↳ working intent:
```

1. Identify every visible item type on the
 - ↳ table.
2. Decide each item's target position
 - ↳ based on the working intent. Examples
 - ↳ to ground your reasoning:
 - * "free up workspace" / "clear desk" ->
 - ↳ push items toward the BACK and SIDES of
 - ↳ the table; leave the FRONT-CENTER clear.
 - * "single-person dining" / "ready to
 - ↳ eat" -> plate or bowl roughly
 - ↳ center-front (toward the diner); fork
 - ↳ left of the plate; knife right of the
 - ↳ plate (blade facing the plate); spoon
 - ↳ right of the knife; drinkware (cup /
 - ↳ glass / can / bottle / pitcher) at the
 - ↳ upper-right; condiments / boxes /
 - ↳ accessories grouped to the upper or
 - ↳ side area, not in the eating zone;
 - ↳ nothing on top of the plate.
 - * "meeting setup" / "ready to take
 - ↳ notes" -> laptop or notebook
 - ↳ front-center; pens and small items at

```

    ↪ one side; cables / chargers grouped
    ↪ away.
    * "reading nook" -> book front-center,
    ↪ mug at upper-right, other items pushed
    ↪ to the sides.
    * "general tidy" -> group same-class
    ↪ items, align edges with the table, no
    ↪ special zoning.
3. General common-sense rules:
  - Items should sit horizontal/vertical
  ↪ relative to the table edges; no
  ↪ arbitrary tilt.
  - Plausible physics: items rest on the
  ↪ table; no unnatural stacking
  ↪ (books/boxes are fine to stack flat).
  - Don't place items on top of plates or
  ↪ bowls when the user is about to use
  ↪ them.
  - Duplicates of the same item type can
  ↪ sit next to each other if no obvious
  ↪ single-slot exists.

```

The working intent is the HIGHEST priority
 ↪ -- if a common-sense rule conflicts
 ↪ with the working intent, follow the
 ↪ intent.

Output format (exactly these two labeled
 ↪ sections, no extra prose before or
 ↪ after):
 [WORKING_INTENT]
 <one short sentence stating the working
 ↪ intent. If you inferred it from the
 ↪ scene, briefly note the items that
 ↪ drove the inference, e.g. "Inferred
 ↪ dining setup (plate + fork + cup
 ↪ present)." If the user stated it, just
 ↪ restate it.>

[PLAN]
 <3-5 sentences in plain English describing
 ↪ the goal layout. Mention each visible
 ↪ item type and its target position.
 ↪ Describe orientation only when it
 ↪ actually matters, and AVOID the bare
 ↪ words "horizontal" / "vertical" -- they
 ↪ are ambiguous to the downstream image
 ↪ editor (horizontal in the image frame?
 ↪ parallel to which table edge?) and tend
 ↪ to flip elongated items like cutlery
 ↪ into the wrong pose. When orientation
 ↪ does not matter, just give the position
 ↪ and let the editor place the item
 ↪ naturally. When it does matter, anchor
 ↪ it to something concrete instead, e.g.
 ↪ "handle toward the diner", "long axis
 ↪ pointing away from the diner", "spine
 ↪ parallel to the left table edge". If
 ↪ you nonetheless use "horizontal" or
 ↪ "vertical", you MUST qualify it with
 ↪ the reference frame and the axis
 ↪ (relative to what, and along which
 ↪ direction). Do not include item counts,
 ↪ editing-process commentary, or numbered
 ↪ steps -- just the spatial plan.>

Braces in the listings below mark values substituted at call time: the user's instruction, the object inventory, the mask ids offered for assignment, and so on.

Listing 3: Goal-image editing prompt (Stage 1, §3.2). Filled with the plan produced by Listing 2.

```

Edit the image to produce this goal
  ↪ arrangement on the same table:

{plan}

Constraints:
- Same number of objects -- do not add or
  ↪ remove anything.
- Same camera viewpoint, same lighting,
  ↪ same table.
- Items rest flat on the table (gravity)
  ↪ and align horizontal/vertical with the
  ↪ table edges.

```

Listing 4: Direct-tidy prompt, the *w/o Plan* arm of the ablation in §5.4. It carries the same intent-inference list, per-intent positioning examples and commonsense rules as Listing 2, so the only variable under test is where the reasoning happens.

```

Edit this image of a messy table so the
  ↪ SAME table becomes a tidy, purposeful
  ↪ arrangement -- how it should look right
  ↪ before the user does what they are
  ↪ about to
do here. Keep it photorealistic and keep
  ↪ every object already on the table.

THE USER'S STATED DIRECTION (may be "(not
  ↪ specified)"):
{user_direction}

First decide the WORKING INTENT for this
  ↪ table:
- If the direction is concrete (e.g.
  ↪ "dining", "free up workspace", "meeting
  ↪ setup",
  ↪ "ready to read"), use it directly.
- If it is missing or vague ("tidy up",
  ↪ "(not specified)"), infer the single
  ↪ most
  ↪ plausible activity from the items
  ↪ actually present:
  ↪ plates/bowls/cutlery/cups + food
  ↪ -> single-person dining;
  ↪ laptop/notebook/pens/papers ->
  ↪ workspace / note-taking;
  ↪ books/lamp/mug -> reading; mostly
  ↪ drinkware + snacks -> casual drinks;
  ↪ mixed clutter
  ↪ with no dominant signal -> just group
  ↪ and align, no special zoning. Do NOT
  ↪ default to
  ↪ dining unless dining items are actually
  ↪ present.

```

Then arrange every item to serve that

```

    ↪ intent:
- "free up workspace" -> push items to the
  ↪ BACK and SIDES; leave the FRONT-CENTER
  ↪ clear.
- "single-person dining / ready to eat" ->
  ↪ plate or bowl center-front toward the
  ↪ diner;
  fork left of the plate; knife right
  ↪ (blade toward the plate); spoon right
  ↪ of the knife;
  drinkware upper-right; condiments /
  ↪ boxes / accessories grouped to the
  ↪ upper or side
  area, never in the eating zone; nothing
  ↪ on top of the plate.
- "meeting / note-taking" -> laptop or
  ↪ notebook front-center; pens and small
  ↪ items to one
  side; cables / chargers grouped away.
- "reading" -> book front-center, mug
  ↪ upper-right, everything else to the
  ↪ sides.
- "general tidy" -> group same-class items
  ↪ and align them to the table edges.

```

Common-sense rules:

- Every item rests flat on the table under
 - ↪ gravity; no floating, no arbitrary
 - ↪ tilt, no
 - unnatural stacking (flat books/boxes may
 - ↪ stack).
- Orient elongated items by a concrete
 - ↪ reference (cutlery handle toward the
 - ↪ diner, book
 - spine parallel to a table edge), not at
 - ↪ random angles.
- Put duplicates of the same item type
 - ↪ side by side.

The working intent has the highest

- ↪ priority: if a common-sense rule
- ↪ conflicts with it,

 follow the intent.

Hard constraints (must hold):

- Exactly the same objects as the input --
 - ↪ do not add, remove, or substitute
 - ↪ anything.
- Same camera viewpoint, same lighting,
 - ↪ same table surface and background.
- Items sit flat and align to the table
 - ↪ edges.

Listing 5: Arrangement-program synthesis prompt (Stage 2, §3.3). The grammar it teaches is the language of Appendix B.

VLM Layout Constraint Guide

Your task: from an image of a tidied goal

- ↪ tabletop, infer a set of executable
- layout constraints (a DSL) describing
- ↪ where each object should be placed on
- ↪ the

 table and with what orientation.

This is not an annotation task. Every

- ↪ geometric relation must be bound to an

 `object\_id` from the input object list.

Inputs

1. ****Current scene image****: the real messy
 - ↪ table. Every object to be arranged
 - carries a segmentation mask and a
 - ↪ numeric label.
2. ****Goal image****: the tidied table. ****No**
 - ↪ segmentation, no numeric labels.** This
 - is the geometric target you must parse.
3. ****Object dictionary****: the key is the
 - ↪ numeric label in the current scene
 - ↪ image,
 - the value is the `object\_id` you must
 - ↪ use when writing constraints. For
 - ↪ example:

```

...json
{
  "1": "dinner_plate",
  "2": "fork",
  "3": "knife",
  "4": "cup"
}
...

```

The numeric labels serve only to locate

- ↪ objects in the current scene image.

****Only `object\_id` values may appear in**

- ↪ the output.**

The object dictionary is the authoritative

- ↪ list: every object in it must be fully
- defined in the output, and you may not
- ↪ invent objects outside it. Anything
- ↪ visible

 in the goal image but absent from the

- ↪ dictionary (background, tablecloth,
- decorations) is ignored.

If an object in the dictionary is not

- ↪ clearly visible or is occluded in the
- ↪ goal

 image, you must still emit constraints for

- ↪ it -- infer a reasonable position from
- the structure already present in the image
- ↪ and from commonsense about the scene.

Coordinate frame and orientation

The image is an oblique top-down view from

- ↪ a person seated at the table, looking
- away from themselves: the long edge of the
- ↪ table is parallel to the top and bottom
- edges of the image, and moving up the
- ↪ image corresponds to moving away from
- ↪ the

 viewer.

In this view the table-normalized

- ↪ coordinate frame is:

- `x`: positive to the right, negative to the left. The table centre is `0`, and the left and right edges are near `-1` / $\rightarrow$ `1`.
- `y`: **positive** away from the viewer (up $\rightarrow$ in the image), **negative** toward the viewer (down in the image). The table centre is `0`.
- `yaw`: rotation about the table normal $\rightarrow$ (perpendicular to the table, opposite gravity). This is the object's only $\rightarrow$ orientation degree of freedom.

Coordinates are table-normalized
 $\rightarrow$ quantities, not physical lengths. You
 $\rightarrow$ do not need
 millimetre precision and you do not need
 $\rightarrow$ to estimate specific angles -- the goal
 $\rightarrow$ is
 to build stable, executable, structurally
 $\rightarrow$ correct geometric relations.

Orientation may only be defined through
 $\rightarrow$ `align\_axis`, whose values are:

- `"vertical"`: the object's principal
 $\rightarrow$ axis is aligned with the **yy** axis
 (pointing away from / toward the viewer).
- `"horizontal"`: the object's principal
 $\rightarrow$ axis is aligned with the **xx** axis
 (left-right).
- `"custom"`: a clearly visible oblique
 $\rightarrow$ placement, aligned with neither the x
 $\rightarrow$ nor
 the y axis. The specific angle is filled
 $\rightarrow$ in by downstream pose estimation.
- `"any"`: orientation is meaningless or
 $\rightarrow$ unimportant (rotationally symmetric
 objects such as plates, bowls, cups).

Output format

Output a single JSON object. Do not output
 $\rightarrow$ any explanatory text, Markdown, or code
 fences. The top level contains only
 $\rightarrow$ `constraints`:

```
```json
{
 "constraints": [
 {"type": "table_xy", "object":
 $\rightarrow$ "dinner_plate", "x": 0.0, "y": 0.0},
 {"type": "align_axis", "object":
 $\rightarrow$ "dinner_plate", "axis": "any"}
]
}
```
```

Three hard rules

The DSL is parsed in order from top to
 $\rightarrow$ bottom. Each constraint **writes**
 $\rightarrow$ certain
 fields, and some also **read** an anchor's
 $\rightarrow$ fields.

Rule 1 -- Completeness. For every
 $\rightarrow$ object in the dictionary, all three
 $\rightarrow$ fields
 `x`, `y` and `yaw` must be written.

Rule 2 -- Single assignment. Each
 $\rightarrow$ field may be written **exactly once**.
 Writing the same field twice immediately
 $\rightarrow$ triggers an over-define error and the
 whole DSL fails.

Rule 3 -- Dependency order. An anchor
 $\rightarrow$ field that a constraint reads must
 already have been written before it.

Before emitting, self-check object by
 $\rightarrow$ object: how many times is x written? y?
 $\rightarrow$ yaw?
 Each must be exactly 1.

Placement levels

Objects belong to different placement
 $\rightarrow$ levels: those resting directly on the
 $\rightarrow$ table
 are one level, and those stacked on
 $\rightarrow$ another object (`on\_top\_of`) or
 $\rightarrow$ inserted into a
 holder (`in\_holder`) are another.

`evenly\_spaced\_from\_anchor` **may** only act
 $\rightarrow$ on objects at the same level. Bread
 resting on a plate cannot participate in
 $\rightarrow$ the same even-space group as the cutlery
 and plates on the table.

`in\_same\_vertical\_line` and
 $\rightarrow$ `in\_same\_horizontal\_line` carry no such
 $\rightarrow$ restriction and
 may be used across levels.

Constraint reference

Only the types and field names below are
 $\rightarrow$ permitted. Each is annotated with the
 fields it reads and writes.

table_x / table_y / table_xy

Absolute placement anchored to the table,
 $\rightarrow$ independent of any other object.

```
```json
{"type": "table_x", "object": "cup", "x":
 $\rightarrow$ 0.3}
{"type": "table_y", "object": "cup", "y":
 $\rightarrow$ 0.2}
{"type": "table_xy", "object": "plate",
 $\rightarrow$ "x": -0.2, "y": 0.1}
```
```

- Writes: `table\_x` $\rightarrow$ `object.x`;
 $\rightarrow$ `table\_y` $\rightarrow$ `object.y`; `table\_xy` $\rightarrow$
 $\rightarrow$ `object.x` and `object.y`.

- Reads: nothing.

Use for reference objects, or when one
↳ axis cannot be characterised by
↳ relations to
other objects and can only be read off as
↳ an absolute position.

align_axis

```
```json
{"type": "align_axis", "object": "fork",
 ↳ "axis": "vertical"}
```
```

- Writes: `object.yaw``.
- Reads: nothing.

in_same_vertical_line

Make several objects share one `x`` (same
↳ column, centres vertically aligned).

```
```json
{"type": "in_same_vertical_line",
 ↳ "objects": ["monitor", "keyboard"],
 ↳ "anchor": "monitor"}
```
```

- Writes: `x`` of every object in `objects``
↳ except `anchor``.
- Reads: `anchor.x``.
- `anchor`` must be inside `objects``. May
↳ cross levels.

in_same_horizontal_line

Make several objects share one `y`` (same
↳ row, centres horizontally aligned).

```
```json
{"type": "in_same_horizontal_line",
 ↳ "objects": ["keyboard", "mouse"],
 ↳ "anchor": "keyboard"}
```
```

- Writes: `y`` of every object in `objects``
↳ except `anchor``.
- Reads: `anchor.y``.
- `anchor`` must be inside `objects``. May
↳ cross levels.

evenly_spaced_from_anchor

Arrange several objects ****at the same**
↳ **level**** at equal spacing along a single
↳ axis.

```
```json
{
 "type": "evenly_spaced_from_anchor",
 "objects": ["fork", "plate", "knife"],
 "anchor": "plate",
 "axis": "x",
 "mode": "footprint",
```

```
 "order": ["fork", "plate", "knife"],
 "spacing": 0.03
}
```

- Writes: the coordinate along `axis`` of  
↳ every object in `objects`` except  
↳ `anchor``.  
- Reads: the `anchor``'s coordinate along  
↳ `axis``.  
- `axis``: `"x"` or `"y"`.  
- `mode``: `"obj_center"` for  
↳ centre-to-centre spacing, `"footprint"`  
↳ for outline  
edge-to-edge spacing (better for  
↳ avoiding collisions, and the default  
↳ choice).  
- `order``: the objects in increasing order  
↳ along that axis.  
- `spacing``: the spacing in  
↳ table-normalized units.  
- Defines one axis only; the other axis is  
↳ usually supplied by a same-line  
constraint or by `table_x`` / `table_y``.

Strategy: first pin down, on that axis,  
↳ the most meaningful reference object in  
↳ the  
group (usually the scene's reference  
↳ object, or the one adjacent to it)  
↳ using other  
constraints, then expand the whole group  
↳ from it as anchor.

```
x_offset_from / y_offset_from /
 ↳ xy_offset_from
```

Single-axis or two-axis relative offsets.

```
```json
{"type": "x_offset_from", "object":
  ↳ "knife", "anchor": "plate", "dx": 0.25}
{"type": "y_offset_from", "object":
  ↳ "cup", "anchor": "plate", "dy": 0.2}
{"type": "xy_offset_from", "object":
  ↳ "cup", "anchor": "plate", "dx": 0.25,
  ↳ "dy": 0.2}
```
```

Semantics: `object.x = anchor.x + dx``,  
↳ `object.y = anchor.y + dy``.  
`dx > 0`` means the object is to the right  
↳ of the anchor; `dy > 0`` means the object  
is on the side further from the viewer.

- Writes: the corresponding `object.x``  
↳ and/or `object.y``.  
- Reads: the corresponding `anchor.x``  
↳ and/or `anchor.y``.

When only one axis needs defining, use the  
↳ single-axis form; do not casually use  
`xy_offset_from`` and write the other axis  
↳ as well -- that will collide with a  
same-line constraint and trigger an

↳ over-define.

### on\_top\_of

A plain stacking/support relation:  
 ↳ `object` lies flat on top of `anchor`.  
 ↳ A bowl on a plate, a spoon in a bowl, bread in a bread basket all express their vertical relation this way.

```
```json
{"type": "on_top_of", "object": "bread",
  ↳ "anchor": "bread_basket"}
```
```

- Writes: **no fields**.  
 - Reads: nothing.

`on\_top\_of` **only** declares the level  
 ↳ relation; it does not define `x` / `y` / `yaw`. Those three fields must still be  
 ↳ written by other constraints. For concentric stacking, use `xy\_offset\_from`  
 ↳ with `dx: 0, dy: 0`.

### in\_holder

An elongated object placed **standing**  
 ↳ upright into a holder that can  
 ↳ maintain that pose (a pen in a pen cup, a spoon in a mug). This is not a general  
 "object inside a container" relation -- an  
 ↳ object lying flat as usual uses `on\_top\_of`.

```
```json
{"type": "in_holder", "object": "pen",
  ↳ "holder": "pen_holder"}
```
```

- Writes: `object.x`, `object.y`,  
 ↳ `object.yaw` (**all three at once**).  
 - Reads: `holder.x`, `holder.y`.

An object using `in\_holder` therefore  
 ↳ **does not need and is not allowed** to  
 ↳ be defined by any other constraint, including  
 ↳ `align\_axis`.

## Generation procedure

1. **Anchor the reference.** Find one or  
 ↳ two salient, reliably positioned large objects in the goal image (plate, keyboard, tray, cutting board) and give  
 ↳ them absolute positions with `table\_xy`.
2. **Expand from near to far.** Around the  
 ↳ reference objects, define the objects that have a clear geometric relation to  
 ↳ them one by one; then use those newly defined objects as anchors and

↳ keep expanding outward. Follow the priority below when choosing  
 ↳ constraints.

3. **Fill in orientation.** Except for  
 ↳ `in\_holder` objects, add one  
 ↳ `align\_axis` for every object. Use `"any"` when  
 ↳ uncertain or rotationally symmetric.
4. **Self-check.** Confirm object by  
 ↳ object that x / y / yaw are each written exactly once, and that every anchor  
 ↳ field read has been written before it is read.

### Constraint selection priority

- If several objects' centres are clearly  
 ↳ aligned vertically in the image ->  
 ↳ `in\_same\_vertical\_line` to define the  
 ↳ shared `x`.
- If several objects' centres clearly form  
 ↳ a left-right row ->  
 ↳ `in\_same\_horizontal\_line` to define the  
 ↳ shared `y`.
- On top of a shared line, if the spacing  
 ↳ in that row/column is clearly  
 ↳ approximately uniform ->  
 ↳ `evenly\_spaced\_from\_anchor` for the  
 ↳ other axis. If the spacing is not uniform, define the other  
 ↳ axis one object at a time with  
 ↳ `x\_offset\_from` / `y\_offset\_from`.
- When two objects have an independent  
 ↳ two-dimensional relative position, use  
 ↳ `xy\_offset\_from`.
- For an object with no reliable geometric  
 ↳ relation to anything in the image, use  
 ↳ `table\_xy` to define its absolute  
 ↳ position in the goal image directly.
- Express stacking with `on\_top\_of` to  
 ↳ declare the level, then add the offset  
 ↳ and  
 ↳ `align\_axis`; express upright insertion  
 ↳ with `in\_holder`.

Never write a row of clearly collinear  
 ↳ objects as a chain of `xy\_offset\_from`  
 ↳ --  
 that discards the "same row" and "evenly  
 ↳ spaced" structural priors, and the error  
 accumulates along the chain.

Relations must be useful for execution and  
 ↳ supported either by the goal image or by  
 solid commonsense. Do not pile on  
 ↳ relations for the sake of structural  
 ↳ complexity.

[Two worked examples follow in the prompt:  
 ↳ a same-line-plus-even-spacing place  
 setting, and a desk scene exercising  
 ↳ cross-level stacking, insertion, and  
 single-axis constraints. Appendix B walks  
 ↳ through an equivalent program.]

Now infer layout constraints using TWO  
 ↳ attached images.

Image 1 is the real current scene  
 ↳ segmentation. It labels the REAL input  
 ↳ objects  
 with numeric ids. Use it to establish  
 ↳ which object\_id each number refers to  
 ↳ and  
 what each real object looks like. Do not  
 ↳ copy the current scene's messy positions  
 as the target layout.

Image 2 is a tidy-scene reference image  
 ↳ without segmentation labels. Use it as  
 ↳ the  
 target arrangement style and layout prior.  
 ↳ It may miss some real objects, merge  
 multiple real objects into one visible  
 ↳ object, or contain extra objects that  
 ↳ are  
 not in the real inventory. The output must  
 ↳ always arrange the REAL objects from  
 Image 1 / the map below.

Important reasoning rules:

- If the tidy reference has one object but  
 ↳ the real scene has multiple similar  
 objects, arrange all real similar  
 ↳ objects as an analogous group,  
 ↳ preferably using  
 same-line or evenly-spaced relations.
- If the tidy reference has extra objects  
 ↳ not present in the real inventory,  
 ↳ ignore  
 those extra objects.
- If the tidy reference is missing a real  
 ↳ object, infer a reasonable position from  
 similar objects and common table layout.
- Always output constraints for object\_id  
 ↳ values, never numeric ids.
- Prefer structure priors such as  
 ↳ in\_same\_horizontal\_line,  
 ↳ in\_same\_vertical\_line,  
 evenly\_spaced\_from\_anchor, on\_top\_of,  
 ↳ and in\_holder only for upright insertion  
 into a holder when visible or inferable.

Numeric-id to object\_id map for the real  
 ↳ current scene:  
 {object\_id\_map}

Return only the final JSON object with the  
 ↳ top-level key constraints.

**Listing 6: Reference-mask assignment prompt (Appendix C.1).**

You will receive two images.

Image 1 is a planned top-down tabletop  
 ↳ layout. The object labels in Image 1 are  
 real object ids. Use the layout image  
 ↳ together with the DSL constraints below

↳ to  
 understand each planned object's relative  
 ↳ position, grouping, stacking, and  
 orientation.

Image 2 is the goal/reference image  
 ↳ segmentation. Its visible masks are  
 ↳ labeled  
 with numeric ids. These numeric ids are  
 ↳ only reference mask labels for this  
 ↳ task.

Task: assign every object id from Image 1  
 ↳ to one numeric goal mask id from Image  
 ↳ 2.

Multiple objects may point to the same  
 ↳ numeric goal mask because the goal  
 ↳ image can  
 miss objects or merge several planned  
 ↳ objects into one visible region.

If Image 2 contains more masks than  
 ↳ needed, do not overthink every extra  
 ↳ object. Do  
 reasonable thinking, but do not overthink.  
 ↳ Choose the most reasonable corresponding  
 masks for the listed Image 1 objects so  
 ↳ the overall tabletop layout matches as  
 ↳ well  
 as possible. Unused Image 2 masks are  
 ↳ allowed.

Return only a JSON object mapping each  
 ↳ object id to one numeric mask id  
 ↳ string. Do  
 not include Markdown, code fences,  
 ↳ explanations, confidence scores, or  
 ↳ extra keys.

Object ids to assign:  
 {object\_ids}

Allowed numeric goal mask ids:  
 {goal\_mask\_labels}

DSL constraints for Image 1:  
 {layout\_dsl}

Output schema example:  
 {  
 "cup\_1": "3",  
 "plate\_1": "1"  
 }

**Listing 7: SE(2)/SE(3) classification prompt (Appendix C.2).**  
 The verdict gates each object's pose-candidate set; an object  
 with no recoverable label defaults to SE(3).

You are given a before-and-after image  
 ↳ pair of the same tabletop scene.  
 ↳ Objects are  
 marked with matching labels across the two  
 ↳ images. A label is either a plain number

(e.g. 3) marking a single object, or a  
 ↳ number followed by a letter (e.g. 1A,  
 ↳ 1B)  
 used when several distinct physical items  
 ↳ in the LEFT image all correspond to the  
 SAME single labeled region (label 1) in  
 ↳ the right image. Each lettered label  
 (1A, 1B, ...) is a SEPARATE physical  
 ↳ object and MUST be classified on its  
 ↳ own.  
 Classify each label's transformation from  
 ↳ the left image to the right image as  
 either SE2 or SE3. Definitions:  
 - SE2: The object only changes tabletop  
 ↳ translation and yaw, including possible  
 height changes caused by stacking or  
 ↳ support changes. The object keeps the  
 ↳ same  
 gravity-consistent orientation.  
 - SE3: The object undergoes a true 3D pose  
 ↳ change, such as tilting, rolling,  
 pitching, flipping, lying down, standing  
 ↳ up, being inserted into a container,  
 leaning against another object, or  
 ↳ changing support mode.  
 The left image is a real photographic  
 ↳ capture of a STABLE, current tabletop  
 ↳ scene:  
 every object in it is at rest in a  
 ↳ gravity-stable pose. The right image is  
 ↳ its  
 organization/target result. Use the table  
 ↳ as the plane anchor to decide whether an  
 SE3 change actually happened or not. Note  
 ↳ that the camera view is not necessarily  
 level or top-down: the camera may be  
 ↳ tilted, so an object that looks tilted  
 ↳ in the  
 image may not actually be tilted relative  
 ↳ to the table plane. Because the left  
 ↳ scene  
 is physically stable, an object that  
 ↳ appears tilted there is almost  
 ↳ certainly just  
 standing upright under a tilted camera,  
 ↳ not truly leaning.  
 Important:  
 - The right image may contain hallucinated  
 ↳ appearance or shape changes.  
 - Do not classify an object as SE3 only  
 ↳ because its texture, color, label,  
 ↳ category  
 detail, handle, or visual appearance  
 ↳ changed.  
 - If the physical pose is still  
 ↳ upright/on-table and only position/yaw  
 ↳ changed,  
 classify it as SE2.  
 - When several lettered labels share a  
 ↳ base number (1A, 1B, ...), judge each  
 lettered item independently against that  
 ↳ same right-image region; they can  
 ↳ receive  
 different verdicts.  
 - Only SE3 objects require reasons.

- SE2 objects must not have reasons.  
 - Every label must appear exactly once,  
 ↳ either in "SE2" or in "SE3", written  
 ↳ exactly  
 as drawn (keep any letter suffix).  
 Return only valid JSON. Do not include  
 ↳ markdown, comments, explanations, or  
 ↳ extra  
 text. Use exactly this schema:  
 { "SE2": ["label"], "SE3": [ { "id":  
 ↳ "label", "reason": "brief reason" } ] }

#### Listing 8: Over-segmentation merge prompt (Appendix F.2).

You are given one segmentation  
 ↳ visualization image of the current desk  
 ↳ scene.

Each visible instance already has a text  
 ↳ label drawn on the image.  
 Your task is ONLY to decide whether some  
 ↳ instance masks should be merged because  
 they are over-segmented parts of the same  
 ↳ real object.

Critical decision rule:  
 - Do NOT merge based on name similarity,  
 ↳ text similarity, or id naming patterns.  
 - Merge ONLY when you can visually confirm  
 ↳ they are actually one physical object  
 split into multiple mask parts.  
 - Also merge when multiple segmented parts  
 ↳ together form one macro-level  
 carried/attached object set that should  
 ↳ be treated as a single object in  
 manipulation, for example keys attached  
 ↳ to a wallet or keychain attached to  
 another item.  
 - If they are separate physical objects  
 ↳ (even with same/similar names), do NOT  
 merge.  
 - A mask must point to a single movable  
 ↳ item, e.g. if multiple building block  
 stacking on together, you should not  
 ↳ merge them into one, unless two masks  
 ↳ are  
 covering the same building block.

You must follow these rules strictly:  
 1. If no merge is needed, output exactly:  
 []  
 2. If merges are needed, output ONLY a  
 ↳ Python-style list of lists of instance  
 ↳ ids,  
 for example:  
 [['tissue\_box\_1', 'tissue\_box\_2'],  
 ↳ ['id\_1', 'id\_2', 'id\_3']]  
 3. Each inner list means these ids should  
 ↳ be merged into one object.  
 4. Do not output explanations.  
 5. Do not output markdown code fences.  
 6. Only use ids from the candidate list  
 ↳ below.  
 7. Do not create singleton groups (group

↳size must be >= 2).  
 8. If uncertain, prefer conservative  
 ↳behavior (output [] or fewer merges).  
 9. Never merge only because labels/ids  
 ↳look related.

Candidate instance ids:  
 {instance\_id\_list}

**Listing 9: Off-table filter prompt (Appendix F.2). The third cleanup pass; conservative by construction, since dropping a real object removes it from the inventory for good.**

You are given one segmentation  
 ↳visualization image of the current desk  
 ↳scene.

Each visible instance already has a text  
 ↳label drawn on the image.  
 Your task is ONLY to identify instance ids  
 ↳that should be removed because they are  
 clearly outside the tabletop work area or  
 ↳clearly not on the desk surface.

Critical decision rule:  
 - Remove ONLY when you can visually  
 ↳confirm the instance is not supported  
 ↳by the  
 tabletop, is fully off the desk, or  
 ↳clearly belongs to background/outer  
 ↳scene  
 clutter rather than a tabletop object.  
 - Do NOT remove an instance merely because  
 ↳it is not the main target object.  
 - Do NOT remove an instance if it still  
 ↳appears to be a normal tabletop object  
 ↳on  
 the desk.  
 - Do NOT remove an object that is  
 ↳partially beyond the table edge if it  
 ↳still  
 appears stably resting on the tabletop.  
 - If uncertain, keep it.

You must follow these rules strictly:  
 1. If no removal is needed, output exactly:  
 []  
 2. If removals are needed, output ONLY a  
 ↳Python-style list of instance ids, for  
 example:  
 ['box\_2', 'cup\_1']  
 3. Do not output explanations.  
 4. Do not output markdown code fences.  
 5. Only use ids from the candidate list  
 ↳below.  
 6. If uncertain, prefer conservative  
 ↳behavior (output [] or fewer removals).

Candidate instance ids:  
 {instance\_id\_list}

**Listing 10: Current-support detection prompt (Appendix E).**

You are given an image of a tabletop scene  
 ↳with object labels.

The scene contains exactly these objects  
 ↳-- use these EXACT ids and nothing else:  
 {object\_list}

Extract only direct physical support  
 ↳relations between objects.

A support relation exists only when one  
 ↳object is physically placed on top of  
 another object, and the lower object  
 ↳supports the upper object.

Output format:  
 [  
     (<upper\_item>, <supporting\_item>),  
     ...  
 ]

Rules:  
 1. Use ONLY the exact object ids listed  
 ↳above, copied verbatim -- same spelling,  
 same underscores, same case. Do NOT  
 ↳transcribe labels off the image and do  
 NOT invent names; every id you output  
 ↳must be one of the ids listed above.  
 2. Only output object-to-object support  
 ↳relations.  
 3. Do NOT include objects that are  
 ↳directly supported by the table.  
 4. Do NOT include the table as a  
 ↳supporting item.  
 5. Output only the direct supporting  
 ↳object, not indirect supports.  
 For example, if cup is on plate and  
 ↳plate is on table, output only ("cup",  
 ↳"plate").  
 6. If an object merely touches another  
 ↳object but is not vertically supported  
 ↳by it,  
 do not output that relation.  
 7. Return a valid Python list of tuples.  
 8. Do not output any explanation or extra  
 ↳text.

**Listing 11: Pairwise judging prompt (§5.1). Presentation order is swapped on the second pass.**

You are an impartial judge comparing two  
 ↳photographs of a tabletop arrangement.  
 The FIRST image is "Image A"; the SECOND  
 ↳image is "Image B".

The two images may differ slightly in  
 ↳rendering style or object count. Do NOT  
 let visual style or one extra item decide  
 ↳the judgement. Judge the arrangement  
 itself: focus on placement, layout, and  
 ↳whether the scene satisfies the user's  
 request.

User request:

```

"{user_prompt}"

Answer the following two questions
 ↪ independently of each other.

Q1 -- Tidiness:
Which arrangement is more tidy and neatly
 ↪ organized? Consider whether items are
aligned, evenly spaced, upright, and free
 ↪ of messy clutter or careless overlap.

Q2 -- User request:
Which arrangement better satisfies the
 ↪ user's request above? Consider whether
the object placement supports the intended
 ↪ activity, access direction, grouping,
and any explicit spatial requirements in
 ↪ the request.

For EACH question pick exactly one verdict:
 "A" -- Image A is clearly better
 "B" -- Image B is clearly better
 "tie" -- roughly equal, or no clear
 ↪ winner
and give a concise reason.

Respond with ONLY a JSON object, no extra
 ↪ text, in exactly this form:
{
 "tidy": {"winner": "A", "reason": "..."},
 "user_intent": {"winner": "A", "reason":
 ↪ "..."}
}

```

## G Template-Clustered Re-analysis of the Main Results

The tests of Appendix A.6 treat the 1,000 scenes as independent. This appendix relaxes that assumption.

**Procedure.** The 1,000 scenes are instantiated from 100 templates, 10 per template (§4), so the tests of Appendix A.6 treat as independent a sample that is nested by construction. We therefore repeat all ten comparisons treating each template as a cluster (Table 12). For every comparison we estimate the intra-cluster correlation of the binary win indicator among decided pairs, the resulting design effect  $1 + (m_0 - 1)\text{ICC}$  with  $m_0$  the mean cluster size, and the effective sample size  $n_{\text{eff}} = n_{\text{dec}}/\text{DEFF}$ ; we then re-test the win rate against 0.5 with a cluster-robust (sandwich) Wald statistic referred to a  $t$  distribution on 99 degrees of freedom, and Holm-correct across the ten clustered tests. As a check that does not depend on the sandwich approximation, we also collapse each template to a single observation — its win rate among its own decided pairs — and apply a sign test to the 100 template-level values. A cluster bootstrap over templates (20,000 resamples) gives intervals matching the sandwich ones to within a percentage point throughout.

Clustering costs real power and we report it as such. Standard errors inflate by 1.2–1.7 $\times$  for the ablation comparisons and by 2.1–2.3 $\times$  for the oracle row, and the effective sample size falls from 655–840 decided pairs to 163–476. The

conclusions of §5.3 nonetheless survive: all six comparisons of ORGANIZEIt against the three ablations remain significant after Holm correction, the weakest at  $p = 2.4 \times 10^{-6}$ , and the template-level sign test agrees, with 64–86 of the 87–98 decided templates favoring ORGANIZEIt in each case. Two secondary readings do not survive and should be qualified accordingly. The oracle row stays significant on tidiness ( $p = 8.3 \times 10^{-9}$ ) but not on user intent ( $p = 0.054$ ; 44 templates against 52), so the narrowing of the human-reference gap on user intent (§5.3) is a descriptive observation rather than a tested one. In the last row, the tidiness advantage of *VLM w/ goal* over *VLM w/o goal* holds ( $p = 0.033$ ) while its user-intent disadvantage does not ( $p = 0.054$ ; sign test  $p = 0.12$ ); the claim that naive conditioning “can even hurt” therefore rests on the descriptive margin alone.

**Why the template clusters are weak.** The measured ICCs, 0.07–0.25 across the ablation comparisons, are modest for a design with ten observations per cluster, and the instantiation procedure explains why. A template fixes only the object categories and their counts, which is the one thing its ten scenes do share. Everything else is resampled. The assets realizing those categories are drawn interchangeably, leaving the ten scenes with near-disjoint inventories: the mean pairwise Jaccard similarity of asset sets within a template is 0.163, and in 98 of 100 templates all ten asset multisets are distinct. Table texture is resampled independently, giving a mean of 7.4 distinct textures per ten scenes. The DSL parameters are perturbed within their admissible ranges, with a median jitter of 3 cm in  $x$  and 5 cm in  $y$ . And the messy scene that each system actually observes is produced by a physics drop onto a random unoccupied region, retaining 89% of the dataset-wide positional spread within a single template (Appendix A.7). Two scenes from one template thus share a category list and little else, which is why grouping by template removes far less independent information than the nominal ten-to-one nesting would suggest.

| Comparison          | Crit. | ICC   | DEFF | $n_{\text{eff}}$ | CR 95% CI    | $p$ (CR, Holm)        | $T^+$ | $T^-$ | $p$ (sign)            |
|---------------------|-------|-------|------|------------------|--------------|-----------------------|-------|-------|-----------------------|
| vs. Perception-only | Tid.  | 0.111 | 1.71 | 433              | [72.1, 80.2] | $1.1 \times 10^{-21}$ | 86    | 12    | $5.9 \times 10^{-15}$ |
| vs. VLM w/ goal     | Tid.  | 0.151 | 1.95 | 374              | [67.8, 77.3] | $1.5 \times 10^{-14}$ | 76    | 17    | $4.5 \times 10^{-10}$ |
| vs. VLM w/o goal    | Tid.  | 0.115 | 1.81 | 443              | [70.7, 79.0] | $7.1 \times 10^{-20}$ | 82    | 11    | $1.4 \times 10^{-14}$ |
| vs. Reference goal  | Tid.  | 0.441 | 4.26 | 197              | [21.6, 34.6] | $8.3 \times 10^{-9}$  | 23    | 71    | $7.3 \times 10^{-7}$  |
| VLM w/ vs. w/o goal | Tid.  | 0.117 | 1.77 | 429              | [51.5, 61.3] | 0.033                 | 57    | 34    | 0.021                 |
| vs. Perception-only | UI    | 0.068 | 1.38 | 476              | [61.9, 70.6] | $2.8 \times 10^{-10}$ | 69    | 18    | $3.3 \times 10^{-8}$  |
| vs. VLM w/ goal     | UI    | 0.232 | 2.35 | 291              | [63.1, 74.3] | $9.3 \times 10^{-9}$  | 69    | 23    | $1.7 \times 10^{-6}$  |
| vs. VLM w/o goal    | UI    | 0.248 | 2.56 | 285              | [59.8, 71.4] | $2.4 \times 10^{-6}$  | 64    | 28    | $2.2 \times 10^{-4}$  |
| vs. Reference goal  | UI    | 0.559 | 4.86 | 163              | [33.2, 49.0] | 0.054                 | 44    | 52    | 0.475                 |
| VLM w/ vs. w/o goal | UI    | 0.154 | 1.89 | 360              | [39.5, 50.1] | 0.054                 | 38    | 54    | 0.117                 |

Table 12: Template-clustered re-analysis of Table 7, treating each of the 100 templates as a cluster. ICC is the intra-cluster correlation of the win indicator among decided pairs;  $\text{DEFF} = 1 + (m_0 - 1) \text{ICC}$ ;  $n_{\text{eff}} = n_{\text{dec}}/\text{DEFF}$ , to be read against the  $n_{\text{dec}}$  of Table 7. The interval and  $p$ -value are cluster-robust (sandwich,  $t_{99}$ ), with Holm applied across the ten tests.  $T^+$  and  $T^-$  count templates whose own win rate is above and below 50%; templates tied at exactly 50% enter neither and the sign test is taken over the rest.