MessyNav: Zero-Shot Navigation via Manipulation in Messy Environments

Kenneth Llontop[1,2] **Cherie Ho**[1] **Dylan Zhou**[1] **Anuva Banwasi**[1]
Priya Sundaresan[1] **Carlota Parés-Morlans**[1] **Jeannette Bohg**[1]

[1]Stanford University [2]UC Berkeley

messynav.github.io



Figure 1: **MessyNav enables zero-shot open-world interactive navigation. [Top]** MessyNav reasons zero-shot about object affordances using fine-grained semantic and contextual cues, identifying the utility cart carrying a plastic bin as interactive (swipeable) and the otherwise movable crate carrying unstable stacked blocks as non-interactive. The robot then swipes the cart aside to clear a path and reach the goal without task-specific training or system modifications. **[Bottom]** MessyNav deployed across diverse real-world indoor and outdoor environments.

Abstract: Real-world environments are messy, with paths often blocked by everyday objects like chairs and toys, requiring robots to perform interactive navigation by manipulating obstacles to clear a path. Existing methods rely on extensive simulation-based learning or per-scene instructions specifying which objects to avoid or manipulate. They struggle to generalize across novel objects, diverse environments, and varied contexts such as a cart of fragile glassware that should be avoided while an empty one can be pushed. We formalize this problem as open-world interactive navigation and present MessyNav, a zero-shot framework based on vision-language models (VLMs). At its core, MessyNav performs explicit VLM-based affordance reasoning before task planning to decide whether to swipe, pick up, or avoid an object and produce a high-level plan. Evaluated on a mobile manipulator across real-world indoor and outdoor environments, MessyNav improves success rate by 27 percentage points (pp) over the strongest baseline in real-robot trials and achieves statistically significant gains in a larger-scale offline evaluation, without task-specific training.

Keywords: Mobile Manipulation, Interactive Navigation, Affordance Reasoning

10th Conference on Robot Learning (CoRL 2026), Austin TX, USA.

1 Introduction

Picture a mom retrieving her child's backpack at daycare. To get to the closet, she needs to push aside a chair and maneuver around a fragile toy train set. It is natural for humans to navigate such scenarios by actively manipulating their environment to create paths. However, robots running classical navigation algorithms treat every obstacle, whether a light toy or an immovable wall, as an obstruction to avoid, often failing in cluttered environments where progress requires interaction.

In messy settings especially, robots need interactive navigation: the ability to move through an environment by manipulating the objects that block their path. While classical planning is primarily a geometric problem, interactive navigation is a semantic decision problem under geometric constraints. When blocked, a robot must reason about how to move an object, and before that, whether it should be moved at all. This decision depends on fine-grained semantic and contextual cues about fragility, portability, and function that are difficult to infer from geometry alone and costly to capture through task-specific training. We refer to this problem as *open-world interactive navigation*, where success requires generalizing across unseen objects, environments, and contexts.

Existing interactive navigation approaches fall short: prior methods [1, 2, 3, 4] rely on simulation-based learning or predefined category-level affordances, limiting their ability to generalize to new objects and contexts. These limitations make it challenging to operate in messy, real-world environments requiring fine-grained semantic reasoning, such as recognizing that a robot can safely push a wheeled chair but must avoid an adjacent box labeled "fragile". Conversely, classical navigation stacks [5] do not require training data but treat all obstacles as static geometric boundaries, planning only collision-free paths without reasoning about object interactability. Achieving open-world interactive navigation requires moving beyond both hand-crafted geometric heuristics and data-intensive simulation toward the zero-shot, commonsense reasoning provided by internet-scale VLMs.

In this work, we present `MessyNav`, a framework for zero-shot interactive navigation in open-world, cluttered environments. Given an egocentric image and a navigation goal, `MessyNav` infers object-level affordance labels from appearance and context, then passes these labels to a VLM-based task planner that generates high-level actions which are executed through navigation and manipulation primitives (Figure 1).

The main contributions of this work are:
(1) Problem formulation: We provide the first formalization of *open-world* interactive navigation: given an egocentric observation and a navigation goal, the agent must decide whether interaction is necessary and, if so, how to interact with obstructing objects to reach the goal, while generalizing zero-shot across (i) object types, (ii) environments, and (iii) semantic contexts.
(2) Framework: We present `MessyNav`, a zero-shot framework whose *core design* separates affordance reasoning (via an explicit VLM module) from task planning. This explicit prior lets the robot make better context-dependent decisions. Ablations show this separation raises interactive navigation success from 40% to 67% over end-to-end VLM planning.
(3) Open-world affordance benchmark: We release a real-world benchmark for instance-level, context-dependent affordance reasoning, consisting of 100 cluttered scenes with ground-truth interactability labels, on which `MessyNav` achieves 85.7% affordance accuracy, 19.3 percentage points above the closed-set oracle baseline that relies on predefined category labels.

Finally, we evaluate `MessyNav` across all three axes of open-world variation, including novel objects, varied contexts, and diverse environments (indoor and outdoor, day and night), without any system modifications. By formalizing this problem and providing a concrete system, `MessyNav` offers a foundation for *in-the-wild* interactive navigation.

2 Related Work

We give an overview of related work below, with a more detailed discussion in the Appendix.

Robot Navigation. Traditional navigation frameworks like Nav2 [5] rely on occupancy costmaps, treating all occupied regions as impassable and failing in traversable clutter. Conversely, end-to-end policies [6, 7, 8, 9, 10, 11] learn expressive motions directly from massive, cross-embodiment datasets. To reduce data overhead, recent methods combine internet-scale VLMs with 3D scene representations for zero-shot, open-vocabulary navigation [12, 13, 14]. However, these approaches are navigation-centric and fail when no collision-free path exists. `MessyNav` instead pairs a traditional navigation stack for standard avoidance with VLM-guided manipulation to clear blocking obstacles.

Mobile Manipulation. One paradigm in mobile manipulation uses large language models (LLMs) to decompose abstract instructions into sequences of open-loop primitives like picking and placing [15, 16, 17]. Another paradigm learns end-to-end policies for fine-grained, whole-body coordination [18, 19, 20, 21, 22, 23, 24]. Yet, these methods are primarily manipulation-first, interacting only with objects explicitly named in the task specification. `MessyNav` treats manipulation strictly as a functional tool to clear paths when navigation is blocked.

Interactive Navigation. Interactive navigation allows robots to manipulate obstacles to reach a goal [25]. Most learning-based methods [1, 2, 3, 26, 27] rely on data-intensive simulation and reward engineering, limiting generalization to novel real-world scenarios. Recent works leverage language models for interactive navigation [28, 29]. However, their affordance reasoning relies primarily on geometric feasibility or simulation training with predefined object categories. Neither explicitly considers instance-level semantic context (e.g., surroundings, signs, contents) when determining whether an object is suitable to manipulate. Closest to our work, Zhang et al. [4] use VLMs for scene reasoning but require per-scene human instructions and rigid, category-level affordances. In contrast, `MessyNav` enables zero-shot, instance-level reasoning over diverse interaction primitives by coupling VLM reasoning with classical planning, requiring no training or simulation.

3 Problem Formulation

Interactive navigation [3, 25] addresses scenarios where a robot must actively manipulate movable obstacles to clear a path to its goal, rather than just avoiding them. To better reflect the complexity of real-world environments, we extend this setting to *open-world* interactive navigation, where the robot must reason about novel objects and their specific context. We define three axes of variation along which an open-world interactive navigation system should generalize zero-shot.

(1) Novel objects: Real environments contain objects that cannot be accounted for in advance, so a robot must infer whether and how to interact with objects it has never encountered. For example, it might find a rolled yoga mat in a hallway or cables on the floor attached to a valuable instrument.

(2) Diverse environments: A deployed robot must operate across varied environments without any scene-specific tuning. The system must generalize across different rooms (kitchens, hallways, living areas), locations (indoor, outdoor), and lighting conditions (daytime, nighttime).

(3) Varied contexts: Affordance is not just a property of an object, but of an object given its surroundings. The same object may be interactive or not depending on its context: a packaged extension cord is movable, but one plugged into a socket should be avoided. Individual toy blocks may be pushable, but when assembled as a child's creation, they are fragile, valuable possessions.

`MessyNav` is designed to meet all three requirements without task-specific training or prompting.

4 Approach

We present `MessyNav`, an interactive navigation framework that leverages VLMs for zero-shot affordance reasoning and task planning to go beyond closed-set prediction or environment-specific tuning. Our central design decision is to separate *what to interact with* from *how to interact*: an explicit affordance reasoning stage first decides which objects can be moved, and only then does a task planner decide how to move them. We make affordance reasoning an explicit stage rather than

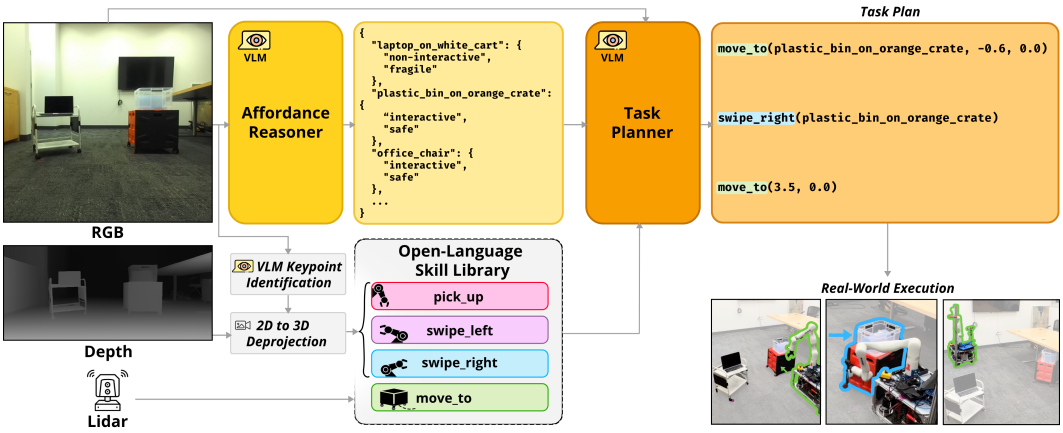


Figure 2: **`MessyNav`: A framework for zero-shot open-world interactive navigation.** `MessyNav` consists of an *Affordance Reasoner* that infers object interactability from an RGB image using a VLM to evaluate both robot-centric physical feasibility and object-centric interaction safety. For example, it will label a cart with a laptop as non-interactive while labeling a wheeled crate with a bin as interactive. Given the object affordance labels, a *Task Planner* then sequences manipulation and navigation actions from an open-language skill library containing a simple set of motion primitives (`pick_up`, `swipe_left`, `swipe_right`) and a LiDAR-based navigation stack (`move_to`) that we find can tackle a wide range of interactive navigation scenarios. Finally, the generated plan is executed by grounding object references in 3D coordinates using VLM keypoint identification, semantic segmentation, and depth-based deprojection.

letting a task planner implicitly handle affordance reasoning, allowing it to make more grounded decisions in context-dependent scenarios. In Section 5, we ablate this core design by comparing `MessyNav` to end-to-end VLM planning.

As shown in Figure 2, the framework consists of three components: **(1) Affordance Reasoner**: Identifies which objects are interactive based on robot feasibility and object properties. **(2) Task Planner**: Generates a sequence of navigation and manipulation actions based on the affordance labels to clear the blocking objects and reach the goal. **(3) Open-Language Skill Library**: Grounds open-language object references in task plans to 3D world coordinates and executes motion primitives. The task description is specified as "`Navigate to (goal_x, goal_y) and handle any obstacles in your path`." We refer the reader to the Appendix for details of the prompts used for VLM affordance reasoning, task planning, and keypoint identification.

4.1 VLM Affordance Reasoning

To navigate through clutter, the robot must first determine which objects can be moved, a decision that depends not just on geometry but on fine-grained contextual cues such as warning tape around a chair or a fragile label on a box. We leverage VLMs, which encode commonsense knowledge from internet-scale pretraining, to reason about object interactability zero-shot from an RGB image of the scene and a task description. For each object, the affordance reasoner outputs an affordance label (interactive or non-interactive) and a property (safe, fragile, or immovable) that explains the predicted affordance.

The reasoner applies a two-stage process. First, it evaluates *robot-centric physical feasibility* based on context embedded into the prompt, which specifies the robot's configuration, gripper payload limits, and manipulation skills. Second, for objects the robot can physically move, the reasoner determines *object-centric interaction safety*. This is based on guidelines embedded into the prompt for when to avoid interaction, determined by potential damage to the robot or environment, and object attributes such as fragility, value, and whether the arrangement appears intentional. An object is only labeled as *interactive* if it passes both stages. Otherwise, its affordance is labeled as *non-*

interactive and and its property is set to *fragile* or *immovable*. As objects often function together in ways that affect interactability, such as a set of china cups on a table that makes the combined unit fragile, the reasoner identifies such cases and treats them as a single unit when determining labels.

4.2 VLM Task Planner

We query the VLM with the affordance labels given by the reasoner, the current scene RGB image, and a planning prompt which enumerates a library of available manipulation skills detailed in Section 4.3. The planner is prompted to analyze free space in the scene, identify objects that block the path, and select appropriate actions. The prompt encodes action decision guidelines for the planner, such as encouraging *pick* for smaller objects that can be grasped and carried, or *swipe* for larger objects that can be cleared with a lateral motion. The output is a structured plan, such as `move_to(suitcase, -0.6, 0), swipe_left(suitcase), move_to(goal_x, goal_y)`.

4.3 Open-Language Skill Library

The task planner outputs actions with either open-language object references (e.g., `swipe_right(suitcase)`) or world-frame coordinates (e.g., `move_to(2.0, 1.5)`). To execute these in the physical world, we ground object names to 3D coordinates when needed and execute motion primitives.

Grounding Language Descriptions to 3D: For actions with object references, we leverage VLMs' open-vocabulary pointing and segmentation capabilities to localize objects in the RGB image. Following HoMeR [21], we then deproject the 2D image coordinate(s) to a robot-centric point cloud using depth and known camera intrinsics, then transform to world frame using the robot's current pose to obtain the object's world-centric 3D coordinate.

Motion Primitives: In this instantiation, we design a small set of motion primitives covering a wide range of interactive navigation scenarios: `move_to()` navigates to a world-frame coordinate or an object; `pick_up()` grasps an object with the arm; and `swipe_left()`/`swipe_right()` laterally clears an object with the arm. Each primitive uses the VLM's pointing and segmentation modules to localize objects and compute 3D grasp/approach points. ROS2 Nav2 [5] and LiDAR are used for `move_to()`'s obstacle avoidance. See Appendix B for full implementation details. As `MessyNav` targets high-level reasoning and is agnostic to skill implementation, we deliberately use simple primitives to isolate the contribution of explicit affordance reasoning from the quality of the underlying controllers. The library is extensible: the primitives can be replaced with richer policies such as vision-language-action (VLA) models, and the affordance reasoner's predicted properties (e.g., fragility) can be passed to parameterize execution, for instance by modulating speed and force.

5 Experiments

In this section, we evaluate the effectiveness of `MessyNav` for open-world interactive navigation.

5.1 Does explicit affordance reasoning enable open-world generalization?

Task. We evaluate `MessyNav`'s affordance reasoning capability in isolation from the full framework. Given an RGB image containing one or more obstacles, the task is to classify whether each obstacle is *interactive* (can be swiped or picked up) or *non-interactive* (must be avoided).

Dataset. In the absence of an established benchmark for this task, we collect and manually annotate an evaluation set of 100 images totaling 217 objects across 14 locations, each containing 1–4 foreground objects with binary interactive/non-interactive labels. The set contains 69% indoor and 31% outdoor scenes under bright artificial (66%), natural (28%), and dim (6%) lighting, with obstacle sizes ranging from small electronics to large trees. Of the 217 objects, 94 (43.3%) are interactive and 123 (56.7%) are non-interactive. Affordance labels were annotated by an expert familiar with

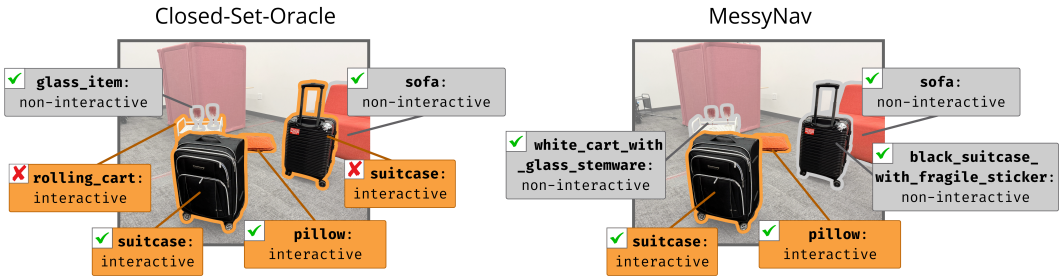


Figure 3: **`MessyNav` reasons about context, not just object category.** In this cluttered scene, `closed-set-oracle` assigns affordances by category alone, misclassifying the rolling cart as interactive (ignoring the glassware it carries) and the suitcase as interactive (ignoring its fragile sticker). In contrast, `MessyNav` reasons at the instance level. It treats the cart and the glass stemware as a single fragile unit and reads the sticker on the suitcase as a contextual cue, correctly labeling both as non-interactive.

Table 1: **Affordance Reasoning Results.** `MessyNav` outperforms geometric and closed-set baselines, highlighting the importance of context-dependent affordance reasoning.

Method	Acc. ↑
`MessyNav`	**85.7%**
`height-only` [5]	59.9%
`closed-set` [4]	24.4%
`closed-set-oracle` [4]	66.4%

Table 2: **Real-World Interactive Navigation Results.** `MessyNav` achieves the highest success rate across 15 real-world scenes. Success requires reaching the goal without any unsafe interactions. *Execution time (excluding VLM queries) averaged across successful trials.

Method	Succ. (%) ↑	Exec.* (s) ↓
`MessyNav`	**67 (10/15)**	**45.8**
`height-only` [5]	13 (2/15)	24.5
`vlm-planner-only`	40 (6/15)	45.9
`closed-set-planner` [4]	27 (4/15)	50.4

the robot's physical capabilities. To stress-test instance-level reasoning, the set deliberately includes same-category objects whose affordance differs with context (Appendix C, Table 6), as well as small but non-interactive objects where height-based heuristics perform poorly. The images span the three axes of generalization: diverse object types (e.g., vases, suitcases), environments (indoor, outdoor, outdoor at night), and contextual variations (e.g., empty carts vs. carts with a laptop precariously on top). Note that this dataset is only used for evaluation, not for training or finetuning.

Metrics. We measure classification accuracy against human-annotated labels. Accuracy is the fraction of objects whose interactivity is correctly classified.

Baselines. We compare against three baselines reflecting common assumptions in interactive navigation. **(1) `height-only`** [5]: a purely geometric heuristic (*i.e.*, Nav2) that treats any object taller than 0.2m as non-interactive and therefore an object to avoid. **(2) `closed-set`** [4]: the closest baseline to `MessyNav`, it uses an open-vocabulary detector [30] to localize objects named in human-provided instructions, which specify each object's interactability. However, this method requires per-scene instructions, which are unavailable in our zero-shot evaluation setting. For a fair comparison, we instead generate a fixed list offline of 100 object categories (50 interactive, 50 non-interactive) with `gemini-3-flash-preview` [31]. The prompt used to generate the object categories can be found in Appendix C. Objects outside this list are ignored, and affordances are fixed per category (e.g., "cart" is labeled as interactive, with no distinction for specific instances like a cart carrying glassware). **(3) `closed-set-oracle`**: an upper bound on `closed-set`, where the object categories are derived from the items in the evaluation set (note that interactivity labels are fixed and not context-dependent).

Results. Table 1 shows that `MessyNav` substantially outperforms all baselines, reaching 85.7% accuracy. The `height-only` baseline achieves 59.9% accuracy, frequently misclassifying objects

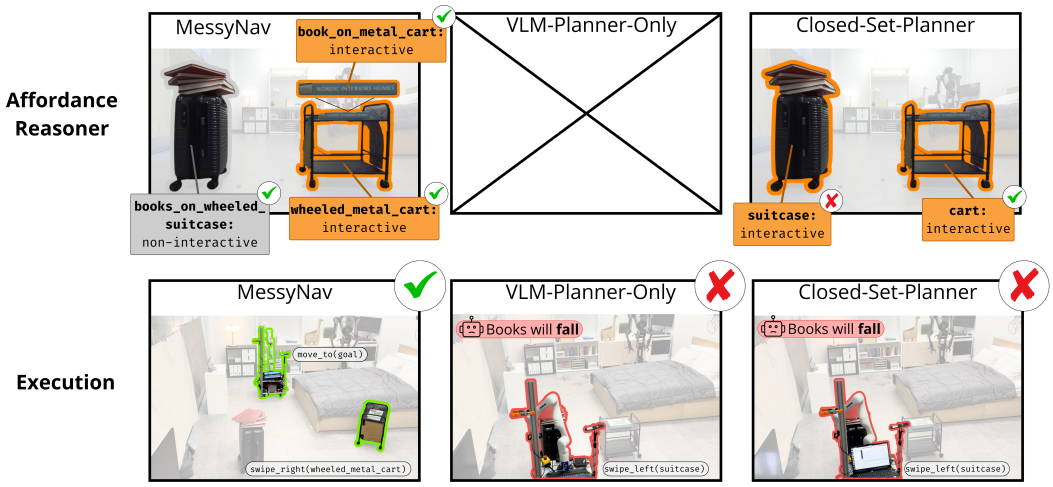


Figure 4: **Explicit affordance reasoning improves context-dependent task planning.** In this scene, a suitcase supporting a stack of books is inherently unstable, while the black cart holding a single book can be safely cleared. **[Left]** `MessyNav` reasons about affordances before planning, correctly labeling the suitcase as non-interactive and the cart as interactive, and swiping the cart aside to clear the path. **[Middle]** `vlm-planner-only` plans directly from the image without an explicit affordance stage and incorrectly decides to swipe the unstable suitcase. **[Right]** `closed-set-planner` assigns fixed category-level affordances that misclassify the suitcase as interactive, leading to an unsafe plan.

that violate simple geometric heuristics. This baseline ignores semantics and context, failing on objects that are tall but movable (e.g., suitcase, rolling cart) or short but fragile. The `closed-set` baseline performs worst (24.4%), as the predefined taxonomy fails to capture many novel objects. We would expect this result, as real environments in an open-world setting inevitably contain objects that cannot be accounted for in advance. Even `closed-set-oracle` underperforms `MessyNav`, demonstrating that predefined category-level labels are insufficient for the context-dependent affordance reasoning required for interactive navigation. `MessyNav` excels in scenarios where affordances depend on broader context rather than object category alone. These results show that interactive navigation cannot be reliably solved using fixed geometric heuristics or category-level affordance labels alone. Open-world, instance-level, and context-aware reasoning is essential.

Sensitivity to VLM choice. To test whether the performance depends on the choice of VLM, we repeat the evaluation using different VLMs for affordance reasoning, while keeping the prompt and all other components fixed. With `claude-opus-5` [32], `gemini-3-flash-preview` [31], and `gpt-5.6-luna` [33], `MessyNav` achieves 87.1%, 85.7%, and 86.2%, compared to 69.1%, 66.4%, and 67.3% for `closed-set-oracle`, consistently outperforming it by $\sim$18 points.

5.2 Does explicit affordance reasoning outperform closed-set and end-to-end VLM planning in real-world interactive navigation?

Task/Baselines. We evaluate `MessyNav` on real-world navigation tasks with full end-to-end execution, comparing against a `height-only` navigation stack [5]. In addition, we include two ablations that directly isolate the impact of our core design decision of explicit VLM-based affordance reasoning prior to VLM task planning. `vlm-planner-only` removes the affordance reasoning module, using only `MessyNav`'s VLM task planner. `closed-set-planner` replaces `MessyNav`'s affordance reasoner with the `closed-set-oracle` [4] affordance reasoner.

Metrics. We report success rate, the percentage of trials where the robot reaches the goal without executing unsafe actions, and execution time averaged across successful trials.

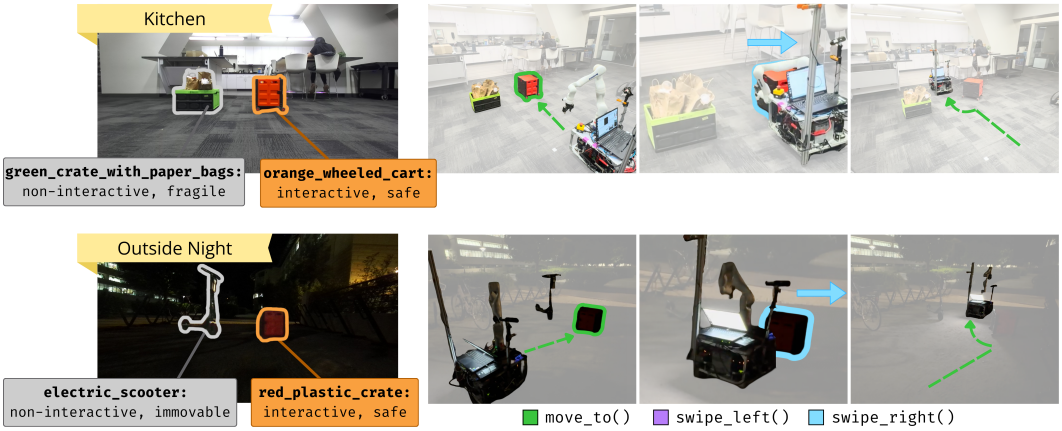


Figure 5: **Qualitative Results across Environments.** We deploy `MessyNav` in varied settings to test its zero-shot capability. The robot successfully reasons about affordances and executes interactions across different lighting conditions, environments, and object types.

Hardware. For real-world experiments, we deploy `MessyNav` on the TidyBot++ platform [18], consisting of a 7-DoF Kinova arm and a holonomic mobile base. We additionally equip the robot with a Unitree L2 LiDAR for base odometry and LiDAR mapping via FAST-LIO [34], and two ZED 2 cameras: one front-facing on the base, and one pole-mounted to the right of the arm pointing toward the front of the robot base. The base camera is used for planning, while both cameras are used for object localization and segmentation. We include more hardware details in Appendix B.

Scenarios. We evaluate all four methods across 15 real-world scenes in two environments, a messy bedroom and a conference room, for a total of 60 robot trials. The scenes span 5 object types (e.g., suitcase, box with vase, stuffed toys) with 3–7 context variations per type (e.g., cart with laptop vs. cart with empty bin). Table 8 lists the full set.

Results. Table 2 shows `MessyNav` achieves the highest success rate (10/15), outperforming all baselines on real-world interactive navigation. The two ablations isolate the contribution of explicit affordance reasoning. `vlm-planner-only` succeeds when the correct action is visually obvious but fails on contextually difficult cases (e.g., a suitcase with vs. without caution tape), since folding affordance into the planner misses cues like fragility, stability, and value. Notably, `closed-set-planner` performs *worse* than `vlm-planner-only`, indicating inaccurate predefined affordance estimates are worse than no prior affordance conditioning at all, and that it is `MessyNav`'s open-world affordance reasoning that drives the gains. Figure 4 illustrates this. In a scene where a suitcase with a stack of books is unstable but an adjacent cart with one book is safely movable, only `MessyNav` classifies both correctly and swipes the cart out of the way. Without explicit affordance reasoning, `vlm-planner-only` decides to swipe the non-interactive suitcase. `closed-set-planner` misclassifies the suitcase as interactive, leading to the same incorrect plan. Appendix Figure 9 provides a failure breakdown of `MessyNav` against the strongest baseline.

Finally, we test `MessyNav` across varied real-world environments, including multiple indoor and outdoor locations, day and night, with diverse objects (suitcase, wagon, cardboard box, cart, scooter) and contextual variations (e.g., a tilted wagon with bottles spilling out). Figure 5 shows representative cases where `MessyNav` reasons over objects and their context to decide what is safely movable, such as swiping the empty cart while avoiding the crate with overflowing paper bags.

5.3 Does explicit affordance reasoning improve planning performance at scale?

Task. The evaluation in Section 5.2 measures end-to-end performance but couples high-level decision quality to low-level execution with one trial per scene. To isolate decision quality and quantify VLM variability, we evaluate `MessyNav` and the strongest baseline, `vlm-planner-only`, of-

Table 3: **Scaled-Up Offline Planning Results.** We evaluate high-level planning performance across 55 real-world scenes in 5 environments, with 15 repeated trials per scene, assuming perfect low-level execution. `MessyNav` outperforms the strongest baseline from the real-world evaluation overall and under well-lit conditions, while nighttime remains challenging for both methods.

	Overall	By Visual Condition		
Method	**Success (%)**	**Outdoor Day**	**Indoor Lit**	**Night**
MessyNav	**62.4**	**78.3**	**62.8**	29.2
`vlm-planner-only`	55.4	60.8	55.5	**44.2**

fline on 55 scenes across 5 environments, scaling up from 15 scenes in 2 environments. A human evaluator assesses each plan assuming perfect low-level execution, marking a trial as successful if the plan reaches the goal without interacting with non-interactive objects. Each method is evaluated over 15 runs, where a run is one pass over all 55 scenes, yielding $n = 825$ trials per method.

Results. `MessyNav` achieves 62.4% success versus 55.4% for `vlm-planner-only`, a gain of 7.0 pp. Table 3 presents the results, with a breakdown of performance across visual conditions. A paired t-test over per-run success rates indicates that the improvement is statistically significant ($t(14) = 4.71$, $p < 0.001$; 95% CI on the difference $[+3.8, +10.2]$ pp). While this improvement is smaller than that reported in the real-world evaluation (Table 2), this evaluation assumes perfect execution and thus provides a complementary assessment of planning performance. `MessyNav`'s advantage is largest outdoors in daytime and persists indoors under good lighting. However, performance for both methods degrades under nighttime conditions, where `MessyNav` underperforms the baseline. We hypothesize that reduced visibility compromises the fine-grained visual cues required for affordance reasoning, motivating further research on robustness under poor illumination.

6 Conclusion

We present `MessyNav`, a zero-shot framework for interactive navigation in open-world, cluttered environments. Its core design decision is an explicit VLM-based affordance reasoning module prior to task planning, which lets `MessyNav` make improved context-dependent decisions that generalize across novel objects, diverse environments, and varied semantic contexts, all without task-specific training or simulation. We demonstrate that `MessyNav` enables zero-shot, generalizable interactive navigation in a wide range of real-world indoor and outdoor environments. We believe `MessyNav` brings robots one step closer to moving beyond tidy, structured environments and operating in the messy, unpredictable settings where robot deployment is most needed.

Limitations. While `MessyNav` demonstrates strong zero-shot interactive navigation in cluttered environments, several limitations motivate future research. First, `MessyNav` currently uses manipulation primitives that are executed open-loop. However, its modular framework can readily incorporate more expressive manipulation skills, such as VLA policies, and finer-grained affordance predictions to parameterize interactions. For example, the robot could adjust manipulation speed and force to handle fragile objects rather than avoiding them entirely. Second, affordance reasoning and task planning are limited by the underlying VLM, particularly under poor illumination and when reasoning about geometry from 2D images. Recent advances in VLM-based visual understanding and spatial reasoning are promising, though further research is needed for robust real-world operation. Finally, `MessyNav` focuses on making appropriate affordance decisions *before* acting, as some errors, such as toppling fragile objects, may be irreversible. However, its lack of a persistent memory limits longer-horizon planning and recovery, such as remembering that a backpack was too heavy to move. Integrating `MessyNav` into an agentic robotic system with closed-loop feedback and interaction-updated spatial memory [35] could enable experience-based affordance reasoning to avoid repeating suboptimal interactions. This could help robots to navigate unfamiliar, cluttered environments while improving through experience.

Acknowledgments

This work was supported in part by Toyota Research Institute. Kenneth Llontop is supported by the SEED Scholarship at the University of California, Berkeley, the Summer Undergraduate Research Fellowship (SURF) at Stanford University, and the NSF CISE Research Experiences for Undergraduates (REU) program. Cherie Ho is supported by the Croucher Fellowship for Postdoctoral Research. Priya Sundaresan is supported by the National Science Foundation Graduate Research Fellowship. Carlota Parés-Morlans is supported by the Knight-Hennessy Scholars program at Stanford University. We thank Satvik Sharma, Francis Engelmann, and Jennifer Grannen for insightful discussions, as well as the reviewers for their thoughtful feedback, which helped improve the paper.

References

[1] X. Wang, Y. Liu, X. Song, Y. Liu, S. Zhang, and S. Jiang. An interactive navigation method with effect-oriented affordance. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16446–16456, 2024. doi:10.1109/CVPR52733.2024.01556.

[2] P. Schoch, F. Yang, Y. Ma, S. Leutenegger, M. Hutter, and Q. Leboutet. IN-Sight: Interactive navigation through sight. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7794–7800, 2024. doi:10.1109/IROS58592.2024.10801928.

[3] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese. Interactive Gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020. doi:10.1109/LRA.2020.2965078.

[4] Z. Zhang, A. Lin, C. W. Wong, X. Chu, Q. Dou, and K. W. S. Au. Interactive navigation in environments with traversable obstacles using large language and vision-language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7867–7873, 2024. doi:10.1109/ICRA57147.2024.10610751.

[5] S. Macenski, F. Martín, R. White, and J. Ginés Clavero. The Marathon 2: A navigation system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725, 2020.

[6] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, and S. Levine. GNM: A general navigation model to drive any robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7226–7233, 2023.

[7] D. Shah, A. Sridhar, N. Dashora, K. Stachowicz, K. Black, N. Hirose, and S. Levine. ViNT: A foundation model for visual navigation. In *Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pages 711–733. PMLR, 2023.

[8] N. Hirose, C. Glossop, A. Sridhar, O. Mees, and S. Levine. LeLaN: Learning a language-conditioned navigation policy from in-the-wild video. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 666–688. PMLR, 2024.

[9] A. Sridhar, D. Shah, C. Glossop, and S. Levine. NoMaD: Goal masked diffusion policies for navigation and exploration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70, 2024.

[10] N. Hirose, C. Glossop, D. Shah, and S. Levine. OmniVLA: An omni-modal vision-language-action model for robot navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

[11] C. Glossop, W. Chen, A. Bhorkar, D. Shah, and S. Levine. CAST: Counterfactual labels improve instruction following in vision-language-action models. *IEEE Robotics and Automation Letters*, 11(10):11785–11792, 2026. doi:10.1109/LRA.2026.3726383.

[12] N. M. M. Shafiullah, C. Paxton, L. Pinto, S. Chintala, and A. Szlam. CLIP-Fields: Weakly supervised semantic fields for robotic memory. In *Robotics: Science and Systems (RSS)*, 2023. doi:10.15607/RSS.2023.XIX.074.

[13] C. Huang, O. Mees, A. Zeng, and W. Burgard. Visual language maps for robot navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 10608–10615, 2023. doi:10.1109/ICRA48891.2023.10160969.

[14] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. M. de Melo, J. B. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull. ConceptGraphs: Open-vocabulary 3D scene graphs for perception and planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028, 2024. doi:10.1109/ICRA57147.2024.10610243.

[15] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

[16] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. SayPlan: Grounding large language models using 3D scene graphs for scalable robot task planning. In *Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pages 23–72. PMLR, 2023.

[17] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser. TidyBot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.

[18] J. Wu, W. Chong, R. Holmberg, A. Prasad, Y. Gao, O. Khatib, S. Song, S. Rusinkiewicz, and J. Bohg. TidyBot++: An open-source holonomic mobile manipulator for robot learning. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 3729–3741. PMLR, 2024.

[19] Z. Fu, T. Z. Zhao, and C. Finn. Mobile ALOHA: Learning bimanual mobile manipulation using low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 4066–4083. PMLR, 2024.

[20] L. Y. Zhu, P. Kuppili, R. Punamiya, P. Aphiwetsa, D. Patel, S. Kareer, S. Ha, and D. Xu. EMMA: Scaling mobile manipulation via egocentric human data. *IEEE Robotics and Automation Letters*, 2026.

[21] P. Sundaresan, R. Malhotra, P. Miao, J. Yang, J. Wu, H. Hu, R. Antonova, F. Engelmann, D. Sadigh, and J. Bohg. HoMeR: Learning in-the-wild mobile manipulation via hybrid imitation and whole-body control. In *Conference on Robot Learning (CoRL)*, 2025.

[22] H. Xiong, R. Mendonca, K. Shaw, and D. Pathak. Adaptive mobile manipulation for articulated objects in the open world. *arXiv preprint arXiv:2401.14403*, 2024.

[23] J. Yang, I. Huang, B. Vu, M. Bajracharya, R. Antonova, and J. Bohg. Mobi-π: Mobilizing your robot learning policy. In *Conference on Robot Learning (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, pages 3516–3536. PMLR, 2025.

[24] H. Ha, Y. Gao, Z. Fu, J. Tan, and S. Song. UMI-on-Legs: Making manipulation policies mobile with manipulation-centric whole-body controllers. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 5254–5270. PMLR, 2024.

[25] M. Stilman and J. J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(4):479–503, 2005. doi:10.1142/S0219843605000545.

[26] C. Li, F. Xia, R. Martín-Martín, and S. Savarese. HRL4IN: Hierarchical reinforcement learning for interactive navigation with mobile manipulators. In *Conference on Robot Learning (CoRL)*, volume 100 of *Proceedings of Machine Learning Research*, pages 603–616. PMLR, 2019.

[27] S. Uppal, A. Agarwal, H. Xiong, K. Shaw, and D. Pathak. SPIN: Simultaneous perception interaction and navigation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18133–18142, 2024. doi:10.1109/CVPR52733.2024.01717.

[28] Y. Zhang and Y. Kantaros. NAMO-LLM: Efficient navigation among movable obstacles with large language model guidance. *IEEE Robotics and Automation Letters*, 10(12):13026–13033, 2025.

[29] K. Zhou, Z. Wen, Z. Zhuo, Z. Yan, P. Wu, I. H. U, S. Li, H. Gao, K. Ding, W. Cao, W. Pan, and C. Liu. CoINS: Counterfactual interactive navigation via skill-aware VLM. *arXiv preprint arXiv:2601.03956*, 2026.

[30] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang. Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. In *European Conference on Computer Vision (ECCV)*, 2024.

[31] Google DeepMind. Gemini 3 Flash model card. Technical report, Google DeepMind, Dec. 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf.

[32] Anthropic. System card: Claude Opus 5. Technical report, Anthropic, July 2026. URL https://www.anthropic.com/system-cards.

[33] OpenAI. GPT-5.6 system card. Technical report, OpenAI, July 2026. URL https://deploymentsafety.openai.com/gpt-5-6.

[34] W. Xu and F. Zhang. FAST-LIO: A fast, robust LiDAR-inertial odometry package by tightly-coupled iterated Kalman filter. *IEEE Robotics and Automation Letters*, 6(2):3317–3324, 2021. doi:10.1109/LRA.2021.3064227.

[35] A. Banwasi, W. Muckelroy III, P. Sundaresan, L. Zhao, J. Bohg, and C. Ho. MessyMem: Learning-from-doing memory for mobile manipulation. In *Conference on Robot Learning (CoRL)*, 2026.

[36] Google DeepMind. Gemini 2.5 Pro model card. Technical report, Google DeepMind, June 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Pro-Model-Card.pdf.

Appendix

A Extended Related Work

A.1 Mobile Manipulation

Beyond navigation, a significant body of research focuses purely on mobile manipulation, where the goal is to enable mobile embodiments such as wheeled systems, quadrupeds with arms, and

humanoid platforms to manipulate objects, often in workspaces far larger than what is feasible for tabletop manipulators. Several works achieve these goals by implementing a library of open-loop, vision-parameterized manipulation primitives like picking, placing, opening and closing of articulated objects, or tossing of objects, and utilize LLMs to decompose abstract instructions into sequences of these primitives, optionally with navigation between steps. For instance, SayCan, SayPlan, and TidyBot [15, 16, 17] leverage this approach to perform tasks like object retrieval and household cleanup zero-shot. On the other hand, data-driven end-to-end learning has also emerged as a popular approach for tackling more complicated kinds of mobile manipulation that require more fine-grained manipulation or whole-body coordination. Imitation learning approaches such as TidyBot++, Mobile ALOHA, EMMA, HoMeR, Mobi-π and others [18, 19, 20, 21, 22, 23], along with reinforcement learning approaches like UMI-on-Legs [24], achieve impressive performance on in-the-wild manipulation tasks in household and office settings. However, these methods remain fundamentally manipulation-first and only interact with objects in the environment that are explicitly part of the task specification. They do not consider using manipulation as an active means to resolve traversability failures or clear paths when the intended navigation route is blocked.

A.2 Interactive Navigation

More recently, there has been growing interest in interactive navigation, which moves beyond avoidance-only navigation by allowing robots to deliberately manipulate obstacles to reach a goal [25]. Early efforts in learning-based interactive navigation, formalized by works such as the iGibson 0.5 benchmark [3], primarily utilized reinforcement learning (RL) and hierarchical methods like HRL4IN and others [1, 26] to learn interaction policies like pushing obstacles, pressing buttons, grasping obstructing objects, or opening doors. However, these works have a high dependency on large-scale training in simulation and task-specific reward design, and do not explicitly demonstrate real-world deployment. Follow-up works like Zhang et al. [4] use VLMs to reason over a scene and construct costmaps; however, this approach oversimplifies objects as being either traversable or non-traversable, which rules out active manipulation. IN-Sight [2] similarly uses self-supervised training in simulation to learn object-oriented traversability scores, and then does sim2real transfer and integration with classical planning pipelines for interactive navigation through clutter in the real world. However, this approach is dependent on simulation data and the sim2real gap hampers fine-grained physical understanding. SPIN is another sim2real approach [27] which learns RL policies for manipulation and navigation that are deployed on a Hello Robot mobile manipulator. SPIN specifically learns one policy for controlling its egocentric camera gaze to plan collision-free whole-body motions, along with separate manipulation policies, but critically these are not used to resolve navigation blockages. To address this, approaches such as NAMO-LLM and CoINS [28, 29] have integrated VLMs for high-level reasoning over how to manipulate objects obstructing navigation. While promising, these methods typically abstract objects into fixed, broad semantic categories, discarding context-aware details like fragility, geometry, or portability that are critical for safe manipulation. In contrast, `MessyNav` enables zero-shot, fine-grained reasoning over a diverse set of interaction primitives by combining VLM-based reasoning with classical planning. This allows for reliable open-world interactive navigation in complex, unseen object arrangements, without *any* training or simulation.

B Real-World Implementation Details

B.1 Hardware

Figure 6 shows the TidyBot++ mobile manipulator [18], the robot used in the paper.

B.2 Affordance Reasoning

We query `gemini-3-flash-preview` [31] with `thinking=minimal` using the front-facing camera image. The prompt encodes the robot's physical constraints and evaluates each object

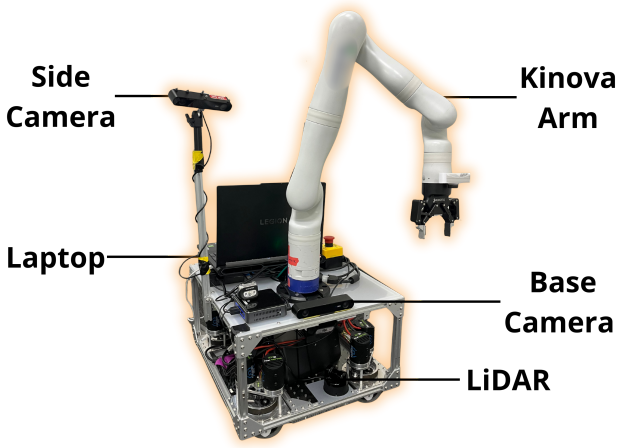


Figure 6: **Hardware:** We deploy on the TidyBot++ holonomic mobile manipulator [18] with two cameras and a LiDAR for perception. An onboard laptop runs the autonomy stack.

through two questions: (1) *can* the robot move it, and (2) *should* it be moved. The full prompt is below.

```
You are analyzing a scene for a mobile robot. The robot needs to
understand object properties so it can manipulate the objects in order
to navigate effectively.


TASK CONTEXT: {task_description}

Identify all the objects in the scene and assign labels (interactive
and non-interactive) based on the following criteria. Ignore the floor
and walls.

An object is labeled as interactive if and only if you deem the answer
is YES to both questions.
    1. Can the robot physically move the object? Reason about the
physical limitations of the robot.
        - The robot is a mobile manipulator with one arm and a 2
fingered gripper.
        - It can only carry 3kg in the gripper at a time.
        - It can swipe objects with its arm.
If the answer is YES, then verify the second question:
    2. Knowing the robot can move the object, should the robot be
moving this object? Answer NO if any of the following are true:
        - Moving the object could damage the robot or the environment
        - The object is fragile, breakable, unstable, damaged, or
carefully arranged
        - The object could become damaged if moved
        - The object is expensive or hard to replace
        - The object contents could spill or create a mess if moved
If the answer to any of the questions is deemed to be NO, then the
correct label is non-interactive.

Analyze all the objects that might affect robot navigation or task
completion individually. Identify any cases where two objects function
together (such as a cup on a table). In these cases, treat them as a
single object, create a combined name, and assign one appropriate label.

GROUNDING EXAMPLES:
- Empty wheeled chair → swipe_right(chair)
- Stuffed toy → pick_up_entity(stuffed_toy)
```

```
- heavy desk → avoid

Respond with ONLY a JSON object in this exact format:
{{
    "object_properties": {{
        "object_name_1": {{
            "mobility": "interactive_or_non-interactive",
            "interaction": "safe_or_fragile_or_immovable"
        }},
        "object_name_2": {{
            "mobility": "interactive_or_non-interactive",
            "interaction": "safe_or_fragile_or_immovable"
        }}
    }}
}}

Use clear, descriptive names (e.g., "red_box", "glass_vase",
"office_chair", "concrete_wall").
```

B.3 Task Planning

We query `gemini-3-flash-preview` [31] with `thinking=minimal` using the affordance labels from the affordance reasoning stage and the front-facing camera image to generate a task plan. Note that `move_to_nav2` and `pick_up_entity` in the prompt correspond to `move_to` and `pick_up` in the main text. The full prompt is below.

```
You are a robot task planner for a mobile robot with one arm and a
2-fingered gripper. The robot constraints:
- It can only carry about 3kg in your gripper at a time
- The gripper can open up to 8.5 cm wide
- The footprint is a 0.8m x 0.8m square

Analyze the image and use the detected object properties to plan your
actions. LOOK AT THE IMAGE CAREFULLY to assess FREE SPACE and use this
information to inform your planning.
The image is the front-facing view from the robot's perspective.

TASK: {task_description}

DETECTED OBJECT PROPERTIES (from Stage 1):
```json
{properties_str}
```

Based on these properties, decide your interaction strategy:
- INTERACTIVE + SAFE: Can pick or swipe
- NON-INTERACTIVE + FRAGILE/IMMOVABLE: Must avoid and maintain safe
distance

For INTERACTIVE objects, choose action based on efficiency and context:
- Pick: Objects that can be easily grasped and carried by the robot
- Swipe: Larger interactive objects - quick lateral clearing motion.
Note: You can NOT swipe objects below 0.3 meters in size.

AVAILABLE MOTION PRIMITIVES:
- move_to_nav2(x, y): Move to position (x, y) in meters
- move_to_nav2(object_name, dx, dy): Move to object with offset (dx, dy
in meters)
- pick_up_entity(object_name): Pick up the named object
- swipe_left(object_name): Swipe object to the left
- swipe_right(object_name): Swipe object to the right

NAVIGATION OFFSETS:
```

```
- Robot Origin: The robot starts at (0.0, 0.0) facing in the +x
direction
- Coordinate System:
    - +x: Forward movement (in front of the robot)
    - -x: Backward movement (behind the robot)
    - +y: Leftward movement (to the robot's left)
    - -y: Rightward movement (to the robot's right)
- dx: Forward/Backward offset in meters (positive = forward)
- dy: Left/Right offset in meters (positive = left)
- Want to be approximately 0.6m away from objects for manipulation
- Example: move_to_nav2(obstacle, -0.6, 0.0) → position 0.6m distance
from the obstacle (approaching it, stopping short)

PLANNING GUIDELINES:
- For movable objects, YOU decide pick vs swipe based on apparent
size/weight
- Position ~0.6m away from objects for manipulation
- Respect fragile/hazardous classifications from Stage 1
- You may not need to explicitly interact with all interactive objects
- only those that necessarily blocking the path you want to take.
- move_to_nav2 primitive calls a navigation planner that avoids
obstacles
- IMPORTANT: move_to_nav2 finds navigation sub goals so there is DO NOT
give intermediate waypoints

GROUNDING EXAMPLES:
- Empty wheeled chair → swipe_right(chair)
- Stuffed toy → pick_up_entity(stuffed_toy)
- heavy desk → avoid

Respond with ONLY a JSON object in this exact format:
{{
    "reasoning": "Explain your planning strategy based on object
properties and visual analysis",
    "geometric_analysis": "Analyze spatial constraints and navigation
paths",
    "interaction_modes": {{
        "object_name": "pick_or_swipe_or_avoid"
    }},
    "property_overrides": ["List any property assessments you're
overriding and why"],
    "plan": [
        "move_to_nav2(x, y)",
        "action_3"
    ]
}}
```

B.4 Open-Vocabulary Keypoint Detection and Segmentation

We use `gemini-2.5-pro` [36] for both 2D keypoint detection and segmentation to condition the motion primitives. As these queries generally occur when the robot is positioned close to the target object for manipulation, we first query the side arm-mounted camera, which has better visibility of the area directly in front of the robot. If the object is not found, we fall back to the front-facing base camera.

The prompt for 2D keypoint detection is as follows:

```
Point to the {prompt}. The {prompt} may have gaps or be partially
occluded.
Make sure your point is on a visible part of the {prompt}.
The answer should follow the json format: "
'[{"point": <point>, "label": <label1>}, ...]. '
"The points are in [y, x] format normalized to 0-1000.
```

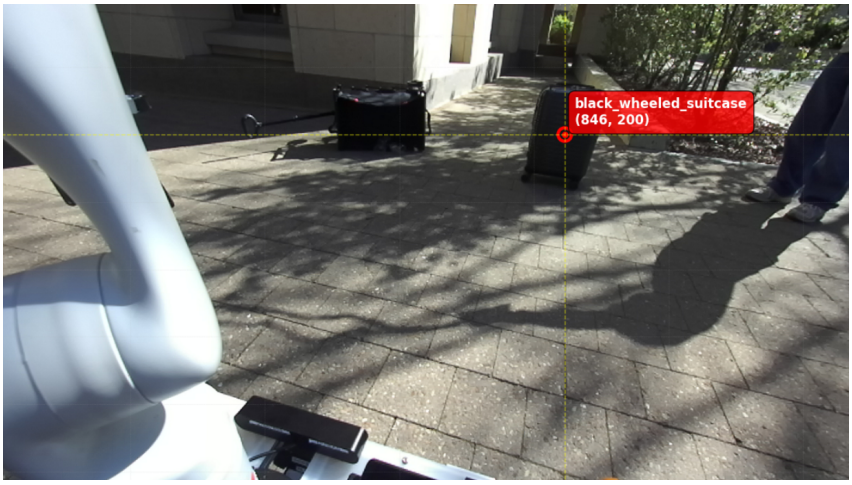


Figure 7: **Example Open-Vocabulary Keypoint Detection.** We use `gemini-2.5-pro` [36] to detect keypoints of objects specified by the task planner through language queries.

Figure 7 shows an example result of 2D keypoint detection on a black wheeled suitcase.

The prompt for 2D segmentation is as follows:

```
Find the following objects in this image: {objects_str}. For each
object, give me ONLY ONE segmentation mask for the single instance
you are most confident about. Focus ONLY on the foreground objects
and exclude any background elements like walls, floors, or surfaces.
The masks should tightly bound just the objects themselves, not any
background or supporting surfaces. Output a JSON list where each
entry contains the 2D bounding box in the key "box_2d", the
segmentation mask in key "mask", and the text label in the key
"label". Choose the clearest, most well-defined instance of each
object type and ignore any other instances. IMPORTANT: Use the EXACT
labels I provided in quotes - do not modify them. For example, if I
ask for "red block", your label must be exactly "red block", not
"red cube" or anything else.
```

B.5 Motion Primitives

In this instantiation of the framework, we design a small set of motion primitives that can address a wide range of interactive navigation scenarios:

- `move_to()`: Navigate to a world-frame coordinate, or offset from an object. For object-relative navigation, we use the VLM's pointing module to localize the object in 3D and compute the offset. We use the ROS2 Nav2 Navigation Stack [5] with a LiDAR sensor for obstacle avoidance.
- `pick_up()`: Grasp and carry an object with the robot arm. We use the VLM's pointing module for the grasping point and the VLM's segmentation to compute the object's 3D point cloud and principal axes for grasp orientation.
- `swipe_left() / swipe_right()`: Clear an object with the arm in a lateral direction. We use the VLM's segmentation to obtain the object's 3D point cloud to compute the approach position for swiping, a heuristic distance away from the object.

B.6 LiDAR-based Navigation Stack

We provide details on the navigation stack used both as the `height-only` baseline and within the `move_to` primitive. We use a Unitree L2 LiDAR for odometry estimation via FAST-LIO [34].

Table 4: **Navigation Costmap Parameters.**

Parameter	Value
Map resolution	0.05 m/pixel
Map size	30m $\times$ 30m
Max laser range	10m

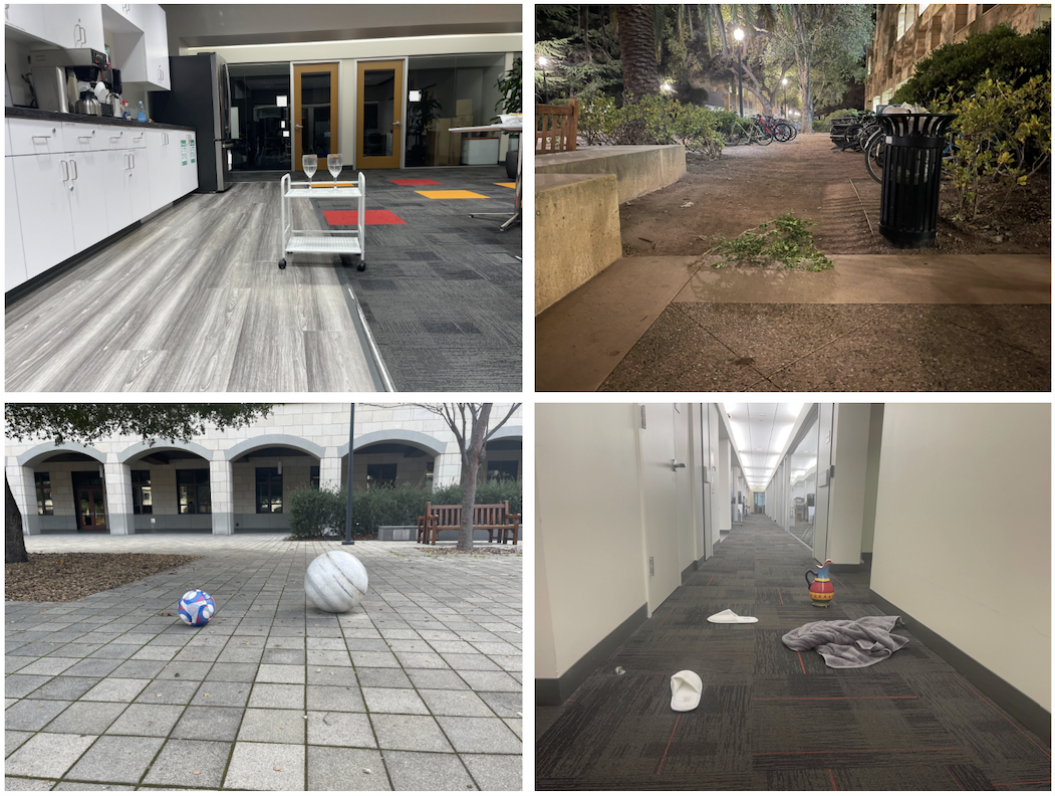


Figure 8: **Sample Images from the Affordance Reasoning Evaluation Set.** To evaluate open-world interactive navigation, we collect images spanning three types of diversity: objects, environments, and contextual variations.

The registered LiDAR points are projected to 2D and aggregated into an occupancy grid, which is used to compute the costmap. Table 4 lists the relevant parameters. We use the ROS2 Nav2 Navigation Stack [5] for path planning and follow the resulting plan with a holonomic controller without rotation, maintaining consistent camera views of objects identified during planning.

C Affordance Reasoning Experiment Details

C.1 Evaluation Set

To test affordance reasoning for *open-world* interactive navigation, we collect a dataset of 100 images covering a range of objects, environments, and contextual variations. Figure 8 shows sample images in the evaluation set. Table 5 summarizes the composition of the evaluation set. All affordance labels were annotated by an expert annotator familiar with the robot's physical capabilities. Table 6 lists examples of how the context variations of the same object type in the evaluation set affect its affordance. For example, a default rolling cart is interactive, but one with a "DO NOT TOUCH" sign is not.

Table 5: **Composition of the Affordance Reasoning Benchmark.** Our benchmark spans diverse environments, lighting, clutter, and obstacle sizes.

Attribute	Composition
Scene type	Indoor (69%); outdoor (31%)
Lighting	Bright artificial (66%); natural (28%); dim (6%)
Object affordance	(I) 94 / 217 (43.3%); (N) 123 / 217 (56.7%)
Clutter	1–4 foreground objects per scene
Obstacle size	Small electronics to large trees

Table 6: **Example Context Variations in Affordance Reasoning Evaluation Set.** The same object type is marked as **interactive (I)** or **non-interactive (N)** depending on context.

Object	Contexts
Rolling cart	Default (I); w/ glassware (N); w/ glasses (N); w/ stuffed toys (I); w/ toys + "DO NOT TOUCH" sign (N); w/ heavy robot arm (N); w/ snack boxes (I)
Wheeled crate	Default (I); w/ toaster box precariously on edge (N); w/ bundled wire on top (I); w/ cable attached (N)
Suitcase	Default (I); w/ caution tape (N)
Chair	Default (I); w/ glass vase on seat (N)
Spheres	Soccer ball (I); marble sphere (N)

C.2 Details for the `closed-set` Baseline [4]

The original approach in [4] relies on per-scene human instructions (e.g., "please go through the curtain and be careful of the chair."). As per-scene instructions are unavailable in our zero-shot evaluation setting, we instead generate a fixed list of 100 object categories (50 interactive, 50 non-interactive) using `gemini-3-flash-preview` [31] with `MessyNav`'s affordance reasoning prompt as context. This ensures the closed-set categories reflect the same affordance reasoning principles as our method, while still producing a fixed category-level list independent of scene-specific instructions. We also replace their traversable/non-traversable labels with interactive/non-interactive to match our task formulation.

The prompt is as follows:

```
Based on this context which has affordance instructions and grounding
examples: '{affordance reasoning prompt}', please make an exhaustive
prompt of 50 categories each based on common objects you may encounter
on the floor. (50 objects that you can interact with, 50 objects you
should avoid). Format it as a collection of sentences. For example -
Interact with items like empty wheeled chair, stuffed toy, .... Avoid
or be careful of items like heavy desk, ....
```

This results in the following prompt which is used by the `closed-set` baseline:

```
Interact with items like an empty wheeled chair, a stuffed toy, a small
cardboard box, a plastic water bottle, a lightweight laundry basket, a
tennis ball, a pair of sneakers, a discarded newspaper, a plastic food
container, a small throw pillow, a light wooden stool, a yoga block, a
slippers, a handheld remote control, a small plastic trash bin, a
folded towel, a soft-sided cooler bag, a frisbee, a light plastic
```

```
bucket, a rolling office footrest, a small backpack, a notebook, a
plastic toy truck, a beanbag chair, a light yoga mat, a cardboard
mailing tube, a small area rug, a plastic watering can, a light step
stool, a handheld umbrella, a soft lunchbox, a plastic storage bin lid,
a small pet bed, a light wooden crate, a foam roller, a plastic
dustpan, a small dry mop, a lightweight folding chair, a soft
briefcase, a plastic tray, a small cardboard tray, a light plastic
stool, a rolled-up poster, a small fabric ottoman, a plastic toy block,
a light wicker basket, a small duffel bag, a plastic clipboard, a light
broom, and a small empty flower pot.
Avoid or be careful of items like a heavy desk, a glass vase, a ceramic
floor lamp, a large bookshelf, a heavy filing cabinet, a television
set, a desktop computer tower, a large potted plant, a glass coffee
table, a heavy wooden dining table, a refrigerator, a microwave, a
space heater, a floor mirror, a fragile porcelain figurine, a stack of
heavy textbooks, a metal safe, a heavy toolbox, a piano, a large
aquarium, a marble pedestal, a heavy leather sofa, a treadmill, a
stationary bike, a large chest of drawers, a fragile glass sculpture, a
heavy radiator, a wall-mounted heater, a large speaker, a heavy metal
trunk, a delicate antique chair, a floor-standing clock, a heavy
printer, a wine rack with bottles, a fragile drying rack, a heavy
workbench, a large trash compactor, a heavy stone statue, a fragile
folding screen, a heavy air conditioner unit, a large dog crate, a
heavy weight bench, a stack of ceramic tiles, a large bag of concrete,
a heavy industrial fan, a fragile neon sign, a heavy metal shelving
unit, a large wooden wardrobe, a fragile model ship, and a heavy
cast-iron stove.
```

C.3 Details for the `closed-set-oracle` Baseline [4]

This represents the upper bound of the above baseline. It uses a manually curated list containing all object categories present in our evaluation set with their ground-truth category-level interaction labels. While no objects are missed, affordances remain fixed per category and do not account for context-dependent variations. The prompt for the affordance reasoning benchmark is as follows:

```
Interact with items like a suitcase, wheeled crate, soccer ball,
stuffed toy, vegetables, fruits, slippers, shoes, towel, napkin, trash
bag, clothing, small table, bundled jump rope, plastic stool, tongs,
microwave, pencil sharpener, waste basket, paper, small basket, small
chair, rolling chair, rolling cart, snack box, cardboard box, trash
bin, ball, beanie, wheeled chair, slipper, branch, tennis balls, empty
bag, plastic cups, markers, binders, car toy, pillow, wheeled table,
jeans, foldable stool, short wire, bag, toy blocks, toy trains, small
stool.

Avoid or be careful of items like a laptop, large table, vase, ceramic
dishes, glass item, electrical appliance, bundle of wires, scooter,
bike, bush, bag of leaves, stone statue, heavy dumbbell, pole,
newspaper stand, shovel, cinder block, laptop box, plants, robot arm,
unfurled wired cables, sofa, tree, bench, human, shop vacuum, picnic
table, electronics.
```

C.4 Additional Result Analysis

Table 7 breaks down affordance reasoning accuracy into Simple and Complex cases. Simple cases are objects in their default state (e.g., an empty cart, a chair with nothing on it), while Complex cases involve context-dependent configurations (e.g., a cart with electronics precariously placed on top, a suitcase with caution tape on top). `MessyNav` is competitive on simple cases and substantially outperforms all baselines on complex cases. The results highlight the gap between `MessyNav` and the strongest baseline, `closed-set-oracle`. Despite having access to privileged information about object categories in the evaluation set, `closed-set-oracle` achieves comparable performance

Table 7: **Affordance Reasoning Comparison**. Number of objects with correctly classified interactivity. **Simple** cases contain objects in their default state, while **Complex** cases involve context-dependent configurations (e.g., cart with fragile items, table with vase, suitcase with caution tape).

	Simple	Complex	Total
`closed-set`	30 / 112	23 / 105	53 / 217
`closed-set-oracle`	106 / 112	38 / 105	144 / 217
`height-only`	57 / 112	73 / 105	130 / 217
`MessyNav (ours)`	**101 / 112**	**85 / 105**	**186 / 217**

Table 8: **Real-World Navigation Scenarios.** To evaluate open-world interactive navigation, we test in a bedroom and conference room environments while deliberately varying objects, contexts, and required interactions across trials. The scenes also contain furniture (e.g., couch, bed, drawers, meeting tables) as fixed non-interactive obstacles. Additional objects are listed and marked as **interactive (I)** or **non-interactive (N)**.

Environment	Additional Objects in Scenario	Interaction Needed
Messy Bedroom	Cart w/ broken wheels (N), Cart (I)	Swipe cart
	Computer Box (N), Clear Box (I)	Swipe clear box
	Suitcase balancing books (N), Cart w/ one book (I)	Swipe cart
	Crate w/ one book (I), Cart w/ loose golf balls (N)	Swipe crate
	Small full box (N), Large clear box (I)	Swipe clear box
	Wheeled drawer w/ one toy block (I), Cart w/ toy castle (N)	Swipe wheeled drawer
	Small clear box w/ vase (N), Clear box w/ plush toy (I)	Swipe clear box
	Cart tangled in wire (N), Crate w/ stuffed toy (I)	Swipe crate
	Cart w/ clear box holding items (I), Crate w/ toy castle (N)	Swipe cart w/ clear box
	Cart w/ active fan (N), Cart w/ bundled wire (I)	Swipe cart w/ wire
Conference Room	Cart w/ laptop (N), Crate w/ pitcher in clear box (I)	Swipe crate
	Cart w/ items in box (I), Cart w/ glass of water (N)	Swipe cart w/ box
	Suitcase w/ caution tape (N), Suitcase (I)	Swipe suitcase
	Suitcase w/ bag (I), Cart w/ loose water bottles in bag (N)	Swipe suitcase
	Suitcase w/ caution tape (N)	None

only on simple cases. Its performance degrades significantly on complex scenes as it is unable to reason beyond its default context. In contrast, `MessyNav` requires no predefined object categories and can reason about the object's broader context.

D Details on Real-World Comparative Experiments

D.1 Setup

The navigation goal is specified as a world-frame coordinate in the task instruction: `"Navigate to (goal_x, goal_y) and handle any obstacles in your path."` We mark environment boundaries beyond the navigable area as occupied in the occupancy map. To evaluate open-world interactive navigation, we test across 15 scenarios in two environments that vary in objects, contexts, and required interactions. For example, the same cart is empty in one scenario and holds a glass of water in another. Table 8 lists the full set of scenarios.

D.2 Details for the `closed-set-oracle` Baseline [4]

The prompt for real-world comparative experiments is as follows:

```
Interact with items like a cart, stuffed toy, storage bin, cardboard
box, toys, stuffed toys, toy blocks, suitcase, plastic bin, tissue,
paper bags, books, golf ball, tennis ball, tennis ball canister, plate,
wheeled crate, clothes, stool, chair, water bottles, bottles, and
bowls. Avoid or be careful of items like wires, electrical appliances,
drawer, bed, laptop, glass items, pitcher, heavy furniture and vase.
```

Subsequently, the results are passed to the VLM planner used in `MessyNav`.

D.3 Details for the `vlm-planner-only` Baseline

The prompt is as follows:

```
You are a robot task planner for a mobile robot with one arm and a
2-fingered gripper. The robot constraints:
- It can only carry about 3kg in your gripper at a time
- The gripper can open up to 8.5 cm wide
- The footprint is a 0.8m x 0.8m square

Analyze the image and plan your actions. LOOK AT THE IMAGE CAREFULLY to
assess FREE SPACE and use this information to inform your planning.
The image is the front-facing view from the robot's perspective.

TASK: {task_description}

AVAILABLE MOTION PRIMITIVES:
- move_to_nav2(x, y): Move to position (x, y) in meters
- move_to_nav2(object_name, dx, dy): Move to object with offset (dx, dy
in meters)
- pick_up_entity(object_name): Pick up the named object
- swipe_left(object_name): Swipe object to the left
- swipe_right(object_name): Swipe object to the right
When interacting with objects, choose action based on efficiency and
context:
- Pick: Objects that can be easily grasped and carried by the robot
- Swipe: Larger objects - quick lateral clearing motion.
Note: You can NOT swipe objects below 0.3 meters in size.

NAVIGATION OFFSETS:
- Robot Origin: The robot starts at (0.0, 0.0) facing in the +x
direction
- Coordinate System:
    - +x: Forward movement (in front of the robot)
    - -x: Backward movement (behind the robot)
    - +y: Leftward movement (to the robot's left)
    - -y: Rightward movement (to the robot's right)
- dx: Forward/Backward offset in meters (positive = forward)
- dy: Left/Right offset in meters (positive = left)
- Want to be approximately 0.6m away from objects for manipulation
- Example: move_to_nav2(obstacle, -0.6, 0.0) -> position 0.6m distance
from the obstacle (approaching it, stopping short)

PLANNING GUIDELINES:
- For movable objects, YOU decide pick vs swipe based on apparent
size/weight
- Position ~0.6m away from objects for manipulation
- You may not need to explicitly interact with all objects - only those
that necessarily blocking the path you want to take.
- move_to_nav2 primitive calls a navigation planner that avoids
obstacles
- IMPORTANT: move_to_nav2 finds navigation sub goals so there is DO NOT
give intermediate waypoints

GROUNDING EXAMPLES:
- Empty wheeled chair -> swipe_right(chair)
- Stuffed toy -> pick_up_entity(stuffed_toy)
- heavy desk -> avoid

Respond with ONLY a JSON object in this exact format:
{{
"reasoning": "Explain your planning strategy based on visual analysis",
```

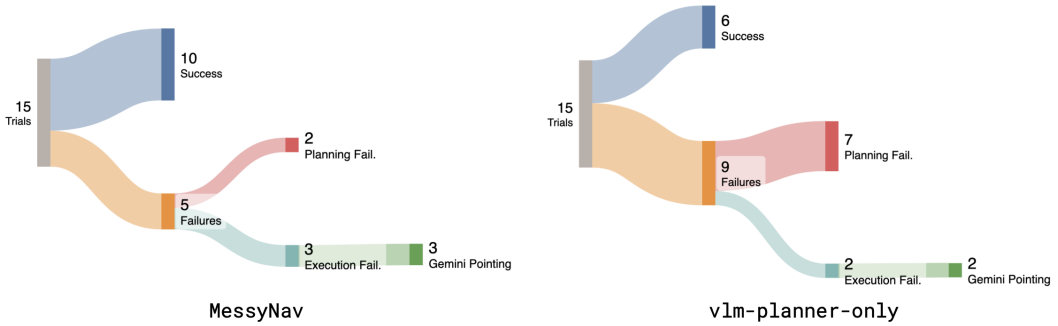


Figure 9: **`MessyNav`'s failure breakdown compared to the strongest baseline.** Sankey diagrams of trial outcomes for `MessyNav` (left) and `vlm-planner-only` (right) over 15 real-world trials. `MessyNav` succeeds on 10/15 trials versus 6/15 for `vlm-planner-only`. While most of `vlm-planner-only`'s failures are planning failures in which `vlm-planner-only` decides to interact with non-interactive objects (7 of 9), `MessyNav` reduces these to 2, showcasing the importance of explicit affordance reasoning.

```
"geometric_analysis": "Analyze spatial constraints and navigation
paths",
"plan": [
    "move_to_nav2(x, y)",
    "action_3"
]
}}
```

D.4 Failure Mode Analysis

Figure 9 compares failure modes between `MessyNav` and the strongest baseline, `vlm-planner-only`. Overall, `MessyNav` outperforms `vlm-planner-only`, particularly in planning success. As `vlm-planner-only` lacks explicit affordance reasoning, it incurs 7 planning failures rather than 2. Most of `MessyNav`'s remaining failures instead stem from incorrect Gemini pointing (e.g., pointing to free space within a cart, resulting in failed execution). Its two planning failures arise from subtle affordance-reasoning errors, such as failing to distinguish a cart with a missing small wheel from an intact one.

E Details on the Offline Planning Evaluation

E.1 Scene Set

In addition to the 15 scenes used in the real-world experiments (Table 8), we include 40 additional scenes from a kitchen, an outdoor daytime setting, and an outdoor nighttime setting, for a total of 55 scenes across five environments. Each scene contains at least one object blocking the direct path to the goal. We vary object contexts so that deciding whether and how to interact requires reasoning beyond object category.

E.2 Protocol

For each scene, both methods receive the same egocentric RGB observation and goal specification and generate a high-level plan. A human evaluator assesses each plan under the assumption of perfect low-level execution. A plan is successful if it would reach the goal without interacting with any object labeled non-interactive in the ground truth. This isolates high-level planning performance from low-level execution failures. We conduct 15 runs per method, where each run evaluates all 55 scenes once, for a total of 825 scene-level evaluations per method.