Agentic Scene Policies:
Unifying Space, Semantics, and Affordances for Robot Action

Sacha Morin[1,2], Kumaraditya Gupta[1,2], Mahtab Sandhu[1,2], Charlie Gauthier[1,2],
Francesco Argenziano[3], Kirsty Ellis[1,2], Liam Paull[1,2]
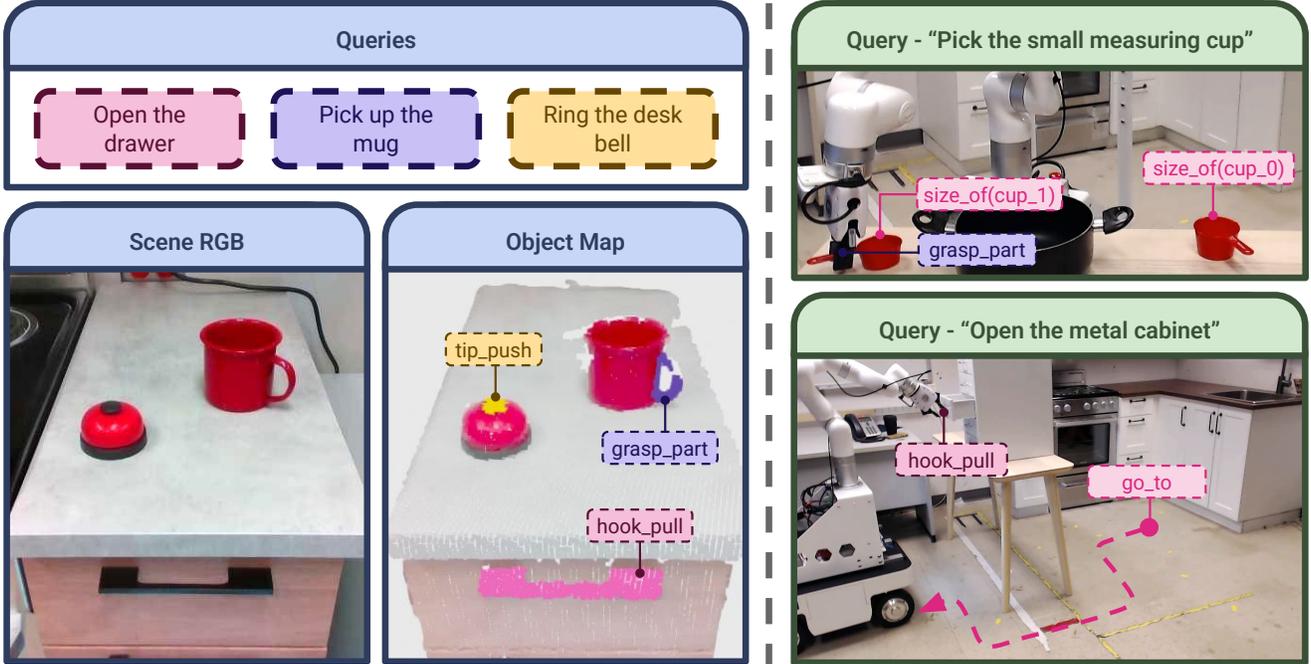[1]Université de Montréal, [2]Mila - Quebec AI Institute, [3]Sapienza University of Rome

Fig. 1: *ASP* implements a language-conditioned robot policy. Leveraging foundation models for open-vocabulary perception and affordance detection, we design a general scene representation that supports a compact and expressive set of tools for an LLM agent to fulfill tabletop and mobile manipulation queries given through natural language. The agent can handle open-vocabulary queries (e.g., `grab the panda plushie`) and reason about various relational spatial concepts (e.g., `larger/smaller`, `nearest/farthest`). Actions are carried out through interactions with affordances (e.g., `handle`, `button`) and corresponding skills (e.g., `grasp_part`, `tip_push`). (Webpage)

Abstract— **Executing open-ended natural language queries is a core problem in robotics. While recent advances in imitation learning and vision-language-actions models (VLAs) have enabled promising end-to-end policies, these models struggle when faced with complex instructions and new scenes. An alternative is to design an explicit scene representation as a queryable interface between the robot and the world, using query results to guide downstream motion planning. In this work, we present Agentic Scene Policies (*ASP*), an agentic framework that leverages the advanced semantic, spatial, and affordance-based querying capabilities of modern scene representations to implement a capable language-conditioned robot policy. *ASP* can execute open-vocabulary queries in a zero-shot manner by explicitly reasoning about object affordances in the case of more complex skills. Through extensive experiments, we compare *ASP* with VLAs on tabletop manipulation problems and showcase how *ASP* can tackle room-level queries through affordance-guided navigation and a scaled-up scene representation. We encourage readers to visit our project page.**

I. INTRODUCTION

Generalist language-conditioned robot policies need to manage the complex interplay between language, space,

and action. Much of the recent progress on this problem has been driven by vision-language models (VLMs) trained on internet-scale data and showing strong general visual understanding in the open-world. Applying VLMs to the robotics domain has broadly followed two paradigms. In the first paradigm, VLMs can serve as backbones for end-to-end policy learning, yielding "vision-language actions" models (VLAs) that directly map sensor data and language commands to robot actions [1]–[5]. In the second paradigm, VLMs are primarily used for perception in the construction and querying of structured scene representations with advanced capabilities for object retrieval and spatial reasoning [6]–[15].

VLAs are increasingly showing zero-shot potential on new tasks [4], [16] but in practice still require task-specific fine-tuning to be truly proficient, which poses challenges in terms of data collection and infrastructure that limit overall deployment. For their part, scene representations preserve the generality of VLMs—they can practically represent any object—but do not offer a direct solution to the motion

problem and are often constrained to navigation and pick-and-place tasks as a result [6], [17], [18].

We observe that a large number of language queries can be solved through a (potentially repeated) three-step process consisting of 1) object grounding, 2) spatial reasoning, and 3) part-level interaction. In this work, we demonstrate that state-of-the-art zero-shot performance can be achieved across a wide range of robotics tasks by implementing all three steps as scene queries. We expose querying functionalities as tools that can be freely called by a large language model (LLM) agent to execute language commands. For interaction, we design an expressive set of skill primitives supported by the strong affordance detection capabilities of VLMs. Our modular policy can map language queries (`Ring the desk bell`, `Remove the thumbtack`) to specific affordances and affordance-based skills (`tip_push`, `pinch_pull`), as well as solve a range of mobile manipulation queries. In summary, our key contributions include:

1) **A**gentic **S**cene **P**olicies (*ASP*), a language-conditioned manipulation policy that can solve a broad range of queries involving specific semantics, spatial reasoning, and affordances.
2) An extensive empirical comparison with leading VLAs on 15 manipulation tasks, providing a valuable data point in the ongoing debate between modular and end-to-end methods.
3) A mobile version of *ASP* that tackles room-level queries through affordance-guided navigation and a scaled-up scene representation.

II. RELATED WORK

Open-Vocabulary Mapping. Open-vocabulary maps replace conventional class labels with multi-modal features from foundation models such as CLIP [6]–[13], allowing a much richer representation of semantics. For mapping, features are typically extracted from vision sensors, grounded to a map element (point, voxel, object), and aggregated across views. At inference time, they are compared with query features, enabling specific queries about objects and their properties. Open-vocabulary maps underpin recent queryable systems for navigation [19] and mobile manipulation [17], [20].

Affordance Detection. Affordance detection aims to segment functional elements of objects, such as handles, buttons, or knobs, that enable specific interactions [21]–[25]. They have the potential to greatly simplify the implementation of robot skills by providing direct cues about the geometry that a robot can manipulate. A number of recent methods has focused on leveraging VLMs for this task [26]–[29] given their increasingly good performance at detecting parts and pointing at images [30], [31]. In the RGB-D setting, they can be prompted zero-shot to detect affordances on a range of objects and combined with class-agnostic segmentation [32], [33] for segmentation. The resulting segment can be lifted to 3D using depth data.

LLM Agents for Robotics. Controlling robots via language is a long-standing problem in robotics [34]. Taking advantage of the coding abilities of LLMs, recent work has framed language-conditioned robot policies as a translation problem between natural language and code, effectively mapping user queries to robot API calls or LLM-generated function calls [35], [36]. This has been followed by more reactive agentic approaches where LLMs interleave text generation and function calls instead of coding entire plans upfront [37], [38], enabling a tighter feedback loop with the API and existing frameworks such as ROS [39]. Agents can also chain tool calls to answer grounding queries in 3D scenes [15].

End-to-End Learning. An alternative to controlling a robot via a predefined API or tools is to directly learn a function mapping vision and language to robot actions. Such end-to-end policies have been transformed by the emergence of VLMs that enable the transfer of web-scale knowledge to robot control. Examples include CLIPort [40], which leverages the general semantics of CLIP for imitation learning, and the more recent VLAs [1]–[5]. VLAs leverage pretrained VLMs and imitation learning to map vision and language inputs to a shared representation that can be decoded into robot actions. While enabling capable policies with zero-shot potential, VLAs still face challenges in terms of generalization to new environments and complex language understanding, often requiring some amount of fine-tuning on specific problems to perform well [41].

III. METHOD

A wide range of language queries can be solved through object grounding, spatial reasoning, and affordance-level interactions. *ASP* is designed around a compact set of scene querying tools implementing these functionalities. Tools are functions that can be called by an LLM agent, and their outputs are fed back to the agent to inform the next tool call during execution (Fig. 2).

This section contains high-level definitions of data structures and tools. We reserve the specific implementation details for Section IV. To simplify exposition, we focus on the tabletop setting first and discuss the mobile setting in Section III-D.

A. Object Map

When the agent needs to perceive the environment, we build a 3D scene representation of the workspace. The `ObjectMap` is a list of `Objects` which itself includes a list of `Affordances`:

```
struct Affordance:
    point_cloud: PointCloud
    part: str
    skill: Skill


struct Object:
    point_cloud: PointCloud
    rgb_crops: List[RGBImage]
    depth_crops: List[DepthImage]
    features: Vector
    affordances: List[Affordance]
```
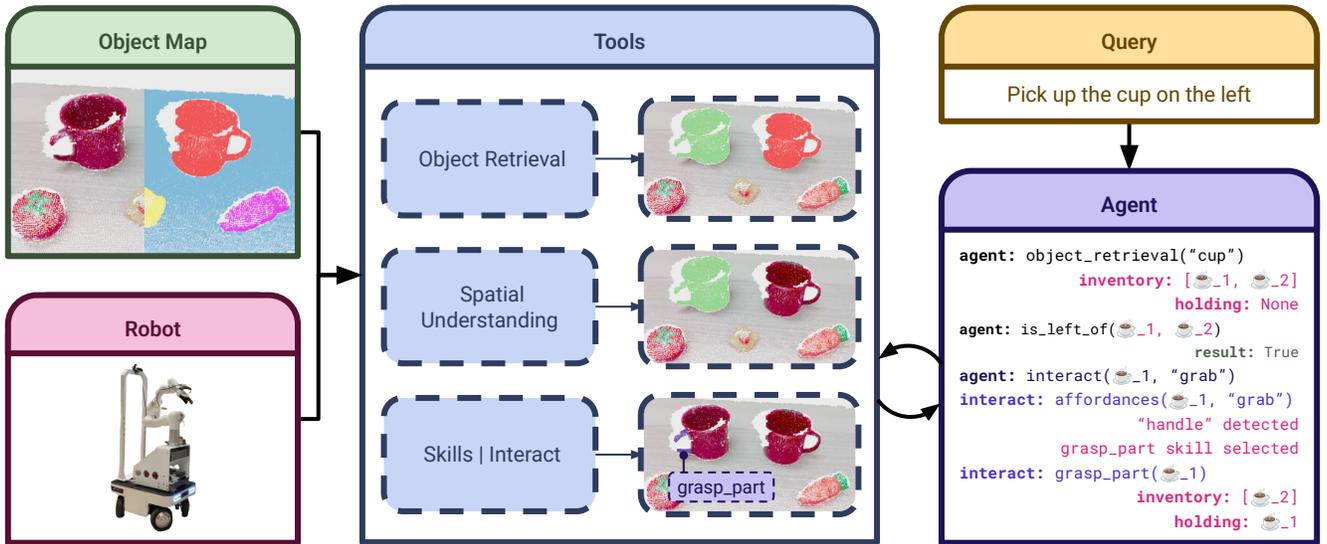
Fig. 2: **Framework.** *ASP* is an agentic framework for language-conditioned tabletop and mobile manipulation. **(Right)** An LLM agent breaks down the user query into a sequence of tool calls. Available tools include (i) open-vocabulary object retrieval, (ii) spatial understanding tools to measure sizes, distances, and other spatial predicates, and (iii) a general interact tool that manages skill tools informed by fine-grained 3D affordances. Tools may return basic data types (e.g., `float` for a distance), or a symbolic state encoding the currently held object (if any) and an inventory of the query-relevant objects that the agent has grounded so far. We use emoticons for illustrative purposes: the agent actually sees string symbols (e.g., `cup_0`). **(Left)** To achieve a truly flexible language-guided policy, LLM agents can be paired with a general scene representation. We build an open-vocabulary object map following [6], [11] and encode object semantics as language-aligned CLIP features. We further detect affordances with foundation models and represent them using 3D point clouds, part descriptions, and corresponding skills. The map plays a central role in the tool implementations.

`Object` encapsulates key perceptual elements that will be required in tool implementations. It describes different facets of an object such as:

- **Geometry**: the `point_cloud` in the scene frame.
- **Semantics**: the `features` are extracted and aggregated from chosen local object views in `rgb_crops` and form a shared vision-language feature space (e.g., CLIP [14]) to enable comparisons with open-ended text queries. We also store the `depth_crops` corresponding to `rgb_crops`.
- **Affordances**: affordances described in terms of their geometry (`point_cloud`), a natural language description of the relevant object part (`part`) and a `skill` that corresponds to an available skill tool on the robot (Section III-C). We store a list of `Affordances` to account for the possibility of multiple affordances on complex objects (e.g., microwave).

In practice, we build an `ObjectMap` first and populate the `Affordances` at inference time given the agent tool calls.

B. LLM Agent

The core decision-making module is an LLM agent which parses the user query and calls a sequence of tools (Fig. 2). The *ASP* agent is not directly exposed to sensor data or the `ObjectMap`. Instead, it perceives the environment through a symbolic state representation:

```
struct State:
    held_object: ObjectKey | None
    inventory: List[ObjectKey]
```

that is returned when calling certain tools (Section III-C) and describes the currently held object (`held_object`) and objects that the agent can reason on, or act on using other tools (`inventory`). An `ObjectKey` is a simple unique string identifier that is initially generated by the `object_retrieval` tool (Section III-C) based on the retrieval query and the number of relevant objects (e.g., `red_ball_0`).

C. Tools

All tools have access to the `ObjectMap` and the current `State` to implement their functionalities. The agent can add objects to its `State` inventory using the **object retrieval** tool, can reason about them using **spatial understanding** tools, and can act on them with the **interact** tool.

Object Retrieval. The object retrieval tool allows the agent to retrieve objects from the `ObjectMap` using an open-vocabulary text query:

```
function object_retrieval(
    query: str
) -> (current_state: State)
```

and adds the relevant `ObjectKeys` to `current_state.inventory`.

The *ASP* agent is prompted to be specific when looking for objects and will break down user queries into multiple retrieval calls (Fig. 2, right).

Spatial Understanding. Spatial understanding tools have the signature:

```
function spatial(
```

```
    objects: List[ObjectKey]
) -> (output: float | bool)
```

and allow the agent to measure specific quantities (e.g., distances, sizes) or verify pairwise spatial predicates (e.g., "is left of") in the current inventory.

Interact Tool. The agent can interact with inventory objects using a natural language description of the intended `action` (e.g., grab, unplug):

```
function interact(
    obj: ObjectKey, action: str
) -> (current_state: State)
```

Internally, `interact` runs an affordance detection workflow (Section IV-A) to segment the relevant `Affordance` corresponding to the `action` description (when needed), and then calls the corresponding `skill` tool:

```
function skill(
    obj: ObjectKey
) -> (current_state: State)
```

Skills are responsible for providing an updated `State`. For example, a successful pick will move the relevant `ObjectKey` from `inventory` to `held_object`.

Tool Preconditions and Feedback. Some tools require preconditions to be met, such as `held_object = None` when trying to pick an object or `obj in current_state.inventory` when calling the spatial and interact tools. Such conditions are explicitly detailed in the tool prompts and verified in the implementations. Moreover, we wrap the output of each tool in a parsable data structure:

```
struct ToolOutput:
    success: bool
    feedback_msg: str
    output: State | float | bool
```

to provide explicit feedback to the agent. `feedback_msg` can detail reasons for failures, such as unfulfilled preconditions or motion planning failures. The agent can leverage feedback to reattempt tool calls, increasing the robustness of the overall policy.

Remapping. Tools govern when the `ObjectMap` is recomputed. Calls to skill tools may trigger remapping when the location of objects becomes unknown (e.g., after failing a grasp), whereas consecutive calls to the object retrieval and spatial understanding tools will reuse the same `ObjectMap`. Whenever a tool internally raises a map update, we clear the agent inventory and recompute the map on the next call to `object_retrieval` by positioning the arm at a default home pose and computing an `ObjectMap` based on the current RGB-D.

D. Mobile ASP

Mobile *ASP* follows the previously introduced framework with an additional tool for navigation:

```
function go_to(
    obj: ObjectKey, action: str
) -> (current_state: State)
```

Similar to `interact`, the agent can specify an `action` description to detect an `Affordance` on the target object. Knowing the affordance orientation will help navigate to a pose that is suitable for interaction.

Mobile *ASP* also includes the following changes: (i) `ObjectMap` aggregates information (Section IV-A) from multiple posed RGB-D keyframes taken across a room. (ii) Following a successful `go_to` call, we build a separate local `ObjectMap` from the current camera frame and retrieve the sought object using the same query that was used to ground the object in the main `ObjectMap`. The agent is then free to apply tabletop skill tools to this local object representation. This "redetection" renders the system more robust to possible localization and mapping errors in the main `ObjectMap`. (iii) We do not allow tools to update `ObjectMap` (**Remapping**), although in principle this could be achieved by revisiting keyframe locations.

IV. IMPLEMENTATION DETAILS

A. Object Map

Open-Vocabulary Object Map. We build the `ObjectMap` following recent work in open-vocabulary object-centric map representations [6], [11]. Whenever we recompute the map in the tabletop setting, we position the arm at home pose to have a good overview of the workspace and process the RGB-D wrist frame. We first run class-agnostic segmentation (MobileSAM [33] in grid-sampling mode) to extract segmentation masks and convert them to bounding boxes. For each mask, we crop a local RGB image and embed it with CLIP to describe semantics. We also backproject the masks using the camera depth to obtain 3D point clouds.

Object Merging. The initial step yields a first set of `Objects` (with empty affordances). We then follow the merging strategy of [6] to merge different `Objects` with similar geometries (point cloud overlap) and semantics (CLIP similarities). This merging step is useful even when processing a single frame to mitigate the over-segmentation of complex objects by SAM.

When merging `Objects`, we accumulate and downsample their `point_cloud`, maintain a running average of the `features`, and combine the `rgb_crops` and `depth_crops`. We sort the crops by segment area (number of pixels in the segment), with a penalty if the segment touches the image border to favor larger crops where the object is central.

Mobile ASP. In the mobile setting, we incrementally build the `ObjectMap` and merge objects across keyframes, identical to [6]. We do not target exploration in this work and collect a few keyframes through teleoperation before deploying the policy. This mapping phase takes less than one minute per query.

Affordance Detection. To detect `Affordances` for an `Object` in the `ObjectMap`, we design a 2-step pipeline leveraging VLMs. First, we use Gemini 2.5 to predict a list of `skills` and corresponding `parts` given the top k best `rgb_crops` of the `Object` and the current `action`

description. Each predicted (`skill`, `part`) pair for each crop defines a distinct `Affordance`. Second, we prompt Gemini 2.5 again with the `part` and `rgb_crop` to produce a bounding box for the image crop, which is used to get a 2D mask using an image segmentation model (SAM 2.1). The mask is then lifted into 3D with the corresponding `depth_crop`, yielding the `point_cloud` representation of the `Affordance`. To avoid duplicate affordances for an `Object`, we run multi-view association by checking the IoU of all affordances and merge them if it exceeds a threshold.

B. Agent

Agent. We implement the agent using LangChain [42] with the package's default prompts and a Gemini backend [30].

C. Tools

Object Retrieval. We implement `object_retrieval` as a search for the top k similar objects in `ObjectMap` based on the similarity between the `query` CLIP features and the object `features` in the map. We then use a VLM to confirm whether each of the top k objects are relevant to the `query` or not using the best object views [20]. In contrast to using a pure CLIP-based retrieval approach, this VLM classification step returns a variable number of relevant object without tuning a specific CLIP similarity threshold. In our experiments, we use $k = 3$ and Gemini 2.5 as the VLM classifier.

Spatial Tools. We expose `distance_to`, `distance_between`, `left_of`, `right_of`, and `size_of`. Spatial understanding tools are implemented as basic operations on the object point clouds and their centroids.

Interact Tool. `interact` runs the affordance detection pipeline to infer the appropriate `skill` given the object and the `action` description. This design allows skill selection to be informed by vision and influenced by the existence of a part (e.g., not all cups have handles) and their shapes (e.g., to try hooking or pulling depending on the handle).

We implement all `skills` using motion planning. Specifically, we expose the general object skills:

- `grasp(obj)`: grasp `obj` anywhere.
- `place(obj)`: place `held_object` on `obj`.
- `drop(obj)`: drop `held_object` on `obj`.

as well as some affordance-based skills, many of which are inspired by the SceneFun3D affordance types [23]:

- `grasp_part(obj)`: grasp a specific object part.
- `tip_push(obj)`: push on a specific object part with the tip of the gripper.
- `pinch_pull(obj)`: pinch the part with the gripper and pull.
- `hook_pull(obj)`: hook the part from above and pull.

Skill implementations generate scripted end-effector pose goals for the motion planner using a combination of operations on the object point cloud (e.g., finding the normal or a point above it) and AnyGrasp [43]. In the case of

part-level skills with affordances, we derive goals using the point cloud stored in `Affordance`. We always run AnyGrasp on a context around the object and only keep the grasps that are near the object point cloud (for `grasp`) or the specific affordance point cloud (for `grasp_part`). For the two grasping skills, we iterate over the top 10 grasps until we find one where the motion planner succeeds (and return an error message otherwise). In the case of `pinch_pull` and `hook_pull`, we use a predefined end-effector rotation to grasp the affordance centroid and infer a horizontal pulling axis using the point cloud normal. For `grasp`, `grasp_part` and `pinch_pull`, we explicitly verify that the gripper holds something after execution and inform the agent of a failure if not.

Go To Tool. `go_to` runs the affordance detection pipeline on the remote object and infers a normal from the `Affordance` point cloud (if any) oriented away from the object centroid. We project this normal on the horizontal plane to infer a preferred viewing position $\mathbf{p}_{aff} \in \mathbb{R}^2$ at a distance of r from the object centroid.

While pre-collecting keyframes in the mobile setting, we also build a 2D occupancy map using the navigation stack. For the exact navigation goal, we find a target pose that is in free space and close to $\mathbf{p}_{aff}$. Specifically, we search for candidate poses $(\mathbf{p} \in \mathbb{R}^2, \theta \in [0, 2\pi))$ at discrete intervals on the circle of radius r centered at the object centroid by minimizing $\mathcal{F}(\mathbf{p}, \theta) + \lambda_{aff}\|\mathbf{p} - \mathbf{p_{aff}}\|_2$ with $\mathcal{F}$ denoting the average footprint cost in the occupancy map, and λ_{aff} being a penalty factor. We always choose candidate θ to face the object and discard candidate poses with high footprint cost (collisions). We start with a small r (0.85m) and progressively relax it if no valid pose is found.

D. Robot

Hardware. Our mobile manipulator consists of a UFactory XArm6 mounted on an Agilex Ranger Mini 2.0, similar to the builds in [44], [45]. We mounted an Intel Realsense D435i on the arm wrist and an Intel Realsense T265 tracking camera on the base. We stream RGB-D data over wifi and run all perception models on a workstation with an NVIDIA Titan RTX.

Software. The robot software is integrated with ROS 2. We use MoveIt 2 [46] for the arm motion planning (RRT-Connect Planner [47]). SLAM and navigation are handled by RTABMap [48] and Nav2 [49] using the wrist RGB-D camera and odometry estimates from the T265 and the base.

V. EXPERIMENTS

We design our experiments around the following questions:

Q1. How does *ASP* compare to state-of-the-art VLAs on zero-shot manipulation tasks?

Q2. How important is affordance detection to policy success?

Q3. How does *ASP* scale to room-level queries involving navigation?

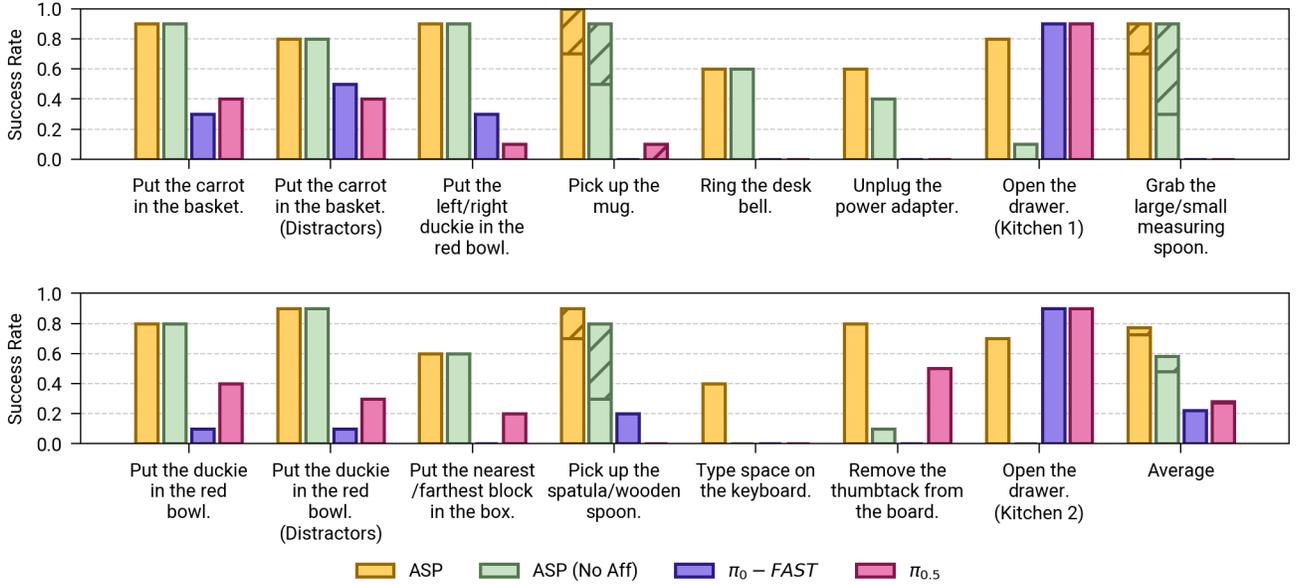**Q4.** When and why does *ASP* fail?

Fig. 3: **Tabletop Manipulation Results.** We run a total of 540 trials to compare *ASP* with an affordance-free ablation, *ASP* (No Aff), and two VLA baselines on 15 tabletop manipulation queries involving a variety of objects, geometric concepts (left/right, large/small), and affordances. **(Success Rate)** We report the average success rate over 10 attempts for each task. **(Hatch Pattern)** For objects with handles, we use a hatch pattern for episodes where methods succeed without using the handle or by using the handle in an unnatural way according to human judgment. **(Progress Rate)** While we focus on hard success rate in this figure, we also provide some task progression numbers in Fig. 7.
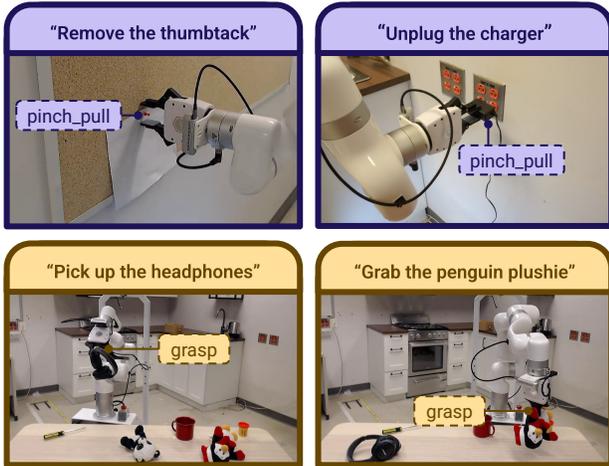

Fig. 4: Examples of manipulation queries. **(Top)** *ASP* can detect the `pinch_pull` affordance and deploy the corresponding skill in different situations involving different verbs (`Remove`, `Unplug`), and objects. **(Bottom)** Examples of double pick queries in the mobile setting involving a variety of objects, including headphones and a penguin plushie.

A. Tabletop Manipulation

Baselines. To answer **Q1** and **Q2**, we consider two baselines and an ablation:

- π_0-FAST [3]: An autoregressive VLA that can perform a wide-range of manipulation tasks in new scenes. We run π_0-FAST for a maximum of 800 timesteps per query.

- $\pi_{0.5}$ [4]: A flow-matching VLA co-trained on heterogeneous data with better open-world generalization. Given the faster inference, we run $\pi_{0.5}$ for a maximum of 1,500 timesteps per query.

- *ASP* (No Aff): an ablative baseline where `Affordance.point_cloud` is replaced with a point cloud of the entire object. All other skills are available. This baseline measures how fine-grained affordance segmentation impacts policy success. We only run this baseline on the 9 queries involving affordances and otherwise report the *ASP* numbers.

For π_0-FAST and $\pi_{0.5}$, we use the `openpi` checkpoints fine-tuned on DROID [50]. We chose to use a Franka arm and the DROID setup to minimize the risk of any distribution shift with the XArm6 (not in DROID). While π_0 does not leverage depth data, it does use two third-person cameras in addition to the wrist camera. Overall, we expect both arms to be equally capable on the considered tasks.

The *ASP* agents may reattempt some skills on failure due to tool feedback. We cap the number of retries at three.

Results. We report manipulation success rate for 15 queries in Fig. 3 and show some queries in Fig. 4. *ASP* outperforms π_0-FAST and $\pi_{0.5}$ on 13 of the 15 queries, showing strong language understanding and manipulation skills. The π models perform well on the drawer opening tasks but fail on comparatively simpler picking tasks showing an overall success rate ($\sim$20%). This is in line with a recent study of π_0-FAST in the zero-shot setting [16]. We did observe increased success with $\pi_{0.5}$ on the duckie queries and the more complex thumbtack query. In a high number
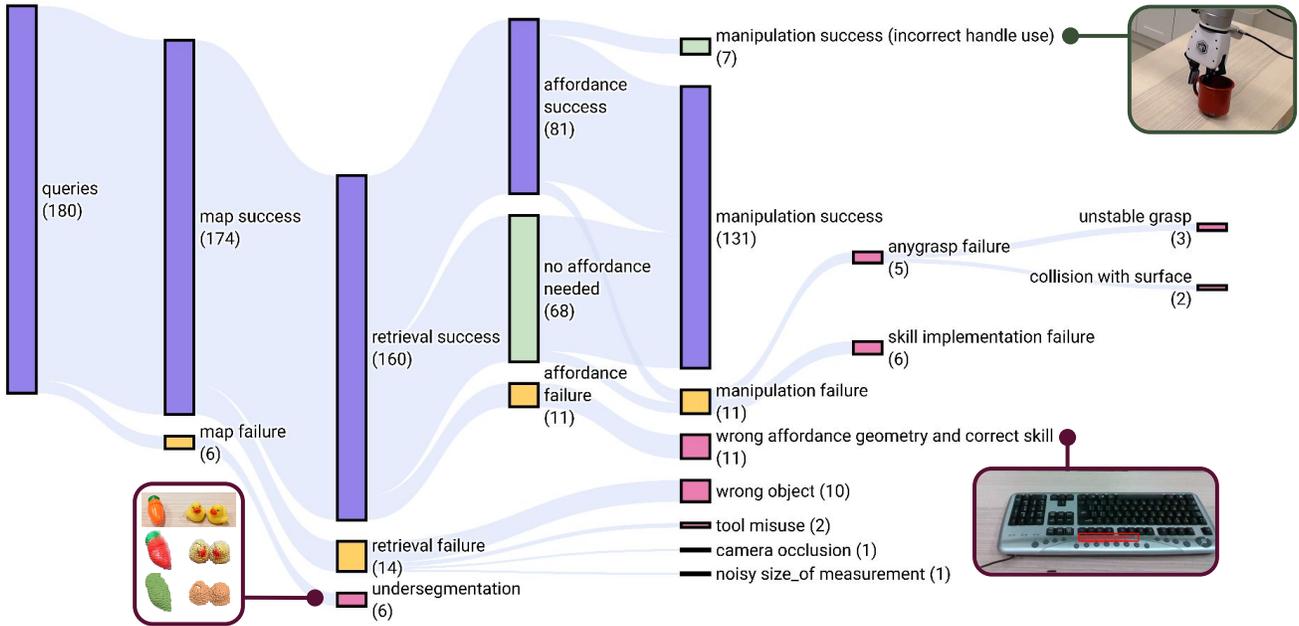
Fig. 5: **Failure Analysis.** The modular nature of *ASP* enables an in-depth analysis of failure modes for manipulation and mobile manipulation queries. We find that a high number of failures can be attributed to perception (31 queries), highlighting potential areas of improvements in the current stack. **(Undersegmentation)** Our SAM-only approach requires tuning some hyperparameters [6] when merging objects in the `ObjectMap`. Finding a set of hyperparameters that is optimal across queries and robust to depth noise is challenging. **(Wrong Affordance Detection.)** While our affordance detection workflow is adept at mapping query verbs to the correct skill (e.g., `Type` to `tip_push`), the detection stage is imperfect. For example, we find that Gemini can be influenced by the "canonical" location of a space bar when faced with an upside down keyboard. **(Incorrect Handle Use)** An example of a "hatched" success in Figure 3. While the handle was correctly segmented and *ASP* successfully picked the mug, the grasp does not correspond to a typical handle use, showcasing the limits of segmentation-driven grasp selection. **(VLAs)** We provide a similar plot for π_0-`FAST` and $\pi_{0.5}$ in Fig. 8.

of cases, we find that π_0-`FAST` (67%) and $\pi_{0.5}$ (75%) show some level of progression towards the task (Fig. 7 and Fig. 8).

Our ablative baseline shows that affordance detection plays a key role when the tasks go beyond pick-and-place, with *ASP* outperforming *ASP* (No Aff) on the keyboard, power adapter, thumbtack, and drawer queries. Affordances also guide manipulation towards more natural handle usage, although some unnatural grasps do still occur (Fig. 5, top right).

B. Mobile Manipulation

Baseline. To answer **Q2** and **Q3**, we run mobile *ASP* and the *ASP* (No Aff) ablative baseline, which this time has access to affordances for manipulation but not navigation: we set $\lambda_{aff} = 0$ and skip affordance detection in `go_to`.

Results. We assess the performance of mobile *ASP* in Figure 6. We find that the agent successfully interleaves navigation and manipulation to solve room-level problems, showing how the underlying scene representation can support decision-masking beyond the tabletop setting. The double pick queries show some degree of planning proficiency while the spatial queries demonstrate the role of the `ObjectMap` in understanding referential queries. When object affordances face a particular direction, incorporating affordance information at the navigation stage proves critical to query success.

C. Failure Analysis

To answer **Q4**, we explore and discuss the *ASP* failure modes in Fig. 5. Perception is behind a number of failures. While a similar analysis for the end-to-end baselines is harder, we classify and discuss some trials in Fig. 8.

VI. DISCUSSION

Strengths. We find that our scene query tools span a wide set of language-guided manipulation behaviors. *ASP* exhibits strong semantic generalization (driven by VLMs), and we expect reported performance to carry over to a wide variety of objects. Moreover, the `ObjectMap` supports robust spatial reasoning, and when combined with search-based motion planning, ensures the policy can solve problems across extended and varied spatial layouts.

Limitations and Future Work. The *ASP* design has been optimized for short-term language-conditioned manipulation tasks. Despite already having some planning capabilities, *ASP* would benefit from improvements on both the scene representation front (dynamic memory [20], [45], hierachical memory [10], exploration [18], [51]) and the task planning front [52] to tackle long-horizon problems.

Our skills only cover drawers with prismatic joints but they could be extended to revolute joints by following recent modular work [53]. More fundamentally, affordance-based skill implementations do not provide a clear avenue for solving long-horizon problems with complex motions,
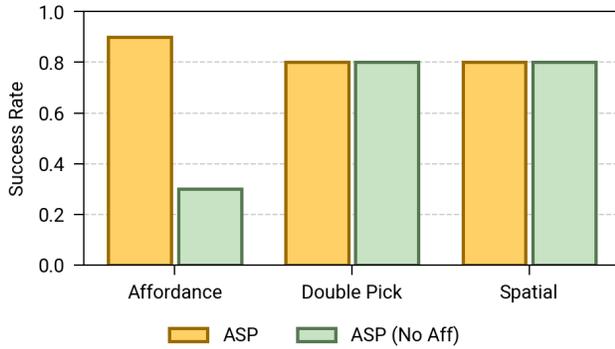
Fig. 6: **Mobile Manipulation Results.** We run mobile *ASP* on queries in a medium-sized room. For each query, we teleop the robot to collect 1 to 5 keyframes from the room before launching the policy. **(Affordance)** We run two queries: `Dial a number on the phone` (5 times) and `Open the metal cabinet` (5 times) (Fig. 1, bottom right). In both cases, the object is accessible from multiple angles but some viewing angles are better for interaction (i.e., facing the keypad and drawer) and the robot must navigate accordingly. **(Double Pick)** *ASP* must put two objects in a target container. The queries cover a large variety of objects including headphones, a screwdriver, a penguin plushie, and a variety of toy food items (Fig. 4). We give a score of 0.5 per object in the container and no other partial points. We run 10 unique double pick queries with distractors. **(Spatial)** *ASP* must disambiguate a mobile pick-and-place query using spatial reasoning, e.g. "`Place the egg that is near the tomato in the pan`". We run 10 unique spatial queries with distractors.

such as `clean the counter`, `fold the shirt`, or `make the bed`—tasks that are achievable through imitation learning and VLAs (generalization notwithstanding). While this suggests a promising integration of structured scene representations and low-level learned policies, our results indicate that current scene representations can effectively generalize to more objects than VLAs can currently interact with in a zero-shot setting. Learned skills would increase the ceiling of *ASP* in terms of manipulation complexity, but would likely hinder semantic generalization. This non-trivial integration presents an interesting problem for future work.

ACKNOWLEDGMENTS

REFERENCES

[1] K. Black *et al.*, "π_0: A vision-language-action flow model for general robot control," *arXiv preprint arXiv:2410.24164*, 2024. 1, 2

[2] B. Zitkovich *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*. PMLR, 2023, pp. 2165–2183. 1, 2

[3] K. Pertsch *et al.*, "Fast: Efficient action tokenization for vision-language-action models," *arXiv preprint arXiv:2501.09747*, 2025. 1, 2, 6

[4] P. Intelligence *et al.*, "$\pi_{0.5}$: a vision-language-action model with open-world generalization," *arXiv preprint arXiv:2504.16054*, 2025. 1, 2, 6

[5] M. J. Kim *et al.*, "Openvla: An open-source vision-language-action model," *arXiv preprint arXiv:2406.09246*, 2024. 1, 2

[6] Q. Gu *et al.*, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 5021–5028. 1, 2, 3, 4, 7

[7] S. Koch, N. Vaskevicius, M. Colosi, P. Hermosilla, and T. Ropinski, "Open3dsg: Open-vocabulary 3d scene graphs from point clouds with queryable objects and open-set relationships," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024. 1, 2

[8] K. Jatavallabhula *et al.*, "Conceptfusion: Open-set multimodal 3d mapping," *Robotics: Science and Systems (RSS)*, 2023. 1, 2

[9] S. Peng *et al.*, "Openscene: 3d scene understanding with open vocabularies," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 815–824. 1, 2

[10] A. Werby, C. Huang, M. Büchner, A. Valada, and W. Burgard, "Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation," in *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024. 1, 2, 7

[11] S. Lu, H. Chang, E. P. Jing, A. Boularias, and K. Bekris, "Ovir-3d: Open-vocabulary 3d instance retrieval without training on 3d data," in *Conference on Robot Learning*. PMLR, 2023, pp. 1610–1620. 1, 2, 3, 4

[12] L. Paull *et al.*, *Towards Open-World Spatial AI*. Cambridge University Press. 1, 2

[13] C. Kassab *et al.*, "Openlex3d: A new evaluation benchmark for open-vocabulary 3d scene representations," *arXiv preprint arXiv:2503.19764*, 2025. 1, 2

[14] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PmLR, 2021, pp. 8748–8763. 1, 3

[15] J. Yang *et al.*, "Llm-grounder: Open-vocabulary 3d visual grounding with large language model as an agent," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 7694–7701. 1, 2

[16] J. Wang, M. Leonard, K. Daniilidis, D. Jayaraman, and E. S. Hu, "Evaluating pi0 in the wild: Strengths, problems, and the future of generalist robot policies," 2025. [Online]. Available: https://penn-pal-lab.github.io/pi0-Experiment-in-the-Wild 1, 6

[17] P. Liu, Y. Orru, J. Vakil, C. Paxton, N. M. M. Shafiullah, and L. Pinto, "Ok-robot: What really matters in integrating open-knowledge models for robotics," *arXiv preprint arXiv:2401.12202*, 2024. 2

[18] S. Yenamandra *et al.*, "Homerobot: Open-vocabulary mobile manipulation," *arXiv preprint arXiv:2306.11565*, 2023. 2, 7

[19] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Visual language maps for robot navigation," *arXiv preprint arXiv:2210.05714*, 2022. 2

[20] P. Liu *et al.*, "Dynamem: Online dynamic spatio-semantic memory for open world mobile manipulation," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 13 346–13 355. 2, 5, 7

[21] P. H. Winston, B. Katz, T. O. Binford, and M. R. Lowry, "Learning physical descriptions from functional definitions, examples, and precedents," in *AAAI Conference on Artificial Intelligence*, 1983. [Online]. Available: https://api.semanticscholar.org/CorpusID:7856739 2

[22] T.-T. Do, A. Nguyen, and I. Reid, "Affordancenet: An end-to-end deep learning approach for object affordance detection," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5882–5889. 2

[23] A. Delitzas, A. Takmaz, F. Tombari, R. Sumner, M. Pollefeys, and F. Engelmann, "Scenefun3d: Fine-grained functionality and affordance understanding in 3d scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 531–14 542. 2, 5

[24] T. Luddecke and F. Worgotter, "Learning to segment affordances," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017. 2

[25] T. Nagarajan and K. Grauman, "Learning affordance landscapes for interaction exploration in 3d environments," *ArXiv*, vol. abs/2008.09241, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221246369 2

[26] C. Zhang *et al.*, "Open-vocabulary functional 3d scene graphs for real-world indoor spaces," *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19 401–19 413, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:277314071 2

[27] J. Corsetti, F. Giuliari, A. Fasoli, D. Boscaini, and F. Poiesi, "Functionality understanding and segmentation in 3d scenes," *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24 550–24 559, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:274233938 2

[28] E. Tong, A. Opipari, S. Lewis, Z. Zeng, and O. C. Jenkins, "Oval-prompt: Open-vocabulary affordance localization for robot manipulation through llm affordance-grounding," *arXiv preprint arXiv:2404.11000*, 2024. 2

[29] S. Qian, W. Chen, M. Bai, X. Zhou, Z. Tu, and L. E. Li, "Affordancellm: Grounding affordance from vision language models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 7587–7597. 2

[30] G. Team *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023. 2, 5

[31] M. Deitke *et al.*, "Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models," 2024. [Online]. Available: https://arxiv.org/abs/2409.17146 2

[32] A. Kirillov *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 4015–4026. 2

[33] C. Zhang *et al.*, "Faster segment anything: Towards lightweight sam for mobile applications," *arXiv preprint arXiv:2306.14289*, 2023. 2, 4

[34] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots that use language," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 25–55, 2020. 2

[35] J. Liang *et al.*, "Code as policies: Language model programs for embodied control," *arXiv preprint arXiv:2209.07753*, 2022. 2

[36] I. Singh *et al.*, "Progprompt: Generating situated robot task plans using large language models," *arXiv preprint arXiv:2209.11302*, 2022. 2

[37] S. Yao *et al.*, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023. 2

[38] S. Salimpour *et al.*, "Towards embodied agentic ai: Review and classification of llm-and vlm-driven robot autonomy and interaction," *arXiv preprint arXiv:2508.05294*, 2025. 2

[39] R. Royce *et al.*, "Enabling novel mission operations and interactions with rosa: The robot operating system agent," in *2025 IEEE Aerospace Conference*. IEEE, 2025, pp. 1–16. 2

[40] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Conference on robot learning*. PMLR, 2022, pp. 894–906. 2

[41] M. J. Kim, C. Finn, and P. Liang, "Fine-tuning vision-language-action models: Optimizing speed and success," *arXiv preprint arXiv:2502.19645*, 2025. 2

[42] "Langchain," https://github.com/langchain-ai/langchain, 2022. 5

[43] H.-S. Fang *et al.*, "Anygrasp: Robust and efficient grasp perception in spatial and temporal domains," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3929–3945, 2023. 5

[44] 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. 5

[45] Z. Yan *et al.*, "Dynamic open-vocabulary 3d scene graphs for long-term language-guided mobile manipulation," *IEEE Robotics and Automation Letters*, 2025. 5, 7

[46] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE robotics & automation magazine*, vol. 19, no. 1, pp. 18–19, 2012. 5

[47] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001. 5

[48] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. 5

[49] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. 5

[50] A. Khazatsky *et al.*, "Droid: A large-scale in-the-wild robot manipulation dataset," *arXiv preprint arXiv:2403.12945*, 2024. 6

[51] D. Honerkamp, M. Büchner, F. Despinoy, T. Welschehold, and A. Valada, "Language-grounded dynamic scene graphs for interactive object search with mobile manipulation," *IEEE Robotics and Automation Letters*, vol. 9, no. 10, pp. 8298–8305, 2024. 7

[52] 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 task planning," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=wMpOMO0Ss7a 7

[53] A. Gupta, M. Zhang, R. Sathua, and S. Gupta, "Opening articulated structures in the real world," *arXiv preprint arXiv:2402.17767*, 2024. 7

APPENDIX

A. Contribution Statement

Sacha Morin conceptualized and coordinated the project, implementing most of the *ASP* perception and robot code and writing the majority of the manuscript.

Kumaraditya Gupta designed and implemented the affordance detection pipeline and wrote sections of the manuscript, also designing most figures, and helping with the *ASP* experiments. Kumaraditya was also pivotal in some early exploratory work, including mobile manipulation in Isaac Sim and segmentation in SceneFun3D.

Mahtab Sandhu led the π_0-FAST and $\pi_{0.5}$ experiments, providing a critical comparison with VLAs.

Charlie Gauthier and **Francesco Argenziano** participated in multiple brainstorming sessions, also respectively contributing to some skill implementations and the deployment of AnyGrasp.

Kirsty Ellis spearheaded the initial integration of the mobile manipulator and provided invaluable hardware support throughout the project.

Liam Paull was the lead advisor on this project, providing critical feedback that shaped the manuscript and the experiments, also writing and proof-reading sections of the manuscript.

B. Progress Rate and VLA Failure Analysis

We study task progression for the manipulation tasks in Fig. 7 and some specific π_0-FAST and $\pi_{0.5}$ failures in Fig. 8.
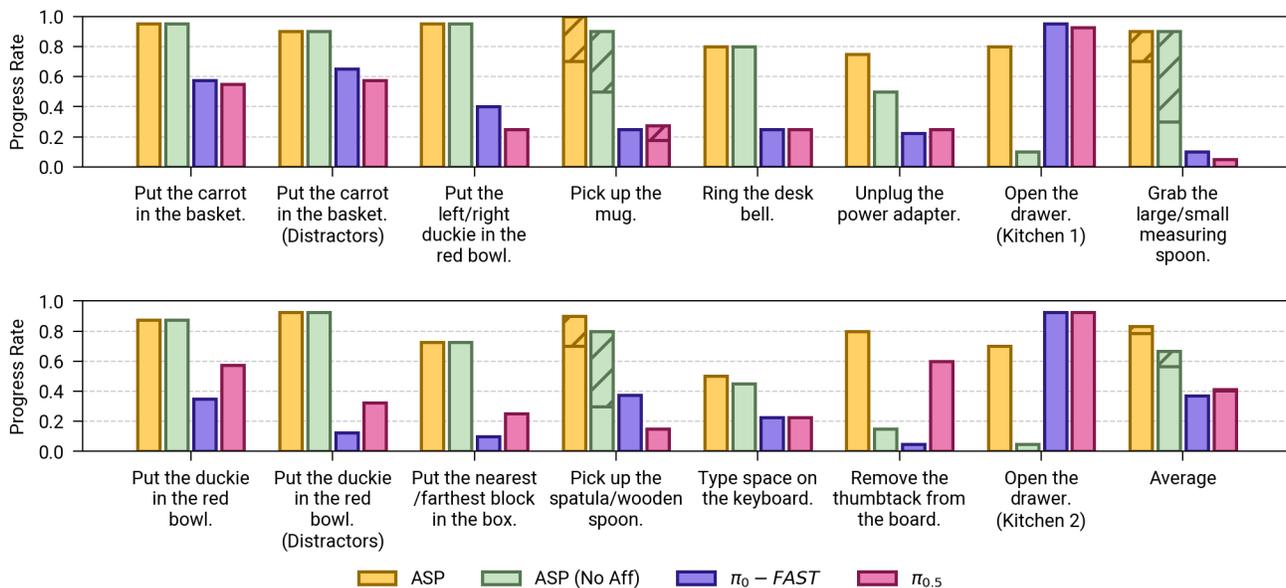
Fig. 7: **Progress Rate.** To complement the success rates in Table 3 we report the average progress rate for *ASP*, *ASP* (No Aff), π_0-FAST and $\pi_{0.5}$. We report the average progress rate over 10 attempts per task, giving partial points when the gripper comes close to the relevant object (0.25), when a pick is successful but subsequent steps fail (0.5), and when the policy attempts a reasonable motion (pushing, pulling) on the object without achieving a successful outcome (0.5). Task successes receive a progress rate of 1.00. While the VLAs show some level of task progression on most tasks, they still underperform *ASP*.
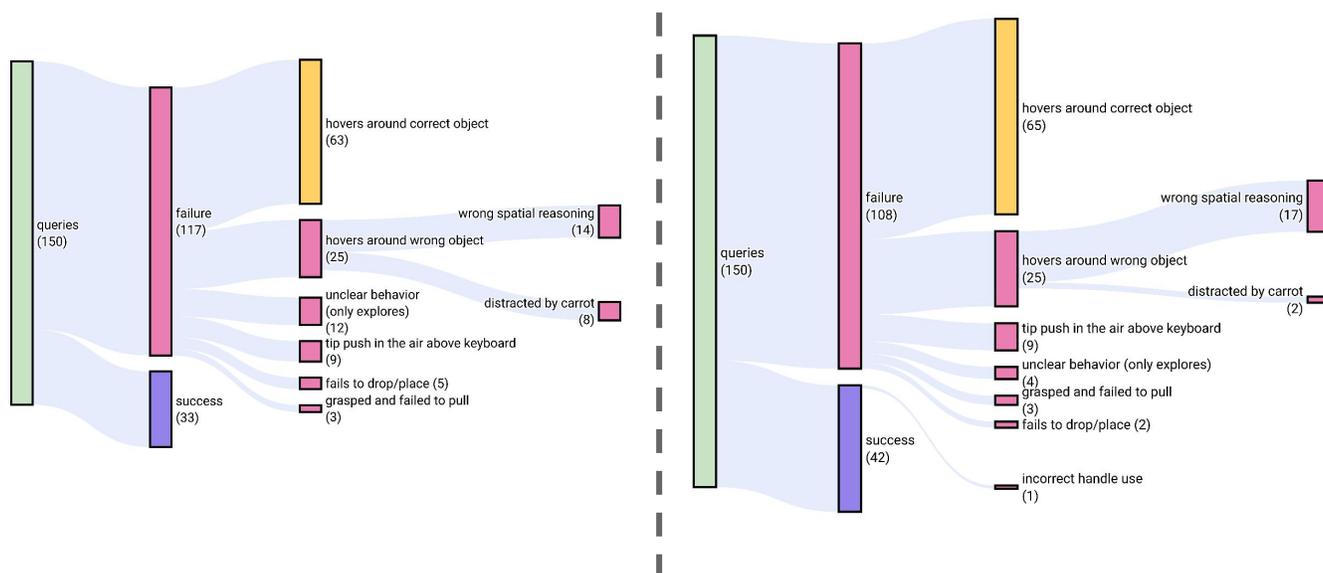

Fig. 8: **Failure Analysis of π_0-FAST and $\pi_{0.5}$.** While the end-to-end nature of π_0-FAST and $\pi_{0.5}$ makes it difficult to find a causal explanation for failures, we can still attempt to classify trials in terms of their outcomes and the behaviors of the methods according to a human evaluator. We analyze outcomes for π_0-FAST (**Left**) and $\pi_{0.5}$ (**Right**). By far, the most common failure modes involve gripper interactions. Examples include "hovering" around the correct object without successfully grasping, or partially executing a skill midair above the object (keyboard), suggesting high-level query understanding, but potential issues with depth perception or excessive collision avoidance behaviors. In terms of language following, we found the VLAs to generally struggle with spatial reasoning (left/right, near/far, small/large). π_0-FAST was especially susceptible to distraction by the toy carrot.