Distilling On-device Language Models for Robot Planning with Minimal Human Intervention

Zachary Ravichandran, Ignacio Hounie, Fernando Cladera, Alejandro Ribeiro, George J. Pappas, Vijay Kumar

University of Pennsylvania

Abstract: Large language models (LLMs) provide robots with powerful contextual reasoning abilities and a natural human interface. Yet, current LLM-enabled robots typically depend on cloud-hosted models, limiting their usability in environments with unreliable communication infrastructure, such as outdoor or industrial settings. We present PRISM, a framework for distilling small language model (SLM)-enabled robot planners that run on-device with minimal human supervision. Starting from an existing LLM-enabled planner, PRISM automatically synthesizes diverse tasks and environments, elicits plans from the LLM, and uses this synthetic dataset to distill a compact SLM as a drop-in replacement of the source model. We apply PRISM to three LLM-enabled planners for mapping and exploration, manipulation, and household assistance, and we demonstrate that PRISM improves the performance of Llama-3.2-3B from 10-20% of GPT-4o's performance to over 93% — *using only synthetic data*. We further demonstrate that the distilled planners generalize across heterogeneous robotic platforms (ground and aerial) and diverse environments (indoor and outdoor). We release all software, trained models, and datasets at https://zacravichandran.github.io/PRISM.

Keywords: LLM-enabled robots, LLM Distillation

1 Introduction

Large language Models (LLMs) have emerged as powerful tools for robot planning by providing contextual reasoning and a natural human interface. LLM-enabled robots have been successfully deployed for mobile manipulation [1, 2, 3], autonomous driving [4, 5], service robots [6], navigation [7, 8], and inspection [9, 10]. Being trained on internet-scale data, LLMs contain rich semantic knowledge and are highly adaptable. With only a few in-context examples, they can attune to task-relevant information and generalize to novel robotic scenarios.

However, the scale of LLMs brings significant computational demands. Current frontier models contain hundreds of billions of parameters and require significant cloud computing infrastructure [11, 12]. Although small language models (SLMs) can run on-device, their relatively poor performance in robot planning domains has precluded widespread adoption by the community [13]. Consequently, nearly all LLM-enabled robots rely on models such as GPT or PALM [14, 1, 3, 9, 4, 15, 16, 6], which tethers them to external compute. While this dependence may be feasible in structured environments such as homes or offices, it severely limits deployment in large-scale, unstructured settings — domains critical for environmental monitoring, industrial inspection, disaster response, and a host of similar problems.

To address these limitations, we introduce PRISM (Planning with Robotic dIstilled Small language Models), a framework for distilling SLMs that run on-device with minimal human intervention. Starting from an existing LLM-enabled planner (termed "source"), PRISM generates diverse tasks and textual environment representations — such as scene graphs or object sets — and uses these generations to elicit plans from the source LLM. This interaction produces high-quality planning demonstrations without manual annotations, simulators, or dataset curation. The user simply pro-

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

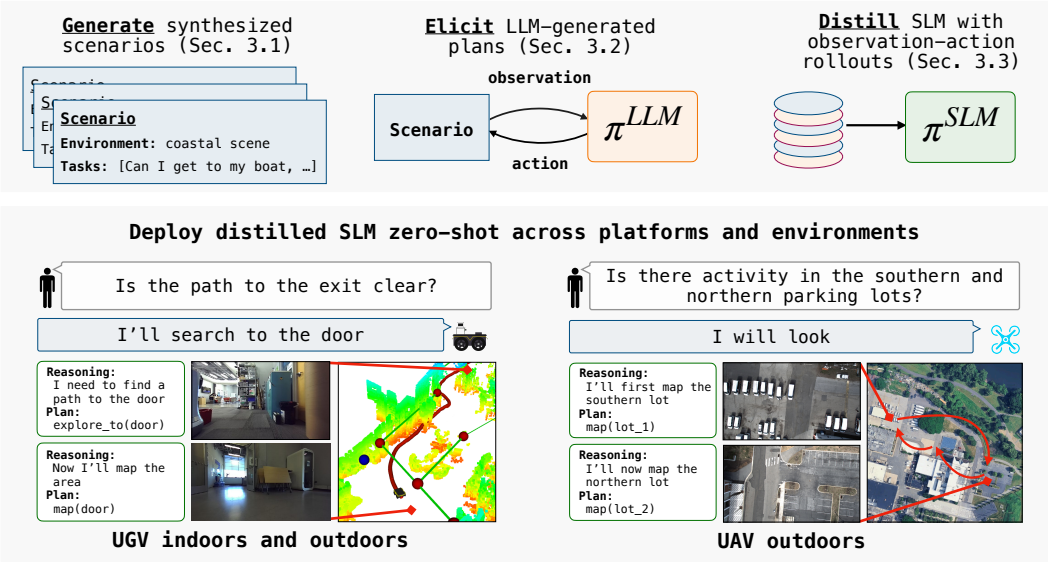

Figure 1: PRISM distills on-device language models for robot planning with minimal human intervention. Given an existing LLM-enabled planner, PRISM synthesizes diverse task and textual environments, elicits plans from the LLM, then uses the resulting dataset to fine-tune an SLM that rivals the performance of the original planner, while being extremely compute efficient. We demonstrate PRISM-distilled models across platforms and environments.

vides the LLM-enabled planner along with its action and observation spaces. PRISM then uses the generated dataset to fine-tune an SLM that shares the original planner's interface, enabling it to directly replace the source LLM. We outline PRISM in Fig. 1, we summarize PRISM's **contributions**:

- **Effectiveness.** PRISM distills SLMs that rival the planning performance of a source LLM-enabled planner, such as GPT-4o, as measured by planning success rate.
- **On-device models.** PRISM effectively distills language models that are small enough to run onboard robot compute. In this work, we consider models with a memory footprint of under 5GB.
- **Minimal human interaction.** PRISM requires a high-level configuration comprising the source planner's observation space, action space, *etc.*, but it does not require manual dataset construction, simulators, details about the deployment environment, or other labor intensive information.

We evaluate PRISM across three LLM-enabled planners — SPINE on real-world air and ground robots [9], LLM-Planner in the ALFRED simulator [16, 17], and SayCan in a manipulation simulator [18]. While undistilled SLMs achieve only 10-20% of GPT-4o's success rate, PRISM boosts performance to over 93% across all three settings. Moreover, the distilled SLMs compute efficiency enables them to plan onboard a robot in closed-loop control settings. We also release datasets, models, and code in order to promote reproducibility. The rest of the paper is structured as follows. We overview related work in §2, describe our framework in §3, present experimental results in §4, provide concluding remarks in §5, and discuss limitations in §6.

2 Related work

LLM-Enabled Robots. LLMs have emerged as a powerful enabler of contextually-aware robots, as evidenced by their deployment in mobile manipulation [1, 3, 2, 18, 19], service robots [6], task and motion planning [20], autonomous driving [21, 4], anomaly detection [22, 23], navigation [14, 8, 7, 24], exploration and mapping [9, 10], and multi-robot teaming [25]. These works forego any model training; instead, they configure a pre-trained LLM via in-context examples. While convenient and effective, this approach requires the use of extremely large models (over 100B parameters) — all the above works use GPT-4 [11], PALM [26], or similar — which are far too large to run onboard a robot and therefore tethers the robot to high-quality network infrastructure.

Distilling LLMs. LLM distillation aims to retain the contextual reasoning abilities of LLMs while overcoming their compute limitations. These methods train an SLM to emulate the generations of a source LLM via supervised fine tuning (SFT) [27, 28, 29], reinforcement learning [30, 31], and ranking optimization [32, 33], among other methods. By finetuning a target SLM rather than training from scratch, these methods enjoy significant compute and data efficiency [34]. While traditional distillation methods curate a dataset of queries with which to elicit LLM generations, an emerging line of *self-distillation* research uses an LLM to generate *both* queries and responses [35, 36, 37], thereby reducing the need for dataset curation. In contrast to prior self-distillation methods which operate in the virtual domain, methods for robot planning must be *grounded* in the physical world by respecting robot affordances, responding to sensory input, and operating in closed-loop settings. While our methodology draws from general distillation techniques, its key novelty lies in addressing the unique challenges of robot planning, which prior distillation work has not considered.

Distilling Robot Planners. Distillation has been applied to virtual planning domains, including web search or video games [38, 39, 40, 41, 42]. These methods typically use high-quality queries — such as a dataset of instructions — to elicit planning examples from an LLM, which are then used to train a smaller model. Qiao et al. [43] apply self-distillation principles to this realm by bootstrapping data with generations from an LLM. Recently, roboticist have applied distillation techniques to problems including multi-robot control [44] and manipulation [45]. Of particular note is the work by Ha et al. [13], who use an LLM to generate expert plans which are used to distill a visuomotor policy. While these methods successfully leverage the contextual knowledge of an LLM to train downstream planners, they still require external knowledge — such as a simulator or curated data — and these works distill auxiliary planners that cannot replace the source LLM. In contrast, PRISM *synthesizes all training data* directly from a source LLM-enabled planner. The resulting distilled SLM is not an auxiliary component but a compact planner designed to *directly replace the source LLM,* making it deployable onboard robots without network reliance.

3 Distilling On-device LMs for Robot Planning

PRISM distills an SLM that mimics the performance of an LLM-enabled planner with minimal human intervention — *i.e.*, without reliance on datasets or simulators. Our key observation is that because the action and observation spaces of an LLM-enabled planner are textual, they can be synthesized and used to elicit plans from the LLM, and the resulting data can be used to train an SLM.

Concretely, PRISM takes as input a *source* LLM-enabled planner, $\pi^{\text{LLM}}(a_i|o_i, s)$, which is conditioned on a task t, takes an observation o_i, and produces an action a_i, at planning iteration i. These input-output sets must be textually represented, but we place no further restrictions on their format. PRISM's objective is to train a *target* SLM — *e.g.,* Llama-3.2-3B or similar — to produce a planner, $\pi^{\text{SLM}}(a_i|o_i, t)$ that mimics the source LLM while being significantly more efficient.

PRISM comprises three modules — *scenario generation*, *plan elicitation*, and *planner distillation*. The scenario generation module synthesizes a diverse set of tasks and environments (termed "scenarios," §3.1). These scenarios are used by the elicitation module to extract plans from the LLM-enabled planner, and this process yields a dataset of task-observation-action sequences (§3.2). Finally, the planner distillation module uses this synthesized dataset to train a target SLM via SFT (§3.3). We discuss each module in detail below.

3.1 Scenario Generation

The scenario generation module takes as input the LLM-enabled planner's action and observation spaces and produces *scenarios* comprising environments — which produce observations for the planner — and semantically coherent tasks. We instantiate this module with an LLM (GPT-4o), given their generative abilities. This scenario generation LLM is prompted with the observation format along with the desired environment size and number of tasks, which are hyperparameters of this module. This generation procedure, combined with a high temperature (1 in our work), promotes data diversity. We provide the high-level format; please see the appendix for the full prompt:

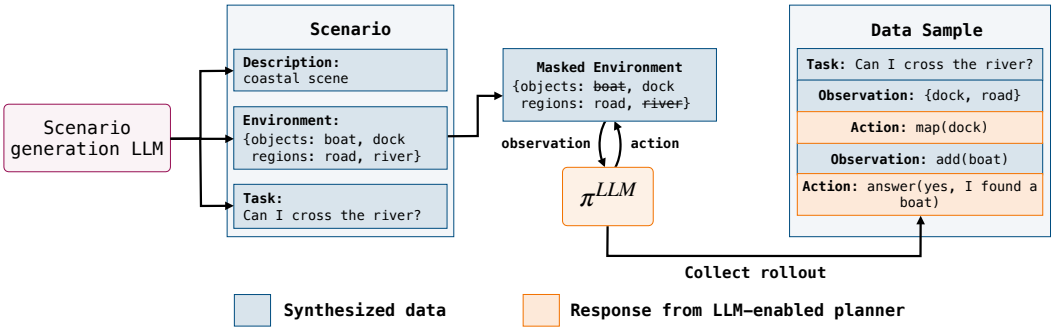

Figure 2: Synthesizing a data sample: The scenario generator synthesizes an environment and semantically coherent task. The environment is masked, yielding an initial partial observation for the LLM-enabled planner. Given this task and partial observation, the LLM-enabled planner iteratively provides actions and receives new observations until the task is complete. This rollout is collected as a data sample and added to the synthesized dataset.

```
You are generating data for training an llm-enabled planner.
The planner takes observations in the following format: <OBSERVATION>.
Generate an environment of size <SIZE> with <N_TASKS> corresponding tasks
and a description of the scenario.
```

Along with `environments` and `tasks`, the generator provides a `description` to enforce semantic coherence. All generations are verified to ensure environments adhere to the representation required by the source planner; the generator is re-prompted if it provides an erroneous environment.

In our instantiation of PRISM, we consider two observation representations. First, we consider a scene graph with nodes of type `region` or `object`; `object` nodes represent semantic entities, while `region` nodes are traversable points in the scene. Edges in the graph are defined between two regions ("region nodes") or an object and a region ("object nodes"). Region edges indicate traversable paths, while object edges indicate visibility from a particular location. Nodes can also be enriched with semantic descriptions derived from a vision-language model or similar perception system. We next consider object sets — a popular representation for manipulation and instruction-following tasks — which define all objects accessible by the planner. If using the scene graph representation, an output from the scenario generation LLM would look like the following:

Scenario generation format

```
{"description": "scene description",
"environment": "environment",
"task": ["task_1", "task_2", "..."]}
```

Graphical environment representation

```
{"regions": ["..."],
"objects": ["..."],
"edges": ["..."]}
```

3.2 Plan Elicitation

Given the generated scenarios, the plan elicitation module extracts plans from the source LLM-enabled planner via three steps — *environment masking*, *closed-loop planning emulation*, and *plan validation*. Environment masking produces a partial observation, o^1, for the planner by hiding a randomized portion of the environment — *i.e.,* removing nodes and edges from a graph, removing objects from a set, *etc.*. The plan elicitation module then emulates closed-loop planning by using the provided action space. Each action defines some operation on the environment — for example, a `map` action may reveal previously unobserved objects, a `goto` action would change the robots location, *etc.* The plan elicitation module uses this action-observation relationship to simulate what the planner *would have seen* had it taken a certain action in a synthesized environment — *i.e.,* observation o^{i+1} given action a^i and the synthesized environment. In our instantiation of PRISM, we consider actions for mapping, navigation, and object interaction, though more actions may be considered if needed. Given this ability to emulate closed-loop planning, the module seeds the LLM-

enabled planner with the initial observation and iterates until termination. Termination occurs when the planner signals task completion (*e.g.,* via a `done` action) or a reaches a pre-defined maximum number of planning iterations. Should the planner timeout, we consider the sample *invalid*. This simple validation procedure has significant effects on distilled SLM performance (see §4.4), and this process yields a rollout of observation-pairs that is collected into dataset:

$$\mathcal{D} = \{s_i, (o_i^1, a_i^1, \ldots, o_i^T, a_i^T)\}_{i=1}^N \tag{1}$$

where N is the number of scenarios, and T is the plan trajectory length (which may differ across scenarios). We provide an example of plan elicitation procedure in Fig. 2.

3.3 Planner Distillation

Given the dataset $\mathcal{D}$, the planner distillation module trains the target SLM-enabled planner π^{SLM} using supervised fine-tuning (SFT), minimizing the cross entropy over every action $a_i^1 \ldots a_i^T$:

$$\min_{\pi^{SLM}} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} CE\left(\pi^{SLM}\left(a_i^t | s_i, o_i^1, a_i^1, \ldots o_i^{t-1}.a_i^{t-1}\right),\ a_i^t\right) \tag{2}$$

Each action prediction utilizes the cumulative context of all preceding observation-action pairs. We train on all actions jointly, which has shown improved efficiency without performance degradation compared to more elaborate training schemes in multi-turn SFT of LMs [46]. Since the objective in Eq. 2 only involves actions from the LLM planner, it can be evaluated in a single forward-pass without the overhead of autoregressive sampling of SLM actions. To reduce GPU memory footprint during finetuning, we leverage LoRA [34], a popular parameter efficient fine tuning technique. LoRA adds a low rank trainable update to pre-trained model weights, thereby shrinking optimizer states. For a 3 billion parameter Llama, LoRA with rank 8 requires only 6 million trainable parameters. Since updates can be merged with the weights, it incurs no additional inference cost. And while PRISM's primary objective is to produce a small language model for robot planning, employing efficient optimization techniques is a further advantage of our method.

4 Experimental Results

We evaluate PRISM in order to assess how well it fulfills the contributions outlined in §1. By design, PRISM requires **minimal human intervention**, and we empirically evaluate the second two contributions. First, we evaluate **effectiveness**: does PRISM distill SLM-enabled planners with comparable performance to the source LLM-enabled planner? We then evaluate **efficiency**: do the produced models run on-device in real-time? We also measure the impact of two key components of our data synthesis pipeline — environment masking and plan validation. We evaluate PRISM over three LLM-enabled planners [9, 18, 16] in distinct environments and tasks with the following methodology. We first evaluate the planner using an LLM, as it was original designed. We then replace the planner's LLM with an SLM, and we measure performance. Finally, we apply PRISM to the planner, and we measure the performance of the planner while using the PRISM-distilled SLM.

4.1 Experimental setup

We consider the following three LLM-enabled planners. For each planner, we describe the evaluation tasks and experimental environments, as outlined in Fig. 3.

SPINE [9] is designed for language-driven navigation, mapping and exploration in partially-known and unstructured environments. The planner operates in a closed-loop manner: actively mapping its scene and updating its plan based on its findings.

- **Tasks.** We consider two levels of difficulty. First, we evaluate against *fully-specified* tasks that directly reference actions and semantics in the scene (*i.e.,* "inspect the desk"). Next, we evaluate against *under-specified* tasks that require the planner to infer relevant actions and semantics (*i.e.,* "find me a place to work"). We consider 20 indoor UGV tasks, 15 outdoor UGV tasks, and 10 UAV tasks, split evenly between fully- and under-specified variants.

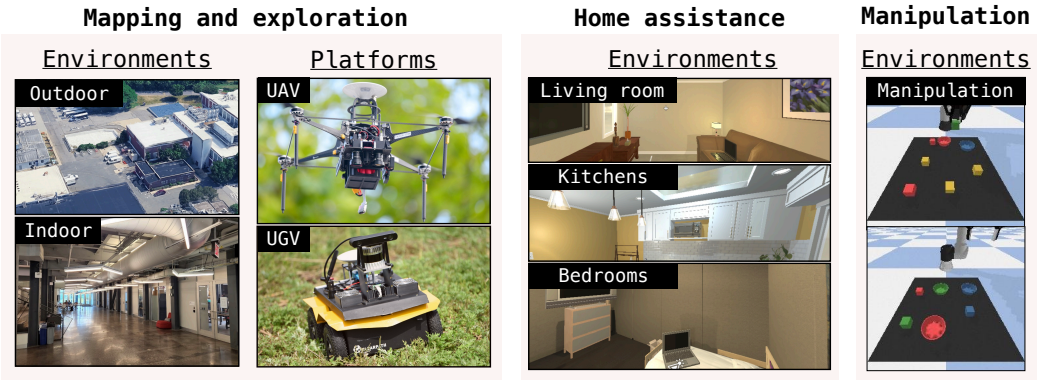

Figure 3: We evaluate PRISM on three distinct LLM-enabled planning domains: mapping and exploration, home assistance, and manipulation. Left: SPINE [9] in both indoor and outdoor environments on bot UAVs and UGVs. Middle: LLM-Planner [16] to household tasks in the ALFRED [17] simulator. Right: SayCan [18] to manipulation tasks.

- **Environments.** We evaluate SPINE in both indoor and outdoor environments with the following levels of difficulty: *mapping* tasks require the robot to collect semantic information, and *exploration* tasks require the robot to collect both metric and semantic information — *i.e,* explore free space. We consider 15 exploration tasks and 30 mapping tasks.

SayCan [1] is a hierarchical language-guided planner that combines an LLM's task-level reasoning with lower-level affordance estimates. Instead of token-wise probabilities used in the original setting, we prompt the LLM to generate candidate actions consistent with our framework's textual input–output planner abstraction.

- **Tasks.** A robot arm is tasked with rearranging colored bowls and boxes on a table. In the absence of an open-source benchmark, we design twenty tasks, each rendered in multiple scene variants. Following [47], we introduce *Attribute ambiguity* — replacing canonical terms with synonyms — and *Numeric ambiguity* — swapping exact counts for vague phrases such as a "some." We also include more challenging variants with arithmetic constraints, color–based negation, and load balancing, forcing the planner to reason over these specifications. See Appendix for details.
- **Environments**. We use the PyBullet [48] simulator. For each task, we create multiple environments with different configurations of colored blocks and bowls, and randomized placings.

LLM-Planner [16] casts language-driven household manipulation tasks as few-shot text-to-plan generation. The planner queries an LLM (GPT4) with the task specification, an object-set description of the observed environment, and 9 in-context examples; these in-context examples are chosen by matching LLM embeddings between the input task and a pre-defined set of 100 expert demonstrations. The LLM returns a sequence of semantic sub-goals (e.g., `Navigation fridge`, `OpenObject fridge`) which, if feasible, are realized by a lower-level controller

- **Tasks.** We consider tasks from Action Learning From Realistic Environments and Directives (ALFRED) [17]. We evaluate on the seven ALFRED templates: *Pick & Place*, *Stack & Place*, *Pick-Two & Place*, *Clean & Place*, *Heat & Place*, *Cool & Place*, and *Examine-in-Light*, and we report results in the Validation split. Directives provide only a single goal sentence, so the planner must infer all concrete actions.
- **Environments.** Experiments are run in the *unseen* split of ALFRED – i.e., environments differ from the in-context examples from the training set – comprising eight AI2-THOR [49] homes with kitchens, bedrooms, bathrooms, and living rooms. The agent must explore cluttered, partially observable rooms containing up to 115 object types.

Baselines and training: We compare PRISM against the above planners equipped with the following language models: GPT-4o (**LLM**) and Llama-3.2-3B (**SLM**), which we choose because of their

Figure 4: In the following examples, PRISM completes the task, and we highlight failure modes of the SLM. Top right: SLM does not identify semantic targets. Top left: SLM incorrectly invokes planner API and mistakes logical ordering; two blocks cannot be simultaneously help by the robot. Bottom right: SLM makes unwarrented assumptions about the environment. Bottom left: SLM incorrectly invokes API and the plan terminates.

widespread use and competitive performance within their size class; we defer comparative studies to future work. Moreover, the selected models have acceptable latency for the experimental setup mentioned above. Together, these baselines allow us to measure both the comparative improvement PRISM provides over a base SLM relative to a SoTA LLM. PRISM's trains on 3,200 examples for SPINE, 1,500 examples for SayCan, and 1000 examples for LLM-Planner.

Hardware Platforms. We evaluate SPINE on two physical platforms. First, we consider a UGV — a Clearpath Jackal — equipped with an Ouster OS1-64 LiDAR and ZED-2i for object detection and obstacle avoidance, and compute comprising a Ryzen 5 3600 CPU and an Nvidia RTX 4000 SSF Ada GPU. Second, we consider the Falcon 4 UAV, equipped with a downward-facing RGB camera and an Nvidia Jetson Orin NX computer.

Method	Success rate (%)		
	SPINE [9]	SayCan [1]	LLM-Planner [16]
SLM	13.54	10.0	1.76
PRISM	92.00	50.0	15.54
LLM	98.07	50.0	16.49

Table 1: Comparison of SLM, PRISM, and GPT-4o in SPINE [50], SayCan [18], and LLM-Planner [16]

4.2 Model Effectiveness

We report success across all three planners in Table 1. GPT-4o achieves a 98.07%, 50%, and 16.49% success rate across SPINE, SayCan, and LLM-Planner respectively, which match trends from the original publications (SayCan is slightly lower due to our custom benchmark). Without distillation, SLM achieves 13.54%, 10%, and 1.76% success. PRISM boosts this performance to 92%, 50%, and 15.54%, which amounts to at least a 93% increase across all three planners. Qualitatively, PRISM addresses two primary failure modes of the SLM. First, the SLM will provide a feasible plan (*i.e.,* using the correct syntax), but the plan will be semantically incorrect. In the second failure mode, the SLM fails to provide a valid plan altogether. We highlight comparative examples in Fig. 4.

	Fully-specified (%)			Under-specified (%)		
	LLM	Ours	SLM	LLM	Ours	SLM
Mapping	100	100	50	100	90	0
Exploration	100	87.5	0	83.3	75	0

Table 2: Planning success rate in varying levels of environment and task difficulty

We then analyze performance over varying levels of task and environment difficulty in SPINE, as reported in Tab. 2. In the easiest scenarios — fully specified mapping tasks — GPT-4o and PRISM both achieve 100% success rate, while SLM only achieves 50% success. We observe a modest (10%) performance decrease for PRISM in under-specified tasks. Exploration presents the most

difficult setting for all models. PRISM's performance drops to 75% in under-specified exploration, while GPT-4o only achieved 83.3% success. Notably, SLM performance drops to 0% in difficulty settings beyond fully-specified mapping missions. These results suggest that spatial reasoning is still comparatively difficult for LLM-enabled planners, and PRISM inherits this weakness.

4.3 Model efficiency

We next perform latency analysis between PRISM-distilled models and GPT-4o — our baseline LLM — using SPINE running on the UGV (see §4.1). We measure the latency of 60 PRISM queries, and we compare this to 60 GPT-4o queries under the following network simulations. *Ethernet - Minimal latency* corresponds to the best scenario for querying GPT-4o, where the robot is connected to the internet using an ethernet connection. The round-trip time to `openai.com` is 6.7 ms, and we obtain a mean query time of 3.64 s, with a standard deviation of 2.25 s. *Ethernet - 200 ms* adds 100ms of latency to each received and transmitted packet using the utility traffic controller for Linux [51], which simulates the conditions of a robot using satellite internet services and mesh networks. In this setting, we obtain a mean query time of 4.90 s, with a standard deviation of 1.73 s. Fig. 5 shows a density estimate for the different evaluated methods. We observe that the latency of PRISM is within 200 ms of GPT-4o under ideal conditions. PRISM is more than 1 second faster than GPT-4o under realistic deployment conditions, while exhibiting less variance. This highlights a major advantage of PRISM: besides being able to operate *without any network*, it offers more *deterministic* latency as it does not depend on uncertainties in the network conditions.

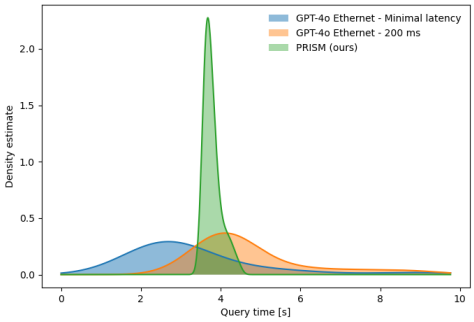

Figure 5: Latency comparison between GPT-4o and PRISM

4.4 Importance of environment masking and plan validation

Finally, we measure the importance of two key components of our data synthesis framework — environment masking and plan validation — by ablating these components. We evaluate performance across eleven tasks and environments sampled from the larger collection described in §4.1, and we report result in Tab. 2. Without environment masking, performance decrease by 27.3% to 63.6%. Having not been trained with interactive data, the this variant fails when success require many planning iterations. Without validation, performance drops by nearly half to 45.5%. The no-validation variant mimics failure modes from source LLM not properly reacting to new observations and repeatedly calling the same, ineffective action; these failure modes are typically filtered out with validation.

Method	Success
Ours	90.9%
(-) environment masking	63.6%
(-) validation	45.5%

Table 3: Data synthesis ablation

5 Conclusion

We present PRISM, a framework for distilling effective SLM-enabled robot planners with minimal human interaction. Given a source LLM-enabled planner, PRISM synthesizes tasks and environments, elicits plans from the LLM-enabled planner in these synthesized environments, and then uses the resulting data to train an SLM-enabled planner that serves as a drop-in replacement for the source model. We evaluate PRISM's ability to distill SLMs that rival the performance of LLMs on three distinct planning domains. We find that while an undistilled SLM (Llama-3.2-3B) achieves between 10-20% performance of GPT-4o, PRISM boosts this to over 93% *using only synthetic data*, and we successfully deploy PRISM-distilled models for planning on air and ground robots. Future work may investigate methods for synthesizing higher quality data — in particular, data that leads to improved spatial reasoning of the distilled SLMs.

6 Limitations

We identify two primary limitations to our work. Firstly, we consider LLM-enabled planners that operate over atomic action primitives, such as those for navigation, mapping, and manipulation. While such action spaces encompass a large portion of the existing literature [3, 9, 10, 18, 52, 2, 8], a class of LLM-enabled planners use formal languages, code, or other action representations [19, 53, 54]. Because the latter action representations are still textual, PRISM is likely comparable with these methods, though future work should validate this hypothesis. Secondly, recent work has shown that even frontier LLMs can produce harmful robotic actions [55]. We anticipate that PRISM inherits these safety vulnerabilities, and we propose two mitigation strategies. In the short term, our work is compatible with recently-proposed safeguards for LLM-enabled robots [50, 56, 57]. Future research could also directly incorporate safety objectives into PRISM's distillation process [30, 58].

Acknowledgments

We gratefully acknowledge support from the Distributed and Collaborative Intelligent Systems and Technology (DCIST), NSF Grant CCR-2112665, DARPA SAFRON grant HR0011-25-3-0135, and the NSF Graduate Research Fellowship. We thank Antonio Loquercio and Dinesh Jayaraman for their support during the inception of this work in *ESE 6800: Real World Robot Learning*, and Varun Murali and Anish Bhattacharya for their helpful feedback on the manuscript.

References

[1] B. Chen, F. Xia, B. Ichter, K. Rao, K. Gopalakrishnan, M. S. Ryoo, A. Stone, and D. Kappler. Open-vocabulary queryable scene representations for real world planning. In arXiv preprint arXiv:2209.09874, 2022.

[2] D. Honerkamp, M. Büchner, F. Despinoy, T. Welschehold, and A. Valada. Language-grounded dynamic scene graphs for interactive object search with mobile manipulation. arXiv preprint arXiv:2403.08605, 2024.

[3] 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. URL https://openreview.net/forum?id=wMpOMO0Ss7a.

[4] M. Omama, P. Inani, P. Paul, S. C. Yellapragada, K. M. Jatavallabhula, S. Chinchali, and M. Krishna. Alt-pilot: Autonomous navigation with language augmented topometric maps. arxiv:2310.02324, 2023.

[5] B. Li, Y. Wang, J. Mao, B. Ivanovic, S. Veer, K. Leung, and M. Pavone. Driving everywhere with large language model policy adaptation. CVPR, 2024.

[6] Z. Hu, F. Lucchetti, C. Schlesinger, Y. Saxena, A. Freeman, S. Modak, A. Guha, and J. Biswas. Deploying and Evaluating LLMs to Program Service Mobile Robots. IEEE Robotics and Automation Letters, 2024. doi:10.1109/LRA.2024.3360020.

[7] D. Shah, B. Osiński, b. ichter, and S. Levine. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In K. Liu, D. Kulic, and J. Ichnowski, editors, Proceedings of The 6th Conference on Robot Learning, volume 205 of Proceedings of Machine Learning Research, pages 492–504. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/shah23b.html.

[8] D. Shah, M. R. Equi, B. Osiński, F. Xia, B. Ichter, and S. Levine. Navigation with Large Language Models: Semantic Guesswork as a Heuristic for Planning. In J. Tan, M. Toussaint, and K. Darvish, editors, Proceedings of The 7th Conference on Robot Learning, volume 229 of Proceedings of Machine Learning Research, pages 2683–2699. PMLR, 06–09 Nov 2023.

[9] Z. Ravichandran, V. Murali, M. Tzes, G. J. Pappas, and V. Kumar. SPINE: Online Semantic Planning for Missions with Incomplete Natural Language Specifications in Unstructured Environments. International Conference on Robotics and Automation (ICRA), 2025.

[10] M. Fadhil Ginting, D.-K. Kim, S.-K. Kim, B. J. Krishna, M. J. Kochenderfer, S. Omidshafiei, and A.-a. Agha-mohammadi. SayComply: Grounding Field Robotic Tasks in Operational Compliance through Retrieval-Based Language Models. arXiv e-prints, pages arXiv–2411, 2024.

[11] OpenAI. Gpt-4 technical report. Technical report, OpenAI, 2023. URL https://cdn.openai.com/papers/gpt-4.pdf.

[12] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

[13] H. Ha, P. Florence, and S. Song. Scaling up and distilling down: Language-guided robot skill acquisition. In J. Tan, M. Toussaint, and K. Darvish, editors, Proceedings of The 7th Conference on Robot Learning, volume 229 of Proceedings of Machine Learning Research, pages 3766–3777. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/ha23a.html.

[14] Z. Dai, A. Asgharivaskasi, T. Duong, S. Lin, M.-E. Tzes, G. Pappas, and N. Atanasov. Optimal scene graph planning with large language model guidance. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 14062–14069. IEEE, 2024.

[15] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. arXiv preprint arXiv:2304.11477, 2023.

[16] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2023.

[17] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020. URL https://arxiv.org/abs/1912.01734.

[18] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In arXiv preprint arXiv:2204.01691, 2022.

[19] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In arXiv preprint arXiv:2209.07753, 2022.

[20] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. arXiv preprint arXiv:2306.06531, 2023.

[21] S. Sharan, F. Pittaluga, V. Kumar B G, and M. Chandraker. LLM-Assist: Enhancing Closed-Loop Planning with Language-Based Reasoning. arXiv preprint arXiv:2401.00125, 2023.

[22] A. Tagliabue, K. Kondo, T. Zhao, M. Peterson, C. T. Tewari, and J. P. How. Real: Resilience and adaptation using large language models on autonomous aerial robots. Conference on Robot Learning (CoRL), 2023.

[23] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone. Real-Time Anomaly Detection and Reactive Planning with Large Language Models. In Robotics: Science and Systems, 2024.

[24] Q. Xie, T. Zhang, K. Xu, M. Johnson-Roberson, and Y. Bisk. Reasoning about the unseen for efficient outdoor object navigation. arxiv:2309.10103, 2023.

[25] S. S. Kannan, V. L. N. Venkatesh, and B.-C. Min. SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models. In 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 12140–12147, 2024. doi:10.1109/IROS58592.2024.10802322.

[26] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, L. Hou, B. Lester, et al. Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311, 2022.

[27] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[28] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

[29] M. Wu, A. Waheed, C. Zhang, M. Abdul-Mageed, and A. F. Aji. Lamini-lm: A diverse herd of distilled models from large-scale instructions. CoRR, abs/2304.14402, 2023. URL https://arxiv.org/abs/2304.14402.

[30] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. arXiv preprint arXiv:2212.08073, 2022.

[31] G. Cui, L. Yuan, N. Ding, G. Yao, B. He, W. Zhu, Y. Ni, G. Xie, R. Xie, Y. Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. arXiv preprint arXiv:2310.01377, 2023.

[32] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. Advances in Neural Information Processing Systems, 36:53728–53741, 2023.

[33] F. Song, B. Yu, M. Li, H. Yu, F. Huang, Y. Li, and H. Wang. Preference ranking optimization for human alignment. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 18990–18998, 2024.

[34] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. ICLR, 2022.

[35] C. Liu, Y. Kang, F. Zhao, K. Kuang, Z. Jiang, C. Sun, and F. Wu. Evolving knowledge distillation with large language models and active learning. arXiv preprint arXiv:2403.06414, 2024.

[36] C. Baykal, K. Trinh, F. Iliopoulos, G. Menghani, and E. Vee. Robust active distillation. arXiv preprint arXiv:2210.01213, 2022.

[37] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. arXiv preprint arXiv:2212.10560, 2022.

[38] B. Chen, C. Shu, E. Shareghi, N. Collier, K. Narasimhan, and S. Yao. Fireact: Toward language agent fine-tuning. arxiv:2310.05915, 2023.

[39] A. Zeng, M. Liu, R. Lu, B. Wang, X. Liu, Y. Dong, and J. Tang. Agenttuning: Enabling generalized agent abilities for llms. arxiv:2310.12823, 2023.

[40] D. Yin, F. Brahman, A. Ravichander, K. Chandu, K.-W. Chang, Y. Choi, and B. Y. Lin. Lumos: Learning Agents with Unified Data, Modular Design, and Open-Source LLMs. arXiv preprint arXiv:2311.05657, 2023.

[41] Y. Kong, J. Ruan, Y. Chen, B. Zhang, T. Bao, S. Shi, G. Du, X. Hu, H. Mao, Z. Li, X. Zeng, and R. Zhao. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems, 2023. URL https://arxiv.org/abs/2311.11315.

[42] W. Choi, W. K. Kim, M. Yoo, and H. Woo. Embodied cot distillation from llm to off-the-shelf agents. arXiv preprint arXiv:2412.11499, 2024.

[43] S. Qiao, N. Zhang, R. Fang, Y. Luo, W. Zhou, Y. E. Jiang, C. Lv, and H. Chen. AUTOACT: automatic agent learning from scratch via self-planning. CoRR, abs/2401.05268, 2024. doi:10.48550/ARXIV.2401.05268. URL https://doi.org/10.48550/arXiv.2401.05268.

[44] J. Liu, C. Xu, P. Hang, J. Sun, M. Ding, W. Zhan, and M. Tomizuka. Language-driven policy distillation for cooperative driving in multi-agent reinforcement learning, 2024. URL https://arxiv.org/abs/2410.24152.

[45] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, and D. Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 15172–15179, 2024. doi:10.1109/ICRA57147.2024.10610455.

[46] Z. Teng, Y. Song, X. Ye, and Y. Ouyang. Fine-tuning llms for multi-turn dialogues: optimizing cross-entropy loss with kl divergence for all rounds of responses. In Proceedings of the 2024 16th International Conference on Machine Learning and Computing, pages 128–133, 2024.

[47] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, et al. Robots that ask for help: Uncertainty alignment for large language model planners. arXiv preprint arXiv:2307.01928, 2023.

[48] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[49] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. arXiv preprint arXiv:1712.05474, 2017.

[50] Z. Ravichandran, A. Robey, V. Kumar, G. J. Pappas, and H. Hassani. Safety guardrails for llm-enabled robots. arXiv preprint arXiv:2503.07885, 2025. URL https://arxiv.org/abs/2503.07885.

[51] 2023. URL https://wiki.linuxfoundation.org/networking/iproute2.

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

[53] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. Lang2ltl: Translating natural language commands to temporal robot task specification. In Conference on Robbot Learning (CoRL), 2023. URL https://arxiv.org/abs/2302.11649.

[54] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan. Nl2tl: Transforming natural languages to temporal logics using large language models. arXiv preprint arXiv:2305.07766, 2023.

[55] A. Robey, Z. Ravichandran, V. Kumar, H. Hassani, and G. J. Pappas. Jailbreaking llm-controlled robots. arXiv preprint arXiv:2410.13691, 2024.

[56] B. Quartey, E. Rosen, S. Tellex, and G. Konidaris. Verifiably following complex robot instructions with foundation models. arxiv:2402.11498, 2024.

[57] Z. Yang, S. S. Raman, A. Shah, and S. Tellex. Plug in the safety chip: Enforcing constraints for llm-driven robot agents. arxiv:2309.09919, 2023.

[58] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. Neural Information Processing Systems, 2022.

[59] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In Conference on robot learning, pages 894–906. PMLR, 2022.

Appendix

Table of Contents

A Summary of Appendices **15**

B LLM Distillation Implementation **15**

C SPINE **15**
- C.1 Action primitives . . . 15
- C.2 Output format . . . 15
- C.3 Observation space . . . 16
- C.4 Tasks . . . 16

D SayCan **16**
- D.1 Action primitives . . . 16
- D.2 Output format . . . 16
- D.3 Observation space . . . 17
- D.4 Tasks . . . 17

E LLM-Planner **17**
- E.1 Action primitives . . . 18
- E.2 Observation space . . . 19
- E.3 Output format . . . 19
- E.4 Tasks . . . 19

F Scenario generation LLM **19**
- F.1 SPINE data generator prompt . . . 19
- F.2 LLM-Planner data generator prompt . . . 20
- F.3 SayCan data generator prompt . . . 21

G Example synthesized environments, tasks, and plans **22**
- G.1 SPINE synthesized data . . . 22
- G.2 SPINE elicited plan . . . 24
- G.3 LLM Planner synthesized data . . . 24
- G.4 LLM Planner elicited plan . . . 25
- G.5 SayCan synthesized data . . . 25
- G.6 SayCan elicited plan . . . 26

A Summary of Appendices

We report the implementation details of our LLM distillation procedure in §E. We then provide further detail on the action primitives, output format, observation space, and evaluation tasks for the three planners used in our evaluation — SPINE (§C), SayCan (§D) and LLM-Planner (§E). We then provide the prompts used for the scenario generation LLM as applied to each planner (§F). Finally, we provide examples of synthesized environments and elicited plans for each planner (§G).

B LLM Distillation Implementation

We use the Unsloth[1] Library, which is built on top of the Huggingface Transformers library and PyTorch. We report hyperparameters used for distilling SLMs via supervised fine tuning in Tab. 4

Parameter	SPINE	SayCan	LLM Planner
learning rate	1e-4	1e-4	2e-4
weight decay	1e-3	1e-3	5e-2
epochs	5	5	5
warmup steps	5	5	5
lora rank	32	32	16
lora alpha	16	16	16

Table 4: SLM hyperparameters

C SPINE

SPINE [9]: The SPINE planner is designed for language-driven mapping, navigation, and exploration in partially-known and unstructured environments. The planner operates in a closed-loop manner: actively mapping its scene and updating its plan based on its findings.

C.1 Action primitives

SPINE composes plans with the following action primitives.

1. `map_region(region_node)`: Go to a region and search or objects.
2. `inspect(object_node, query)`: Inspect an object for attributes described in the query.
3. `explore_region(region_node, exploration_radius)`: Go to a region and explore within a given radius.
4. `goto(region_node)`: goto a region node.
5. `answer(message_to_user)`: Terminate task and provide a message to the user.

C.2 Output format

At each iteration, the LLM provides its primary goal (`primary_goal`) — which is particularly relevant for under-specified tasks — relevant portions of the semantic graph for achieving that goal (`relevant_graph`), its `reasoning` for its plan, and the `plan`, all via JSON

```
{"primary_goal": "",
"relevant_graph": "",
"reasoning": "",
"plan": ""}
```

[1] https://unsloth.ai/

C.3 Observation space

The observation space is a semantic graph with nodes of type `regions` and `objects` and edges defined between and object and region (`object_edges`) or two regions (`region_edges`). The initial observation is provided to the LLM via the JSON format below.

```
{"regions": "",
"objects": "",
"region_connections": "",
"object_connections": ""}
```

During an episode, the planner receives updated observations via the graph-manipulation API: `add_nodes`, `remove_node`, `add_connections`, `remove_connections`, `update_robot_location`, `update_node_attributes`.

C.4 Tasks

Experimental tasks for the UAV and UGV are reported in Tab. 5, Tab. 6, and Tab. 7.

Fully-specified Indoor	Under-specified Indoor
Go to the desk	Is the path to the exit clear?
Go to the table	Find a place to sit
Inspect the chair	Find a place to work
Map the area near the desk	I want to watch a movie. Help
Go to the door	I need to do some robotics experiments
Go to the ground_1	Go to the southern part of the scene
Inspect the television	Help me find my backpack. I last saw it when I was working.
Go inspect the table then return to where you started	I lost my laptop. Help.
Inspect the plant then return to where you started	What is near the person?
Inspect the chair then inspect the table	I will be giving a tour of the lab for kids. Is there anything dangerous around?

Table 5: Indoor UGV tasks

D SayCan

SayCan [1] is a hierarchical language-guided planner that combines an LLM's task-level reasoning with lower-level affordance estimates. We follow the tabletop rearrangement environment. Lower level language skill primitives are implemented with CLIPort [59].

D.1 Action primitives

Actions are specified as `robot.pick_and_place(object, place)`. Objects and places must be listed in the observation. For example, "`robot.pick_and_place(red block, blue bowl)`".

D.2 Output format

Plans are sequences of actions that must end with the action "done()". Unlike the original setting where the probability of the model is evaluated for each possible action given the observed

Fully-specified Outdoor	Under-specified Outdoor
Map the path	Where can I park my car?
Go to the trees	I want a shady place to read
Inspect the car	Are the chairs under the trees occupied?
Go to sidewalk 2	Is my car damaged?
Go to parking lot 1	Is there debris on the path?
	Is construction blocking the sidewalk?
	I lost my car. Help.
	What is near the person?
	Is the path clear?
	Can I get to the parking lot?

Table 6: Outdoor UGV tasks

Fully-specified	Under-specified
Go to parking lot 1	Is there space to park my car?
Map the building	Is construction blocking the road?
Go to the canine center	Find my dog
Go to the landing strip	Has rain damaged any roofs?
Map the bike rack	Is the playground built?
Return to the landing strip	Return to land

Table 7: UAV tasks

environment (i.e. pick and place actions for all combinations of objects) , we simply prompt the LLM to output the next action to accommodate our textual input-output abstraction. For example: "`robot.pick_and_place(red block, blue bowl)`
`robot.pick_and_place(blue block, red bowl)`
`done()`"

D.3 Observation space

We list all objects with non-zero affordances as observation inputs, e.g. "`objects = [red block, blue bowl]`". Following [18] affordances are implemented with a ViLD object detector [17].

D.4 Tasks

The environment is a tabletop setting in the Pybullet simulator [48] with a UR5 robot and randomly generated placings of sets of colored blocks and bowls. We specify 10 tasks each with two different object configurations, with varying levels of difficulty and ambiguity.

E LLM-Planner

LLM-Planner [16]: The planner queries an LLM with the task specification and an object set description of the observed environment, and 9 in-context demonstrations. The demonstrations are selected via a nearest neighbor search using BERT embeddings from a small set of task-plan expert demonstrations (100 samples from training set). The LLM returns a sequence of semantic sub-goals (e.g., `Navigation fridge`, `OpenObject fridge`). A separate low-level controller then grounds each sub-goal to primitive actions in the partially observed scene.

Task	Objects
Give every bowl an odd number of blocks—or	red/green/blue block, red/blue/green bowls red/blue block, red/blue/green bowls
Put pairs of same-colored blocks into bowls. Leave single blocks.	red/red/blue/green block, red/blue/green bowls red/red/blue/yellow/yellow block, red/blue bowls
Move only the colors that appear exactly once into the yellow bowl,	red/blue/yellow block, yellow bowl
sort all remaining blocks into bowls that match their color.	red block/red block/blue block/blue block/red/blue/green/yellow bowls
Put the blue stuff together	blue/red/yellow block, blue/red/yellow bowls blue/blue/red/green block, blue/red/green bowls
Place all blocks on top of their matching bowls	red/blue/green block, red/blue/green bowls red/red/blue block, green block/red/blue/green bowls
Separate the blocks by color, putting each color group into a different bowl.	red/blue/green/green block, red/blue/green/yellow bowls red/blue/yellow/yellow block, red/blue bowls
Distribute the blocks as evenly as possible	red/blue/yellow/red block, red/blue bowls red/blue/green/yellow/red block, red/blue bowls
Stack the blocks that aren't red	red/blue/green/yellow block, blue/green/yellow bowls red/red/green block, blue/green bowls
Put blocks in bowls with non-matching colors	red/blue/green/blue/green/yellow bowls red/yellow/green block, blue/green/yellow bowls
Place every block into a bowl, ensuring no block is in a bowl of its own color.	red/blue/green block, red/blue/green/yellow bowls red/blue/green/yellow block, red/blue/green bowls

Table 8: Block–bowl manipulation tasks used for the SayCan planner. Bowl and block positions are randomized.

E.1 Action primitives

We include seven object interaction actions and a navigation action, as described in [17, Sec. 3.1] and in [17, Sec. 4.1], respectively.

1. `PickObject` — pick up (grasp) the target object move it into the agent's inventory.
2. `PutObject` — place the held object at the current focal location or receptacle (e.g., counter top, table).
3. `OpenObject` — open an openable object such as a cabinet, drawer, microwave, or door.
4. `CloseObject` — close an object previously opened by `OpenObject`.
5. `ToggleOnObject` — switch a toggleable object to its "on" state (e.g., faucet, microwave, stove burner).

6. `ToggleOffObject` — switch a toggleable object from "on" back to "off".
7. `SliceObject` — use a knife currently held to slice a slice-able object (e.g., apple, bread, potato) that is specified by the mask.
8. `Navigation` — discretised egocentric movement and viewpoint-control primitives: `MoveAhead`, `RotateLeft/Right`, `LookUp/Down`, which translate or rotate the agent without interacting with objects.

E.2 Observation space

The planner recieves a string with visible objects, e.g. "`Visible objects: fridge, cabinet, countertop`".

E.3 Output format

Comma-sepparated string of `Action Object` pairs, e.g.: *"Navigation microwave, OpenObject microwave, PutObject egg microwave, CloseObject microwave, ToggleObjectOn microwave"*

E.4 Tasks

We consider tasks from Action Learning From Realistic Environments and Directives (ALFRED) [17]. We evaluate on the seven ALFRED templates:

- *Clean & Place* (e.g., "Wash the mug and put it in the cabinet")
- *Heat & Place* (e.g., "Heat the potato and put it on the counter")
- *Cool & Place* (e.g., "Cool the soda and put it on the table")
- *Examine-in-Light* (e.g., "Find the keycard and examine it under the lamp").

Environments Experiments are run in the *unseen* split of ALFRED — i.e., environments differ from the in-context examples from the training set — comprising eight AI2-THOR [49] homes with kitchens, bedrooms, bathrooms, and living rooms. The agent must explore cluttered, partially observable rooms containing up to 115 object types.

F Scenario generation LLM

F.1 SPINE data generator prompt

```
You are generating data for training an llm-based planner,
like the SPINE paper from ravichandran et al.

Generate a scene graph for training in the following format <OBSERVATION_FORMAT>
for  example, <EXAMPLE_GRAPH>

Make sure all nodes referenced in the conntections are listed in the objects and
regions list. Provide your answer in the following JSON format:
<OUTPUT_FORMAT>

Add a "description" attribute to each node that provides information.
These will be hidden from the robot

Task generation instructions
- DO NOT reference specific objects or nodes. Make the planner infer theese.
- Tasks should request specific information, not general exploration.
Make the planner map or inspect certain entities.
For example, start tasks with phrases such as "what", "I heard",
"find out", "map", "inspect", "Can I", "is there", and likewise
```

The variable OBSERVATION_FORMAT takes the form:

```
{"objects": [{"name": "object_1_name", "coords": ["west_east_coordinate",
    "south_north_coordinate"]}, "..."],
"regions": [{"name": "region_1_name", "coords": ["west_east_coordinate",
    "south_north_coordinate"]}, "..."],
"object_connections": [["object_name", "region_name"], "..."],
"region_connections": [["some_region_name", "other_region_name"], "..."]
"robot_location": "region_of_robot_location"}
```

The variable EXAMPLE_GRAPH takes the form:

```
{"objects": [
    {"name": "shed_1", "coords": [78, 9]},
    {"name": "gate_1", "coords": [52, -56]}],
"regions": [
    {"name": "ground_1", "coords": [0, 0]},
    {"name": "road_1", "coords": [5.7, -8.3]},
    {"name": "road_2", "coords": [19.3, -6.5]},
    {"name": "road_3", "coords": [35.7, -12.1]},
    {"name": "road_4", "coords": [52.7, -20]},
    {"name": "road_5", "coords": [57.2, -31.6]},
    {"name": "bridge_1", "coords": [54.3, -46.7]},
    {"name": "road_6", "coords": [52.4, -56.5]},
    {"name": "driveway_1", "coords": [78.4, 9.1]}],
"object_connections": [
    ["shed_1", "driveway_1"],
    ["gate_1", "road_6"]],
"region_connections":[
    ["ground_1", "road_1"],
    ["road_1", "road_2"],
    ["road_2", "road_3"],
    ["road_3", "road_4"],
    ["road_4", "road_5"],
    ["road_5", "bridge_1"],
    ["bridge_1", "road_6"],
    ["road_6", "driveway_1"]],
"robot_location": "ground_1"}
```

While OUTPUT_FORMAT comprises a graph, task list, and description.

F.2 LLM-Planner data generator prompt

We provide the prompt for the LLM-Planner data generator below.

```
You are a data generator for synthesizing tasks for the ALFRED simulator.

You should use objects from the following list: 'AlarmClock', 'Apple', 'ArmChair', 'BaseballBat',
↪  'BasketBall', 'Bathtub', 'BathtubBasin', 'Bed', 'Blinds', 'Book', 'Boots', 'Bowl', 'Box', 'Bread',
↪  'ButterKnife', 'Cabinet', 'Candle', 'Cart', 'CD', 'CellPhone', 'Chair', 'Cloth', 'CoffeeMachine',
↪  'CounterTop', 'CreditCard', 'Cup', 'Curtains', 'Desk', 'DeskLamp', 'DishSponge', 'Drawer',
↪  'Dresser', 'Egg', 'FloorLamp', 'Footstool', 'Fork', 'Fridge', 'GarbageCan', 'Glassbottle',
↪  'HandTowel', 'HandTowelHolder', 'HousePlant', 'Kettle', 'KeyChain', 'Knife', 'Ladle', 'Laptop',
↪  'LaundryHamper', 'LaundryHamperLid', 'Lettuce', 'LightSwitch', 'Microwave', 'Mirror', 'Mug',
↪  'Newspaper', 'Ottoman', 'Painting', 'Pan', 'PaperTowel', 'PaperTowelRoll', 'Pen', 'Pencil',
↪  'PepperShaker', 'Pillow', 'Plate', 'Plunger', 'Poster', 'Pot', 'Potato', 'RemoteControl', 'Safe',
↪  'SaltShaker', 'ScrubBrush', 'Shelf', 'ShowerDoor', 'ShowerGlass', 'Sink', 'SinkBasin', 'SoapBar',
↪  'SoapBottle', 'Sofa', 'Spatula', 'Spoon', 'SprayBottle', 'Statue', 'StoveBurner', 'StoveKnob',
↪  'DiningTable', 'CoffeeTable', 'SideTable', 'TeddyBear', 'Television', 'TennisRacket', 'TissueBox',
↪  'Toaster', 'Toilet', 'ToiletPaper', 'ToiletPaperHanger', 'ToiletPaperRoll', 'Tomato', 'Towel',
↪  'TowelHolder', 'TVStand', 'Vase', 'Watch', 'WateringCan', 'Window', 'WineBottle'

The agent has the following capabilities
OpenObject
CloseObject
PickupObject
PutObject
ToggleObjectOn
ToggleObjectOff
SliceObject
Navigation

Your response should be json with the following keys
{tasks: ["description of task 1", "description of task 2", ...],
visible objects: [list of objects in the scene],
reasoning: your reasoning}
```

F.3 SayCan data generator prompt

We provide the prompt for the LLM-Planner data generator below.

```
You are a task generator for the SayCan pick-and-place environment.
Each task should have a 'raw_input' field describing the goal and a 'config' field specifying 'pick'
↪  and 'place' lists. Output a JSON array of task objects.
Examples:
objects = [red block, yellow block, blue block, green bowl]
# put the yellow one the green thing.
robot.pick_and_place(yellow block, green bowl)
done()

objects = [yellow block, blue block, red block]
# move the light colored block to the middle.
robot.pick_and_place(yellow block, middle)
done()

objects = [blue block, green bowl, red block, yellow bowl, green block]
# stack the blocks.
robot.pick_and_place(green block, blue block)
robot.pick_and_place(red block, green block)
done()

objects = [red block, blue block, green bowl, blue bowl, yellow block, green block]
# group the blue objects together.
robot.pick_and_place(blue block, blue bowl)
done()

objects = [green bowl, red block, green block, red bowl, yellow bowl, yellow block]
# sort all the blocks into their matching color bowls.
robot.pick_and_place(green block, green bowl)
robot.pick_and_place(red block, red bowl)
robot.pick_and_place(yellow block, yellow bowl)
done()

Generate 3 tasks using 4 blocks.
```

G Example synthesized environments, tasks, and plans

G.1 SPINE synthesized data

We provide an example synthesized graph and corresponding tasks below. Please note that the output is split to enable readability.


```
{ "graph": {
"objects": [
    {"name": "bench_1", "coords": [5, 10],
    "description": "A wooden bench facing the ocean."},
    {"name": "bench_2", "coords": [15, 10],
    "description": "A wooden bench near a lamppost."},
    {"name": "lamppost_1", "coords": [10, 12],
    "description": "A tall lamppost providing light at night."},
    {"name": "food_stall_1", "coords": [20, 8],
    "description": "A stall selling ice cream and snacks."},
    {"name": "food_stall_2", "coords": [25, 8],
    "description": "A stall offering hot dogs and beverages."},
    {"name": "kite_1", "coords": [30, 5],
    "description": "A colorful kite flying high in the sky."},
    {"name": "bicycle_1", "coords": [35, 15],
    "description": "A parked bicycle near the storefronts."},
    {"name": "scooter_1", "coords": [40, 15],
    "description": "A parked electric scooter."},
    {"name": "street_performer_1", "coords": [45, 10],
    "description": "A musician playing guitar."}],
"regions": [
    {"name": "boardwalk_1", "coords": [0, 10],
    "description": "The starting point of the boardwalk."},
    {"name": "boardwalk_2", "coords": [10, 10],
    "description": "A section of the boardwalk with benches."},
    {"name": "boardwalk_3", "coords": [20, 10],
    "description": "A section of the boardwalk with food stalls."},
    {"name": "boardwalk_4", "coords": [30, 10],
    "description": "A section of the boardwalk near kite flyers."},
    {"name": "boardwalk_5", "coords": [40, 10],
    "description": "A section of the boardwalk with street performers."},
    {"name": "beach_1", "coords": [0, 0],
    "description": "A sandy beach area with children playing."},
    {"name": "beach_2", "coords": [10, 0],
    "description": "A beach area with people sunbathing."},
    {"name": "beach_3", "coords": [20, 0],
    "description": "A beach area with kite flyers."},
    {"name": "beach_4", "coords": [30, 0],
    "description": "A beach area with volleyball nets."},
    {"name": "beach_5", "coords": [40, 0],
    "description": "A quieter section of the beach."},
    {"name": "storefront_1", "coords": [0, 20],
    "description": "A row of colorful storefronts."},
    {"name": "storefront_2", "coords": [10, 20],
    "description": "A section with souvenir shops."},
    {"name": "storefront_3", "coords": [20, 20],
    "description": "A section with cafes and restaurants."},
    {"name": "storefront_4", "coords": [30, 20],
    "description": "A section with clothing stores."},
    {"name": "storefront_5", "coords": [40, 20],
    "description": "A section with art galleries."},
    {"name": "parking_lot_1", "coords": [50, 15],
    "description": "A small parking lot for cars."},
    {"name": "palm_tree_1", "coords": [5, 15],
    "description": "A tall palm tree providing shade."},
    {"name": "palm_tree_2", "coords": [25, 15],
    "description": "A palm tree near the food stalls."},
    {"name": "palm_tree_3", "coords": [35, 15],
    "description": "A palm tree near the parked bicycles."},
    {"name": "ocean_1", "coords": [0, -5],
    "description": "The ocean with gentle waves."}],
```

```
"object_connections": [
    ["bench_1", "boardwalk_2"],
    ["bench_2", "boardwalk_2"],
    ["lamppost_1", "boardwalk_2"],
    ["food_stall_1", "boardwalk_3"],
    ["food_stall_2", "boardwalk_3"],
    ["kite_1", "beach_3"],
    ["bicycle_1", "storefront_3"],
    ["scooter_1", "storefront_3"],
    ["street_performer_1", "boardwalk_5"]],
"region_connections": [
    ["boardwalk_1", "boardwalk_2"],
    ["boardwalk_2", "boardwalk_3"],
    ["boardwalk_3", "boardwalk_4"],
    ["boardwalk_4", "boardwalk_5"],
    ["beach_1", "beach_2"],
    ["beach_2", "beach_3"],
    ["beach_3", "beach_4"],
    ["beach_4", "beach_5"],
    ["storefront_1", "storefront_2"],
    ["storefront_2", "storefront_3"],
    ["storefront_3", "storefront_4"],
    ["storefront_4", "storefront_5"],
    ["boardwalk_2", "beach_2"],
    ["boardwalk_3", "beach_3"],
    ["boardwalk_4", "beach_4"],
    ["storefront_3", "parking_lot_1"],
    ["boardwalk_1", "storefront_1"],
    ["boardwalk_5", "storefront_5"],
    ["boardwalk_5", "ocean_1"],
    ["beach_5", "ocean_1"]],
"robot_location": "boardwalk_1"
{"tasks": [
    "What is the closest food stall to the starting point of the boardwalk?",
    "I heard there is a musician playing nearby. Can you find out where?",
    "Map the locations of all benches along the boardwalk.",
    "Inspect the area where kites are flying.",
    "Can I find a place to park a bicycle near the storefronts?",
    "Is there a lamppost providing light near any benches?",
    "Find out which section of the beach has volleyball nets.",
    "Map the palm trees along the boardwalk.",
    "What is the closest section of the boardwalk to the ocean?",
    "Inspect the parking lot for available spaces.",
    "Is there a food stall offering beverages nearby?",
    "Find out where the kite flyers are located.",
    "Map the sections of the boardwalk with street performers.",
    "Is there a quieter section of the beach nearby?",
    "Inspect the area with souvenir shops.",
    "What is the closest palm tree to the food stalls?",
    "Find out where the parked scooters are located.",
    "Map the sections of the boardwalk with lampposts.",
    "Is there a section of the beach with children playing?",
    "Inspect the area with cafes and restaurants.",
    "What is the closest section of the boardwalk to the parking lot?",
    "Find out where the art galleries are located.",
    "Map the sections of the beach with sunbathers.",
    "Is there a section of the boardwalk with cyclists?",
    "Inspect the area with clothing stores."
  ],
"description": "A bustling coastal boardwalk scene flanked by a sandy
beach on one side and a row of colorful storefronts on the other. The environment
includes a wooden boardwalk populated with scattered benches, lampposts,
and occasional food stalls selling snacks and beverages. Close to the boardwalk
are cyclists and pedestrians enjoying their time, with children playing
and flying kites on the beach. The ocean waves gently lap against the shore,
with occasional sailboats in the distance. Behind the storefronts, an array
of parked bicycles and scooters lines the street, while a small parking
lot accommodates cars. Tall palm trees provide intermittent shade along
the boardwalk, and street performers attract crowds. The scene is vibrant
and dynamic, with seagulls flying above and people capturing the picturesque
views on their cameras."
}
```

G.2 SPINE elicited plan

We provide the elicited plan with the above semantic graph (§G.1) for the task "'Is there a lamppost providing light near any benches?" is below. We preface plans with "// plan" and observations with "// observation".

```
// plan
{'primary_goal': 'Determine if there is a lamppost
providing light near any benches.',
'relevant_graph': 'boardwalk_1, lamppost_1',
'reasoning': 'I cannot reach lamppost_1 from my current location, boardwalk_1.
I will map boardwalk_1 to try to find a connection.',
'plan': '[map_region(boardwalk_1)]}

// observation
add_nodes({boardwalk_2: {coords: [10.0, 10.0], type: region, description:
A section of the boardwalk with benches., name: boardwalk_2}},
{storefront_1: {coords: [0.0, 20.0], type: region, description:
A row of colorful storefronts., name: storefront_1}}),
add_connections([boardwalk_1, boardwalk_2], [boardwalk_1, storefront_1]),
update_node_attributes({name: boardwalk_1, description:
The starting point of the boardwalk.})

// plan
{'primary_goal': 'Determine if there is a lamppost providing light
near any benches.',
'relevant_graph': 'boardwalk_2, lamppost_1, bench_2',
'reasoning': 'I discovered a new region, boardwalk_2, which contains benches.
I will navigate to boardwalk_2 and explore the region to
see if there is a lamppost providing light near the benches.',
'plan': '[goto(boardwalk_2), explore_region(boardwalk_2, 3)]'}

// observation
update_robot_location(boardwalk_2)

// plan
{'primary_goal': 'Determine if there is a lamppost providing light near any
benches.',
'relevant_graph': 'boardwalk_2, lamppost_1, bench_2',
'reasoning': 'I have reached boardwalk_2, which contains benches.
I will now explore the region to see
if there is a lamppost providing light near the benches.',
'plan': '[explore_region(boardwalk_2, 3)]'}

// observation
add_nodes({bench_1: {coords: [5.0, 10.0], type: object,
description: A wooden bench facing the ocean., name: bench_1}},
{boardwalk_3: {coords: [20.0, 10.0], type: region,
description: A section of the boardwalk with food stalls., name: boardwalk_3}}),
add_connections([bench_1, boardwalk_2], [bench_2, boardwalk_2],
[boardwalk_2, lamppost_1], [boardwalk_2, boardwalk_3],
[beach_2, boardwalk_2]),
update_node_attributes({name: boardwalk_2,
description: A section of the boardwalk with benches.})'

// plan
{'primary_goal': 'Determine if there is a lamppost
providing light near any benches.',
'relevant_graph': 'boardwalk_2, lamppost_1, bench_1, bench_2',
'reasoning': 'I discovered that boardwalk_2 has connections to both
bench_1 and bench_2, as well as lamppost_1. This indicates that lamppost_1
is near the benches and likely provides light to them.',
'plan': '[answer(Yes, there is a lamppost, lamppost_1,
providing light near the benches, bench_1 and bench_2, on boardwalk_2.)]'}
```

G.3 LLM Planner synthesized data

We provide an example synthesized object set and corresponding tasks below.

```
{
  "Task description": "Navigate to the DiningTable and use the Knife to slice the bread. Then, place
  ↪  the sliced pieces onto a Plate.",
  "Completed plans": [],
  "Visible objects": [
    "DiningTable",
    "Bread",
    "Knife",
    "Plate",
    "AlarmClock",
    "Desk",
    "Bed",
    "Fridge",
    "Egg",
    "CounterTop",
    "CoffeeMachine"
  ],
  "Next Plans": [
    "Navigation DiningTable",
    "PickupObject Knife",
    "SliceObject Bread",
    "PutObject Knife DiningTable",
    "PickupObject Bread",
    "Navigation Plate",
    "PutObject Bread Plate"
  ]
}
```


G.4 LLM Planner elicited plan

We provide an example elicited plan below.


```
{
    "trial": "trial_T20190908_145356_918528",
    "scene": "FloorPlan10",
    "type": "pick_heat_then_place_in_recep",
    "repeat_idx": 1,
    "goal_instr": "Put a heated apple in the trash can.",
    "initial_high_level_plans": [
      "Navigation apple",
      "PickupObject apple",
      "Navigation microwave",
      "OpenObject microwave",
      "PutObject apple microwave",
      "CloseObject microwave",
      "ToggleObjectOn microwave",
      "ToggleObjectOff microwave",
      "OpenObject microwave",
      "PickupObject apple",
      "CloseObject microwave",
      "Navigation garbagecan",
      "PutObject apple garbagecan"
    ],
    "completed_plans": [
      "Navigation apple",
      "PickupObject apple",
      "Navigation microwave",
      "OpenObject microwave",
      "PutObject apple microwave",
      "CloseObject microwave",
      "ToggleObjectOn microwave",
      "ToggleObjectOff microwave",
      "OpenObject microwave",
      "PickupObject apple",
      "CloseObject microwave",
      "Navigation garbagecan",
      "PutObject apple garbagecan"
    ],
    "failed_plans": [],
    "success": true
  }
```


G.5 SayCan synthesized data

We provide an example synthesized object set below.

```
objects = [red block, green block, blue block, red bowl, green bowl, blue bowl]
# Place all blocks on top of their matching bowls
```

G.6 SayCan elicited plan

We provide an example elicited plan below.

```
objects = [red block, green block, blue block, red bowl, green bowl, blue bowl]
# Place all blocks on top of their matching bowls
robot.pick_and_place(red block, red bowl)
robot.pick_and_place(blue block, blue bowl)
robot.pick_and_place(green block, green bowl)
done()
```