2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Lifelong Robot Learning with Human Assisted Language Planners

Meenal Parakh*,α,γ, Alisha Fong*,α,γ, Anthony Simeonovα,γ, Tao Chenα,γ, Abhishek Guptaα,β,γ, Pulkit Agrawalα,γ
αImprobable AI Lab βUniversity of Washington γMassachusetts Institute of Technology
*Authors contributed equally

***Abstract*— Large Language Models (LLMs) have been shown to act like planners that can decompose high-level instructions into a sequence of executable instructions. However, current LLM-based planners are only able to operate with a fixed set of skills. We overcome this critical limitation and present a method for using LLM-based planners to query new skills and teach robots these skills in a data and time-efficient manner for rigid object manipulation. Our system can re-use newly acquired skills for future tasks, demonstrating the potential of open world and lifelong learning. We evaluate the proposed framework on multiple tasks in simulation and the real world. Videos are available at: `https://sites.google.com/mit.edu/halp-robot-learning`**

I. Introduction

A dream shared by many roboticists is to instruct robots using simple language commands such as "clean up the sink." Large language models (LLMs) can support this dream by decomposing an abstract task into a sequence of executable actions or "skills" [12]. Several LLM-based works use a *fixed* set of skills (i.e., *skill library*) for planning [1, 11]. However, the available skills may not suffice in certain task scenarios. For instance, given the task, "clean up the sink", an LLM may plan a sequence of picks and places that move all the dishes to a dishrack. Suppose one cup contains water which must be emptied before the robot puts it away. Without access to an "empty cup" skill, the system is fundamentally incapable of achieving this task variation. On detecting failure, LLM planners may attempt to expand their abilities – the system could *request* a new skill for "pouring" if it detects water in the cup. However, unless the robot can also *execute* new skills, the problem remains unsolved.

Based on the tasks and scenarios the robot encounters, the planner must have the capacity to request and acquire *new skills*. Further, such skill acquisition ought to be *quick* – a system that requires days, weeks, or months to acquire the new skill is of little utility. Concurrent to our work, the ability of an LLM-based planner to acquire new skills has been demonstrated in the virtual domain of Minecraft [32]. However, in virtual domains, new skills can be simply represented as code that can execute high-level and abstract actions. In contrast, learning a new skill for a robot also involves finding low-level actions that can affect the physical world. To the best of our knowledge, the ability to add skills to a skill library in a time and data-efficient manner and utilize them for future tasks, especially in the context of LLM-based planners, has not been demonstrated.

Existing LLM-based robotic systems struggle with online skill acquisition because common mechanisms for learning skills (e.g., end-to-end behavior cloning or reinforcement learning) typically require a large amount of data and/or training time. Some methods are able to acquire new skills in a more data-efficient manner in limited scenarios such as in-plane manipulation (e.g., TransporterNets [35]), but these skills are insufficient for 6-DoF actions (e.g., "grasp the mug from the side", "hang the mug on a rack" or "stack a book in a bookshelf"). Another body of work such as in few-shot imitation learning can efficiently solve new instances from a task family but requires large amounts of pre-training data [7, 23] which is seldom available for new skills. We first present a method that allows LLMs to request new skills to complete the given task. Second, we propose to use Neural Descriptor Fields (NDFs) [27] to realize these new skills. We choose NDFs as they require only 5-10 demonstrations to perform rigid body manipulation in the full space of 3D translations and rotations.

Our system works by prompting an LLM with a textual scene description obtained by a perception system, a library of skills expressed as Python functions, and a natural language task specification. With this information, the LLM plans and produces a sequence of skills (in the form of code) that achieves the task. Along with the skills in the skill library, we also provide the LLM with a special function for requesting a new skill to be added to the library. When the LLM plans call this `learn_skill` function, it returns a new skill name and a docstring description of the skill. However, such a skill is abstract and is not mapped to actions. NDFs allow the user to quickly realize this new skill by providing a few demonstrations, after which the skill is added to the skill library and can be re-used on future tasks. In summary, this work demonstrates a *proof-of-concept* implementation of an LM-powered robotic planning agent that can interactively grow its skill library based on the needs of the task. We show an instance of such a system using NDFs and perform experiments that highlight the abilities of our system.

II. Related Work

a) LLMs as Zero-Shot Planners: Prior work that uses large language models (LLMs) as planners include SayCan [1], InnerMonologue [11], NLMap-SayCan [4] and Socratic Models [34]. These methods make significant contributions: [1] and [34] using LLMs as planners; [11] emphasizes the importance of feedback, and [4] improves upon [1] by introducing the ability of open-vocabulary detection for grounding using CLIP and ViLD features [24] [10]. The planners in these methods either generate the plan in textual

format or choose the next step based on a given set of actions described through text. Another set of methods [18] [31] [29] [19] using LLM as planners chose to output the plans directly using a Python or symbolic API, given the function documentation and sufficiently expressive function names.

b) End-to-End Language Conditioned Manipulation: Another class of methods processes inputs from different modalities such as vision, text, and sound, and trains models to use these inputs for predicting robot actions end-to-end (e.g., CLIPort [25], Interactive Language [20], RT1 [3], PerAct [26] and VIMA [14]). Palm-E [6] fine-tunes a large pre-trained model on multimodal sentences to output textual steps that are used as input to a small set of low-level policies. One benefit of end-to-end training is more faithful LLM grounding. However, unless very large training datasets are used, such approaches typically struggle with generalization. Finally, many of these works are limited to performing 3-DoF (top-down) manipulation actions.

c) Low-Level Robot Primitives: The modular approaches [1] [11] [4] [34] [18] [31] use a predefined set of primitive skills, often hardcoded or learned from behavior cloning. These low-level primitives can also be learned through methods such as [13], [8], [5], [35]. While these skills can be composed to perform a wide range of actions, many times a required skill cannot be composed from the primitive set and adding a new primitive may require careful engineering, or large number of demonstrations. Thus, we employ [27] to incorporate new skills at runtime using only a few demonstrations, with the only drawback of limiting the skills to known object categories.

III. Method

In the spirit of prior work on performing long-horizon tasks wherein a high-level planning algorithm chains together different low-level skills [9, 17, 21, 34], our system has explicit modules for perception, planning, and control (Fig. 1). The modularity of our system allows us to take advantage of state-of-the-art (SOTA) models like SAM [15] for segmentation and GPT-4 [22] for planning skill sequences. At a high level, our perception module describes the scene from RGB and depth observations, generating a language-based scene description containing information about the objects in the scene and the spatial relationship between them. Given the scene description and a library of skills, the planning module plans a sequence of steps to solve the task based on the scene description and task requirements. The skill sequence corresponds to a set of executable behaviors on the robot.

In contrast to previous work that uses LLMs in robotics, our planning module can request to learn a new skill when it determines that the existing skills are insufficient, and a data-efficient skill learning method can be used to extend the skill library with this new executable behavior. With an expanded skill library, the planner can utilize both the original primitive skills and the newly learned skills when completing subsequent tasks. Thus, our approach endows the system with a form of continual learning. In the following subsections, we describe each module in detail.

A. Perception

The perception module (Fig. 2a) processes RGBD images to obtain and store information about the scene objects. First, the module identifies objects using an open-vocabulary object detector [36]. We also perform segmentation to obtain object masks using SAM [15] and combine them with the depth images to obtain object point clouds. In addition to object labels, the planner may require additional information about the spatial arrangement of the scene. For example, if a robot needs to empty a mug, it first needs to know whether there is an object *in* the mug, and only execute the skill of emptying it if there is. We generate spatially-grounded scene descriptions automatically by computing positional relationships between objects using the object point clouds. A scene description that is not spatially grounded only describes the objects present in the scene, without specifying the spatial relationship between them. Lastly, to enable open-vocabulary language commands that target specific object instances, we extract CLIP embeddings of each segmented object in the scene. In this way, given a scene with multiple mugs, if the task is to "pick up the red mug," we can identify the object that matches the "red mug" description (additional examples in Appendix). Overall, our perception components output segmented object point clouds with associated detection labels, inter-object relations, and CLIP embeddings.

Spatially-grounded Textual Scene Description: To inform the planner about the environment state, we format the perception outputs into a language-based scene description with information about the scene objects and their inter-object relations. This involves constructing a string with the names of the objects along with the relations that hold between them. Please see Appendix for further details. Note that the particular method of describing the scene is not critical to our work and in the future vision-language models capable of describing objects and the relationship between them can replace this system.

B. Planning and Control

Given the language command and the textual scene description from the perception system, GPT-4 is used to plan a sequence of the steps to be executed. The inputs and outputs of the LLM are structured as follows:

a) Skill Definitions via Code API: One way to design a planner is to output a plan in natural language. However, a more machine-friendly alternative is to have the planner output programming code [18, 29], which avoids the need to map a textual plan to a robot-executable plan. In addition, communicating with LLM in a programming language allows a human to give prompts in the form of comments, docstrings, and usage examples, which helps the planner understand how each skill operates. To take advantage of these benefits, we define each skill as a Python function that takes input arguments such as object identifiers and environment locations. We provide the planner with a description and set of input/output examples for each function. The code API is initialized with a skill library $\mathcal{S}_0$ containing five

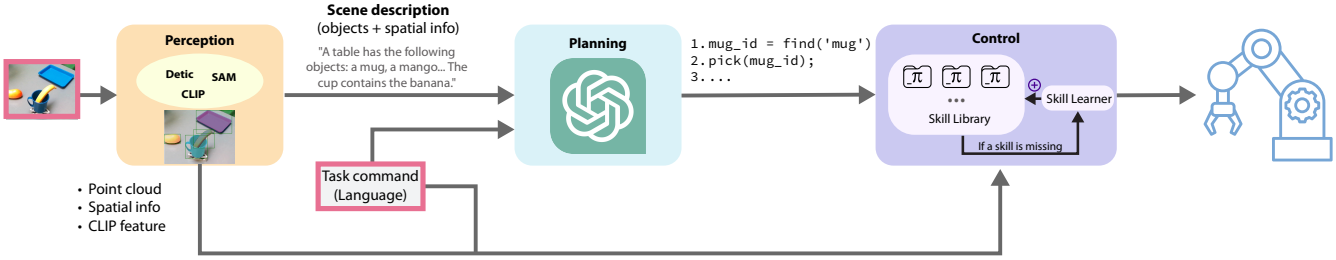

Fig. 1: Our system consists of three modules: *perception*, *planning*, and *control*. The *perception* module processes RGB-D images and outputs a textual scene description that identifies objects and their spatial relationships. The *planning* module uses GPT-4 to plan a sequence of steps based on the available skills and the task command. We added a `learn_skill(skill_name)` function to the planner so that it can plan to learn a new skill if such learning is necessary for completing the task. Finally, the *control* module executes the planned steps using the available skills or starts learning a new skill.

Fig. 2: (a) From RGBD images, our perception module obtains information about the objects and their relations, creates an object information dictionary, and generates a scene description (detection, object pairs corresponding to given object relations, and the template is in black). (b) An example showing the interaction between the robot, the user, and the planner.

primary functions: `find`, `pick`, `get_place_position`, `place`, and `learn_skill` shown in Figure 3.

The above API functions also output a signal indicating whether or not the function executes properly (i.e., to catch and correct runtime errors due to syntax mistakes). If new skills are learned (discussed in Sec. III-C), the library is updated $\mathcal{S}_i = \mathcal{S}_{i-1} \cup \{\pi_i\}$ where π_i denotes the new skill.

b) Full Planner Input/Output and Skill Execution: The planner is prompted to produce the plan in two steps. First, given the scene and task description, the planner generates a sequence of steps described in natural language. Next,

```
find(object_label=None, visual_description=None, location_description=None)
# searches with the perception system for an object based on category, visual
property, or location. Returns an object_id.

pick(object_id)
# uses Contact-GraspNet [34] to find a 6-DoF grasp for the object point cloud
associated with the object_id and executes the grasp.

get_place_position(object_id, reference_id, relation)
# for the object given by object_id, returns the (x, y, z) location determined
by the text description relation relative to reference_id.

place(object_id, place_position)
# places the object at the (x, y, z) value given in place_position.

learn_skill(skill_name)
# returns a new executable skill function and a docstring describing the skill
behavior.
```

Fig. 3: The set of primary functions.

the planner is provided with the code API of skills as discussed above and tasked to write code for executing the task using the given skills. For example, if the first step in the plan is to "find" a mug with the `find` function, the planner may output `object_id = find("mug")`. Since our system uses a LLM planner, the human user can interact with the planner at either stage of the planning to further refine the plan or correct mistakes. An example of the interaction between the user, planner, and robot is shown in Fig. 2. We qualitatively observe this two-step process helps the model generate higher-quality plans, as compared to producing the full plan directly. The two-step breakdown potentially helps in the same way "chain-of-thought" prompting has helped LLM find better responses [33].

The code returned by the LLM is executed using the `exec` construct in Python. For skills involving robot actions, the skill function calls a combination of inverse kinematics (IK), motion planning, and trajectory following using a joint-level PD controller.

C. Learning New Skills and Expanding the Skill Library

a) Requesting New Abilities with `learn_skill` function: The code API for the `learn_skill` contains a docstring detailing the role of the function and includes a few examples of the desired output of using the `learn_skill` function. The reason for providing examples is to exploit the in-context learning ability of LLMs – these examples help the LLM figure out how to use the `learn_skill` function. More details are in the Appendix. Calling `learn_skill(skill_name)` returns the handle to a new executable skill function along with a docstring that describes the behavior of the function. The function is parameterized by either one or

two `object_id`s - `target_id` for specifying which object `skill_name` acts upon, and `reference_id` for specifying a reference object for relational skills (e.g., `pick(bottle_id)` vs. `insert(peg_id, hole_id)`). The exact parameterization is decided by the LLM. The function handle and docstring returned by `learn_skill` is added to the skill library and can be reused in the future.

This skill parameterization can express skills which move a target object to some new pose, which is potentially expressed relative to the optional reference object. For example, in a "peg-in-hole" task, the peg (target object) pose would need to align with the hole (reference object) location and axis. However, this parameterization cannot represent skills involving more than two objects (e.g. placing a red bowl between two blue bowls). For certain multi-object tasks, our planner can still succeed by computing a target position relative to multiple components of the scene and calling `place`, but we leave the handling of multi-object skill learning in its full generality to future work.

One limitation of our parameterization is the category- and part-specificity of each learned skill. An alternative approach could try learning more flexible skills that incorporate several skill sub-categories (e.g., `pick("mug by handle")`, `pick(mug_id, "handle")` or `pick(mug_id, "rim")` [1, 34]). However, such generally parameterized skills are challenging to represent and would likely require more data to learn, thus limiting online skill acquisition. In contrast, by constraining skills to use a specific category and interaction part, with a target and a reference object ID taken as input, our parameterization trades off flexibility with increased performance and data efficiency.

b) Data- and time-efficient skill grounding with NDFs: Our framework is agnostic to the specific method used to ground newly learned skills into actions. It can be end-to-end learning with reinforcement learning, or behavior cloning from demonstrations. In this work, we choose to use NDFs [27] to learn new skills because it allows efficiently learning a skill from just a few ($\leq$10) demonstrations. NDFs also facilitate a degree of category-level generalization across novel object instances, as well as generalization to novel object poses due to built-in rotation equivariance. More information on NDFs can be found in [27, 28].

c) Learning from Feedback: If a task cannot be solved using available skills (such as "pick up the mug by the handle", when the available "pick" skill grasps the mug from the rim), we would expect the LLM to directly request a new skill with `learn_skill`. While this occurs the majority of the time (see Experiments Section), the planner sometimes directly attempts the task using a skill that does not satisfy the task requirements. In these cases, if a user provides the *outcome* of a task attempt (e.g., "the mug was grasped by the rim"), the planner can use this information to register its usage of an incorrect skill and subsequently call `learn_skill` to expand its abilities and reattempt the task.

This highlights the need for feedback mechanisms that, in addition to detecting runtime errors, also inform the planner about the state of the environment after skill execution. To achieve this, we allow a human operator to manually but *optionally* provide feedback before and after each step of code execution. The combination of *outcome* feedback from the user and the *execution* feedback from the skill functions enables the system to detect failures, replan and if necessary expand its skillset using `learn_skill`.

d) Continual Learning: Learning new skills allows one to execute a task that was previously not possible. However, the full potential of learning new skills is realized when we allow the system to *continually* acquire and *re-use* skills to solve future tasks. This creates a system with ever-expanding capabilities. There are many ways this can be achieved – our implementation involves simply adding a new skill function expressed as a code API to the skill library, and using the updated library for future tasks.

IV. Experiments

We design our experiments to achieve three goals: (1) Show a proof-of-concept implementation of LLM-based task planning and execution with interactive skill learning in the real world, (2) Evaluate the abilities of current LLMs to appropriately request and re-use new skills based on the needs of different manipulation tasks, and (3) Compare the performance of the system when different components (such as object relations) are included vs. removed.

In the real world, we tested our system on the Franka Panda robot with a Robotiq 2F-140 parallel jaw gripper. We used four calibrated RealSense cameras to obtain RGB-D images and point clouds. We also evaluated the LLM planner in isolation with a set of manually crafted tasks, scene descriptions, and success criteria. Additional system evaluations performed in simulation are in Appendix **??**.

A. Real-world tasks requiring `learn_skill`

We first showcase the benefits of incorporating `learn_skill`. The system is deployed to perform three tasks in the real world: (1) grasping a mug by a specific part, such as the handle, (2) placing a bottle in a container that must fit on a small shelf, and (3) emptying a mug from a "sink". Each task can be completed in multiple ways, some of which do not fulfill the full set of task requirements. Our reference point for comparison is the overall system with no feedback mechanism and no `learn_skill` capability, thus, limiting to the base set of primitive skills. Below, we discuss the differences between this baseline and the full version of our system. The full set of planner inputs/outputs for these tasks can be found in the Appendix.

1) Learning and requesting new pick and pick-place skills: **Task 1: Grasp mug by handle** Our warm-up task that highlights how learning new skills can benefit our system is to perform grasping by a specific part. In this case, we ask the system to "grasp the mug by the handle" (see Fig. 4A). Without `learn_skill`, the planner directly calls `pick` on the mug. This triggers a grasp detector [30] to output a set of grasps on the corresponding mug point cloud. Since most of these grasps are along the rim of the mug, the robot executes a grasp along the rim of the mug, and the task finishes.

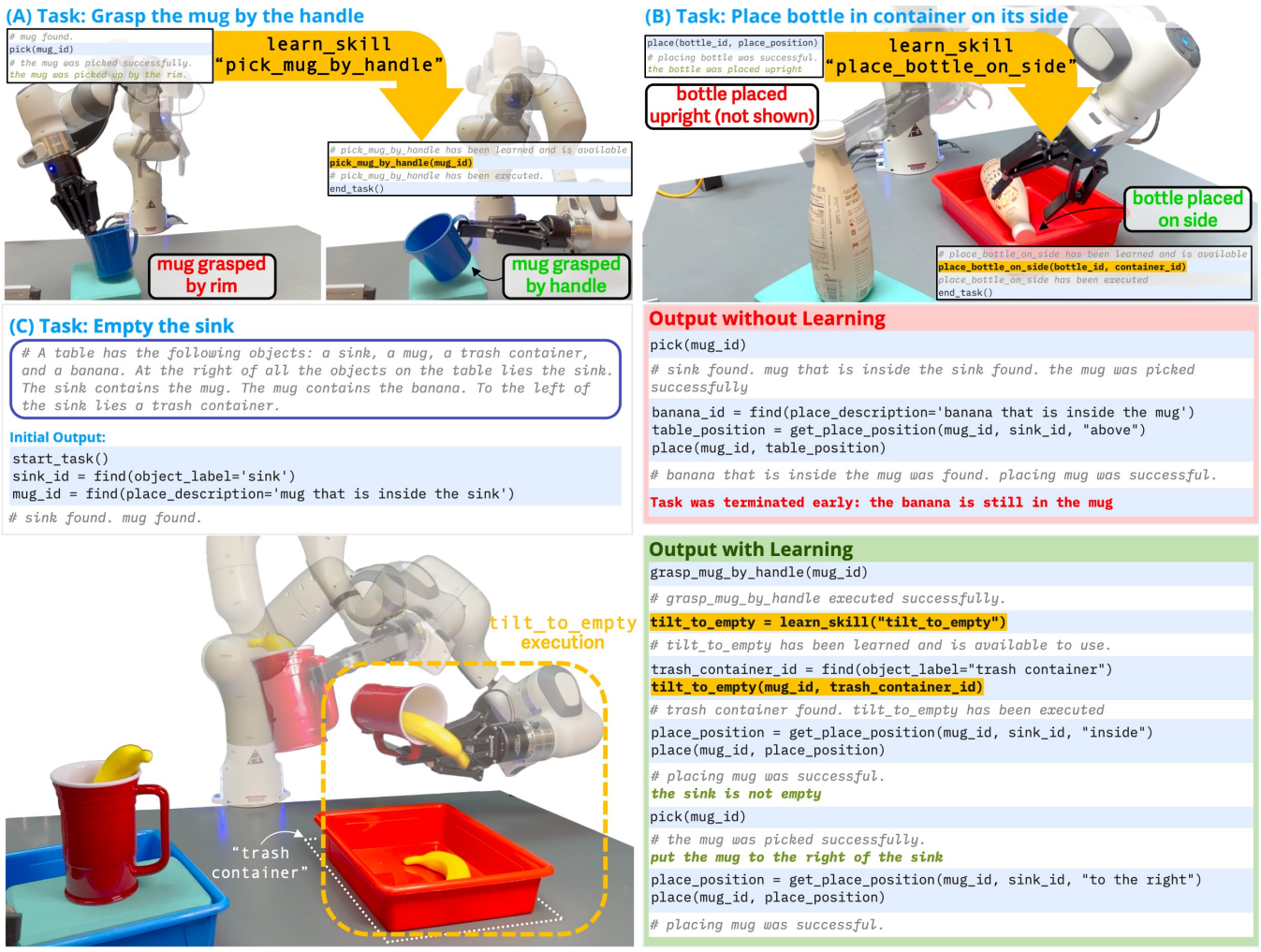

Fig. 4: High-level plan and images for three tasks requiring a new skill: (A) Grasp mug by the handle, (B) Place bottle in container on its side, and (C) Empty the sink. The `# gray comments` represent execution feedback, `green text` is human feedback, and **`blue highlights`** are code outputs from LLM. When `learn_skill` is not available, the robot fails to complete the tasks. However, by learning new skills, the planner expands its abilities and satisfies each task requirement.

If an incorrect skill is used, the human can provide feedback. By telling the system "the mug was picked up by the rim", the planner puts the mug down and requests to learn `pick_mug_by_handle`. We teach this as a side-grasp at the handle using NDFs with five demos, after which `pick_mug_by_handle` is added to the library. The LLM then calls `pick_mug_by_handle` and finishes the task successfully.

Task 2: Place bottle in flat tray Our next task is to place a bottle in a container that must eventually fit in a small shelf. Here, we prompt the system to "place the bottle sideways in the container" (see Fig. 4B). When the pipeline runs using the base set of skills, the robot uses the only available "place" skill, which places the bottle upright in the tray.

Instead, given feedback "the bottle was placed upright in the tray", the LLM calls `learn_skill` to acquire `place_bottle_sideways_in_tray`. This is implemented via NDFs as a side grasp on the bottle along with a reorientation and placement inside of the tray. Once this new skill has been added, the robot is able to successfully complete the task.

2) Continual learning by re-using previously-learned skills: **Task 3: Empty mug from sink** Finally, we prompt the system with the abstract objective of emptying a "sink" by removing a mug from the container and placing it on the table (see Fig. 4C). This task implicitly requires *emptying* the mug before placing it. We test the LLM's ability to satisfy this requirement by placing an additional small object (banana) inside the mug (ensuring the object is at least visible by the cameras, but difficult to pick up directly). The baseline system directly calls a combination of `pick` on the mug and `place` to put the mug down on the table.

However, with access to `learn_skill` and the dynamic skill library, the planner *reuses* `pick_mug_by_handle` learned in Task 1 and immediately requests to learn `tilt_mug` so it can dump objects from the mug into the trash container. We again use NDFs to teach `tilt_mug`, which reorients the mug above the tray. After emptying, the system places the mug back *into* the sink. The user provides the system with feedback that the sink is still not empty, after which the LLM re-plans and achieves the final placement on the table.

B. LLM-only skill learning evaluation

In this section, we examine the isolated ability of the LLM-planner to utilize the `learn_skill` function and to appropriately re-use and/or *not* re-use newly-learned skills on subsequent runs. This enables further analysis of GPT-4's ability to interpret manipulation scenarios represented via textual scene descriptions and correctly use the available skills provided in the code API. For each task in the following subsections, we provide a manually-constructed scene description (that does not correspond to any particular real-world scene) along with a task prompt and the skill API. We ask the planner to output code that completes the task using the API functions. The code output is manually evaluated as correct/incorrect by a human.

Requesting new skills First, we study the ability to properly (i) call `learn_skill` or (ii) *not* call `learn_skill`, for a variety of tasks where either the base skill set is (ii) or is not (i) insufficient for the task, respectively. We report the fraction of attempts that correctly use or ignore `learn_skill` in a scenario where human feedback is not provided. The results are shown in the top two rows of Table I. The 91% success rate for using `learn_skill` without feedback indicates GPT-4 can effectively expand its skill set in a purely feed-forward fashion. Similarly, the LLM usually does not call `learn_skill` when it is not needed (87% success). However, some performance gap remains in both settings.

Varying skill docstring level of detail Next, we focus on the ability to properly re-use the previously-learned skills on subsequent runs, when they can either be applied or when they should *not* be applied (e.g., in scenarios where they are inappropriate or infeasible). We consider varying levels of detail in the description that accompanies the newly-learned skill as it is added to the code API. For instance, we can provide minimal information and only add the name of the new skill, or we can modify the return values of `learn_skill` so that the LLM writes its own docstring/function description to accompany the new skill when we add it to the API. The results are shown in the last two rows of Table I. The success rates indicate that the language model correctly uses the newly-learned skills with higher frequency when the skill descriptions also include docstrings. This makes intuitive sense, as it provides extra context for both the ability and applicability of the newly learned skill, which the LLM can attend to when generating the output code for executing the task (mimicking the chain-of-thought and "let's think step-by-step" improvements observed in prior work [16, 33]).

Despite the performance increase when describing newly-added skills in more detail, the LLM only achieves moderate overall performance (75% success rate). We observe this is due to a combination of sometimes using new skills when they should not be used (e.g., calling a `side_pick_bottle` skill even when the scene description says "the bottle *cannot* be reached from the side") and re-learning the same skill multiple times (while occasionally calling it a very similar name) rather than directly utilizing the function that is already available in the API. We deem this as a somewhat negative result which points to potential gaps in such a method of LLM-based task planning. Namely, directly outputting a sequence of high-level skills does not allow more information about the operation of high-level skills (such as scenarios when they are or are *not* applicable) to be provided or utilized during planning/reasoning.

Eval Metric	Variation	Success Rate
Correct use of `learn_skill`	–	0.91
Correctly did *not* use `learn_skill`	–	0.87
Correct re-use of new skill (varying skill description)	Name only	0.50
	Name + docstring	0.75

TABLE I: LLM-only `learn_skill` evaluation.

V. Limitations

Our system employs user feedback to recover from failures and to verify task success. However, repeated user interaction makes the system less autonomous, which also makes it difficult to run evaluation experiments at scale, and limits our evaluations to qualitative demonstrations. Leveraging learned success detectors would make the system more autonomous.

Also, parameterizing skills with `(target, reference)` allows us to use efficient algorithms (eg. NDFs) to learn skills. However, such skills may be too specialized with limited future applicability. For example, it would currently require learning two separate skills `place_bottle_sideways(bottle_id)` and `place_bottle_standing(bottle_id)` when the need arises. In contrast, learning a skill `place(bottle_id, "how-to-place")` once would allow for solving both types of placement, increasing the skill's applicability and making the system more scalable. Skill composition (e.g. [2]) may be useful for more generalized skill sets in the future.

VI. Conclusion

This paper presents a modular system for achieving high-level tasks specified via natural language. Our framework can actively request and learn new manipulation capabilities, leading to an ever-expanding set of available skills to use during planning. We show how an LLM planner can use this ability to adapt its skill set to the demands of real-life task scenarios via both feed-forward reasoning and environmental feedback. In conjunction with perceptual scene representations obtained from off-the-shelf components and a data-efficient method for learning 6-DoF manipulation skills, we provide an example of a complete system. Our results demonstrate how this combination of full-stack modularity, spatially-grounded scene description, and online learning enables a qualitatively improved ability to perform manipulation tasks specified at a high level.

VII. Acknowledgement

We thank the members of Improbable AI for their feedback. This work is supported by Sony, Amazon Robotics Research Award, the US Government, and ARO MURI W911NF-23-1-0277. A. Simeonov is supported in part by an NSF Graduate Research Fellowship.

REFERENCES

[1] Michael Ahn et al. "Do As I Can and Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691*. 2022.

[2] Anurag Ajay et al. "Is Conditional Generative Modeling all you need for Decision Making?" In: *The Eleventh International Conference on Learning Representations*. 2023.

[3] Anthony Brohan et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv preprint arXiv:2212.06817* (2022).

[4] Boyuan Chen et al. "Open-vocabulary queryable scene representations for real world planning". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023.

[5] Cheng Chi et al. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2023.

[6] Danny Driess et al. "PaLM-E: An Embodied Multimodal Language Model". In: *arXiv preprint arXiv:2303.03378*. 2023.

[7] Yan Duan et al. "One-shot imitation learning". In: *Advances in neural information processing systems* 30 (2017).

[8] Pete Florence et al. "Implicit behavioral cloning". In: *Conference on Robot Learning*. PMLR. 2022.

[9] Caelan Reed Garrett et al. "Integrated task and motion planning". In: *Annual review of control, robotics, and autonomous systems* 4 (2021).

[10] Xiuye Gu et al. "Open-vocabulary Object Detection via Vision and Language Knowledge Distillation". In: *International Conference on Learning Representations*. 2021.

[11] Wenlong Huang et al. "Inner Monologue: Embodied Reasoning through Planning with Language Models". In: *Conference on Robot Learning*. PMLR. 2023.

[12] Wenlong Huang et al. "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents". In: *International Conference on Machine Learning*. PMLR. 2022.

[13] Eric Jang et al. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *5th Annual Conference on Robot Learning*. 2021.

[14] Yunfan Jiang et al. "VIMA: General Robot Manipulation with Multimodal Prompts". In: *arXiv preprint arXiv: Arxiv-2210.03094* (2022).

[15] Alexander Kirillov et al. "Segment anything". In: *arXiv preprint arXiv:2304.02643* (2023).

[16] Takeshi Kojima et al. "Large language models are zero-shot reasoners". In: *Advances in neural information processing systems* 35 (2022).

[17] John Leonard et al. "Team MIT urban challenge technical report". In: (2007).

[18] Jacky Liang et al. "Code as policies: Language model programs for embodied control". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023.

[19] Kevin Lin et al. "Text2Motion: From Natural Language Instructions to Feasible Plans". In: *arXiv preprint arXiv:2303.12153* (2023).

[20] Corey Lynch et al. *Interactive Language: Talking to Robots in Real Time*. 2022. arXiv: 2210.06407 [cs.RO].

[21] Michael Montemerlo et al. "Junior: The stanford entry in the urban challenge". In: *Journal of field Robotics* 25.9 (2008).

[22] R OpenAI. "GPT-4 technical report". In: *arXiv* (2023).

[23] Deepak Pathak* et al. "Zero Shot Visual Imitation". In: *International Conference on Learned Representations* (2018 (*equal contribution)).

[24] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[25] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "CLIPort: What and Where Pathways for Robotic Manipulation". In: *Proceedings of the 5th Conference on Robot Learning (CoRL)*. 2021.

[26] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation". In: *Proceedings of The 6th Conference on Robot Learning*. Ed. by Karen Liu, Dana Kulic, and Jeff Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2023, pp. 785–799.

[27] Anthony Simeonov et al. "Neural descriptor fields: Se (3)-equivariant object representations for manipulation". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022.

[28] Anthony Simeonov et al. "Se (3)-equivariant relational rearrangement with neural descriptor fields". In: *Conference on Robot Learning*. PMLR. 2023.

[29] Ishika Singh et al. "Progprompt: Generating situated robot task plans using large language models". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023.

[30] Martin Sundermeyer et al. "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021.

[31] Sai Vemprala et al. *ChatGPT for Robotics: Design Principles and Model Abilities*. Tech. rep. MSR-TR-2023-8. Microsoft, 2023.

[32] Guanzhi Wang et al. "Voyager: An Open-Ended Embodied Agent with Large Language Models". In: *arXiv preprint arXiv: Arxiv-2305.16291* (2023).

[33] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022).

[34] Andy Zeng et al. *Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language*. 2022. arXiv: 2204.00598 [cs.CV].

[35] Andy Zeng et al. "Transporter Networks: Rearranging the Visual World for Robotic Manipulation". In: *Conference on Robot Learning (CoRL)* (2020).

[36] Xingyi Zhou et al. *Detecting Twenty-thousand Classes using Image-level Supervision*. 2022. arXiv: 2201.02605 [cs.CV].