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

How to Prompt Your Robot:
A PromptBook for Manipulation Skills with Code as Policies

Montserrat Gonzalez Arenas, Ted Xiao, Sumeet Singh, Vidhi Jain, Allen Ren, Quan Vuong, Jake Varley,
Alexander Herzog, Isabel Leal, Sean Kirmani, Mario Prats, Dorsa Sadigh, Vikas Sindhwani, Kanishka Rao, Jacky Liang, Andy Zeng

Google DeepMind

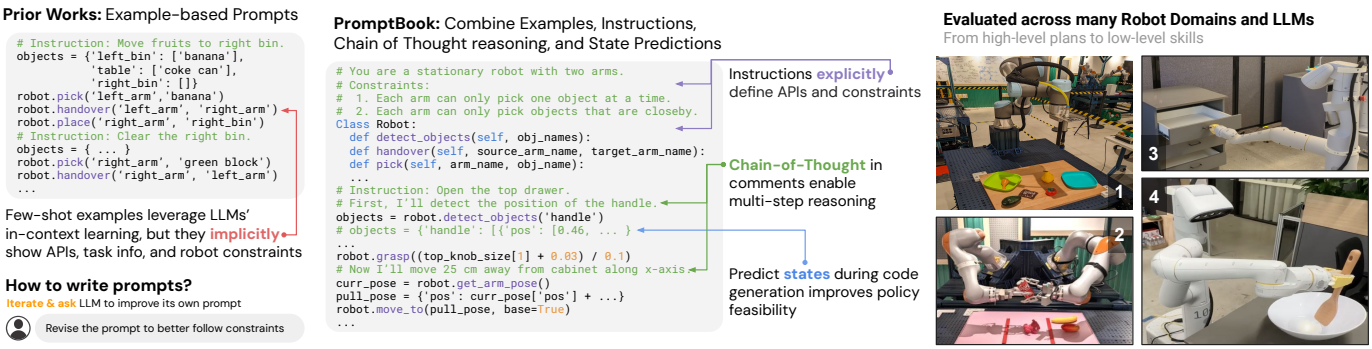

Figure 1: We present PromptBook, a recipe to combine CaP example-based prompts (left) with instructions that describe APIs and constraints, Chain-of-Thought reasoning, and world state information. Experiments show that PromptBook has higher planning success rates across multiple robots task domains (right), and interestingly give rise building new motion primitives code on-the-fly with LLMs (e.g., from picking to drawer opening and whisking policies, zero-shot).

***Abstract*—Large Language Models (LLMs) have demonstrated the ability to perform semantic reasoning, planning and write code for robotics tasks. However, most methods rely on pre-existing primitives (i.e. pick, open drawer) or similar examples of robot code alone, which heavily limits their scalability to new scenarios. We present PromptBook, a collection of different prompting paradigms to generate code for successfully executing new manipulation skills. We demonstrate example-based, instruction-based and chain-of-thought to write robot code; as well as a method to build the prompt leveraging LLMs and human feedback. We show PromptBook enables LLMs to write code for new low-level manipulation skills in a zero-shot manner: from picking diverse objects, opening/closing drawers, to whisking, and waving hello. We evaluate the new skills on a mobile manipulator with 83% success rate at picking, 50-71% at opening drawers and 100% at closing them. Notably, the LLM is able to infer gripper orientation for grasping a drawer handle (z-axis aligned) vs. a top-down grasp (x-axis aligned).**

I. Introduction

Large language models (LLMs) exhibit a wide range of capabilities that can be used on robots – from high-level planning [1–4] and logical reasoning [5–8], to writing robot code [9, 10]. Methods such as Code as Policies (CaP) [9], enable LLM-based planning to express functions, logic structures, and feedback loops that can process perception inputs (e.g., from open-vocabulary object detectors) and parameterize control primitive APIs. CaP provides code-writing LLMs with several examples of language commands (formatted as comments) followed by corresponding policy code (via few-shot prompting, in gray), then take in new commands (in green) and autonomously to generate new code (highlighted) respectively:

```
# if you see an orange, move backwards.
if detect_object("orange"):
    robot.set_velocity(x=-0.1, y=0, z=0)
# move left until you see the apple.
while not detect_object("apple"):
    robot.set_velocity(x=0, y=-0.1, z=0)
```

This matches how LLMs are often trained in-context [11] (i.e., referred to as supervised meta-learning [11–13]): the few-shot input-output examples serve as a form of task specification, where the autoregressive model is expected to complete further instances of the task by predicting what comes next. However, *example-based prompting* can be limiting as it requires an extensive number of examples that cover a wide range of properties and constraints that the robot needs to follow. More recently, there has been growing interest in *instruction-based prompting* – from generating PDDL plans [14], to synthesizing reward functions for predictive control [15] – where the model is provided with a brief language description of the robot, its constraints, and accessible APIs, then expected to directly complete new robot code given a new task (often zero-shot). Instruction-based prompts can benefit from LLMs that are specifically fine-tuned to follow instructions [16], where the prompts resemble training data from web tutorials, README files, and API documentation. In robotics, we can use instruction-based prompts to explicitly describe the robot's embodiment, function definitions, output formats and coordinate frame conventions:

```
There is a mobile robot with a camera.
Constraints:
    1. Its default speed is 0.1 m/s.
    ...
class Robot:
    def detect_object(self, object_name):
    def set_velocity(self, x, y, z):
    ...
 # move left until you see the apple.
while not detect_object("apple"):
    robot.set_velocity(x=0, y=-0.1, z=0)
```

While both prompting methods have shown promising results for generating robot code, the choice of prompting has often been arbitrary in prior work. In this work, we provide systematic evaluations to investigate the distinctions between prompting methods across various robot setups (including 3 distinct real hardware platforms: a stationary single-arm pick and place task, bi-arm pick and place, and a single-arm mobile manipulator) and language models (including an in-context pre-trained PaLM 2-L, the same LLM instruction fine-tuned, and a smaller in-context pre-trained 24B LLM fine-tuned on code data). Our experiments suggest several interesting (and to an extent, surprising) findings, including that:

- Instruction-tuned models work surprisingly well with example-based prompting, at times exceeding the overall performance of non-instruction-tuned models.
- Including both instructions and examples in the CaP prompt yields the best of both worlds – performance improvements (in terms of planning success rates) can be observed across all language models including smaller ones.
- Robot constraints (such as reachability) can be explicitly specified via instruction-based prompting, whereas they tend to be implicitly demonstrated via example-based prompting. In quantifying errors specifically related to constraints, instruction-tuned models perform better with instruction-based prompts, while non-instruction-tuned models perform better with example-based prompts.
- On more complex settings with a mobile manipulator (similar to the setting in Herzog et al., [17]), we observe that both modes of prompting can struggle to return strong performance. Here, experiments show that (i) instruction-based prompting benefits from human feedback corrections, to which the LLM can be instructed to improve its own prompt and (ii) examples in the prompt may benefit from interleaving linguistic descriptions of predicted robot states (e.g., from a VLM) between lines of generated robot code.

Experiments show that taking the best out of these comparisons yields PromptBook: combining instruction-based and example-based prompting with feedback and interleaved states leads to more robust planning performance in generating robot code. Interestingly, this prompting method gives rise to new ways of building motion primitives (that combine low-level perception and control APIs) in ways that can transfer to new primitives – providing the capacity execute entirely new skills such as "open/close the drawer" or "open the fridge door" without any additional human intervention, data collection, or model training. These capabilities are not sufficient today to replace specialized algorithms, but nevertheless offer a glimpse of the capacity of LLMs to compose motion primitives for low-level skills. We provide comprehensive details on the setup, metrics, and analysis in Section IV together with notes and discussions (tips and tricks) for prompt engineering LLMs on robots that may be useful for other practitioners of Code as Policies. More information will be made available at promptbook.github.io

II. Related Work

Language is a powerful way to encode our high-level intent, actions, and commonsense priors about the world. Large language models have shown capabilities in various reasoning and robotics tasks, ranging from writing math programs [18], thinking step-by-step [19], zero-shot generation of high-level task plans [1, 2, 20, 21], designing reward functions [15, 22, 23], and even reasoning about multimodal inputs such as gestures [24] or demonstrations [25]. While LLMs trained on web-scale text corpora already show impressive results, they can still be limited when generalizing to new settings. Instruction-tuned models [26, 27] show substantially improved zero-shot performance on unseen tasks [16, 28], even with very selected instruction-response pairs [29]. Prior work on following language instructions and feedback focused on either mapping language directly to motion primitives [30–33], or learning language-conditioned policies [34–40].

Example-based prompting is a promising paradigm for general in-context learning [41, 42], and has been effective in robotics applications [9, 43]. [10] uses example function definitions to prompt LLMs and code for tabletop tasks. [21] uses examples based prompting for the high-level task planning, followed by learned low-level policies. In contrast, our work presents code examples for both task and motion planning. By providing these example-based prompts, we are *implicitly* guiding the LLM with our preferences to control the robot.

Instruction-based prompting has been explored in a number of prior work [15, 44–46]. This includes iterative prompting approaches that integrate environment feedback, execution errors, and self-verification [45, 46] or prompt design techniques for knowledge acquisitions and progress monitoring [47]. In addition, recent work consider automatic approaches for generating prompts that exhibit strategic and chain of thought reasoning [48, 49]. While these techniques provide powerful approaches for instruction-based prompting, they have not considered real robot settings.

In our work, we investigate in-context learning approaches in LLMs by prompting through both API guidelines and examples.

III. The PromptBook Recipe

The high-level problem our work tackles is enabling robots to perform a wide variety of tasks specified by natural language. To do so, we leverage Large Language Models (LLMs) to map natural language instructions to robot code [9]. While code-writing LLMs have the innate ability to write generic code leveraging elements such as control structures (e.g., for loops) and popular third-party libraries (e.g., NumPy), they cannot directly write domain-specific robot code, which involve concepts and APIs not included in the LLMs' training data. Some knowledge specification or grounding is required in order for general purpose LLMs to produce performant domain-specific robot code. As such, we need to teach the LLM how to write robot code to solve robotics tasks. This is done through *prompting* the LLM, which is prepending input task commands with a text "prompt" that steers the LLMs' autoregressive generation behavior, so it generates the desired output code.

We propose PromptBook, an LLM prompting recipe for improving LLM robot code generation. In the sections that follow, we will first detail the PromptBook recipe, then we will give guidelines on how to apply PromptBook for specific robot task domains. See Figure 1 for an overview of our approach.

A. PromptBook Elements

The PromptBook recipe contains 7 elements:

1) Examples. We can teach the LLM how to write robot code via packing a list of examples in prompt, where each example is a command-response pair. For code-writing examples, we can format each command as a comment, and each response as the code block that immediately follows:

```
# if you see an orange, move backwards.
if detect_object("orange"):
    robot.set_velocity(x=-0.1, y=0, z=0)
```

We refer to prompts with only examples as **Example-Based Prompting**, which implicitly shows the LLM two types of information: how to ground robot commands (e.g., how to map spatial descriptions like "backwards" to code) and how to use first-party APIs (e.g., custom robot action functions not seen in the LLM's training set, such as `robot.set_velocity`). Example-based prompting is conceptually intuitive for autoregressive in-context learning, and it is used in the Code as Policies work [9].

However, some concepts are difficult to teach to LLMs through examples, such as robot constraints. Suppose there is a constraint that robot speed must not exceed 1 m/s. Multiple examples of calling `robot.set_velocity` must be shown to implicitly infer that total speed, that is, L2 norm of velocity, does not exceed 1. Instead of steering LLM behavior through implicit examples, we may prompt LLMs to follow explicit natural language instructions, that describe the objectives, constraints, and other information relevant for solving the task. We refer to this as **Instruction-based Prompting**, which uses a prompt that has elements 2 through 5:

2) High-Level Robot and Task Description. Directly specifying high-level robot embodiments and task information can give the LLM useful context when performing task planning. For example, e.g., we can specify that the robot is a bimanual stationary robot, with descriptions of important environment features and task information:

```
# You are a stationary robot with a left arm and right arm placed on a table with a bin to the right and another bin to the left. # Your task is to write code that will sort objects into the correct bins as defined from given instructions. # Use your best world knowledge and make any necessary assumptions when selecting objects to sort.
```

Task descriptions are important because they explicitly define the function that the LLM should perform (e.g., writing code to move objects to desired locations). In the above example, we also tell the LLM that it is acceptable to make certain kinds of assumptions; eliciting this behavior is much more challenging with Example-Based Prompts.

3) Robot API Documentation. Beyond high-level robot and task descriptions, Instruction-Based Prompts should also include low-level details about how to write domain-specific robot code. To do so, we first provide the robot perception and action APIs, written in the style of skeleton Python code. See Figure 1 for a simplified example for a single-arm robot with a mobile base. Then, we provide additional details on the given robot APIs by directly specifying them in language, which is possible because Instruction-Based Prompts are not limited to conveying information through examples. For instance, for a robot API that uses 6D poses, we can specify the pose formats and canonical directions as follows:

```
# Pose is a dict with 'position' and 'orientation' keys in robot frame.
# The 'position' value is a 3D array of [x, y, z]  coordinates in meters.
# The 'orientation' value is a 4D array quaternion in [w, x, y, z].
# In the robot frame, directions are defined as follows:
# Positive x / Negative x: Front and Back
# Positive y / Negative y: Left / Right
# Positive z / Negative z: Upward / Downward
```

4) Robot Policy Constraints. In addition to API documentation, we can also detail robot policy constraints that are not immediately obvious from the API themselves. For example, for a bimanual robot setup, we may wish to inform the LLM on the different reachability constraints of workspace objects given the two arms, as well as additional preconditions of executing low-level pick place actions. See Figure 1 for a simplified example of such constraints in the prompt.

5) Code Guidelines. Finally, we can directly specify the desired policy code properties without relying on showing many implicit examples. Consider the requirement that the code written should strictly call the provided robot API and avoid API functions that do not exist. We can communicate requirements like this as part of the instructions:

```
# Your response should be exclusively in the form of Python code and Python-formatted comments. Do not use additional if statements or loops. Only write code that calls the provided robot API.
```

6) Chain of Thought Policy Reasoning. Beyond adding instructions to the prompt, PromptBook makes two changes to the given examples to improve LLM planning performance. The first is Chain of Thought [5] (CoT), a popular prompting method that writes the step-by-step reasoning process of solving a task in the prompt. By doing so, the LLM learns to write such thoughts when generating the answers, and this has been shown to significantly improve LLMs' reasoning capabilities We can naturally incorporate CoT in code generation by formatting each thought step as a comment in the code. See a simplified example in Figure 1.

7) Interleaved State Predictions with Policy Code. Inspired by and analogous to CoT, we also include, in each example code output, explicit environment state predictions formatted as comments after each robot action. When LLMs write robot code for multi-step long-horizon tasks, the world state changes after each step, which in turn changes what subsequent actions are feasible. If the LLM outputs only a sequence of actions, it may make mistakes by outputting ones that are made infeasible by prior actions. This is because, by only writing action code, the LLM is reasoning over changes in world state in an implicit fashion, where inferring action feasibility becomes more difficult the more actions are performed.

Instead, we can steer LLMs to predict and record current states explicitly, interleaved with the robot policy code, in its outputs. See Figure 1 for a simple example. At run time, we provide a new instruction and the initial state information, and

the LLM will autoregressively generate the remaining sequence. Explicit state predictions encourage the LLM to utilize a simple transition model for more precise planning and to better obey the given constraints.

B. How to Develop PromptBook Prompts

The final PromptBook prompt is assembled by combining the 8 prompt elements above. Different prompts are needed for different task domains and robot embodiments (similar to [9]). In this section, we provide a procedure that constructs and improve theses prompts, with the aid of LLMs. Specifically, we focus on developing a method that can leverage LLM's retrospection capabilities that leverage language feedback to improve both its immediate outputs as well as the initial prompt. We provide our 3 step process as follows:

Step 1: Initial Prompt Draft. Given a new robot task domain, a robot engineer first drafts an initial prompt following the PromptBook recipe. This initial prompt may be suboptimal, either due to insufficient domain information, or style and presentation that is difficult for the LLM to understand.

Step 2: Human-in-the-Loop Code Improvement. With the initial prompt draft ready, the robot engineer then uses the prompted LLM to perform a series of validation tasks. In this stage, the robot engineer first gives the LLM the task command, the LLM then writes the policy code, then, if the policy fails, the human engineer provides error feedback to the LLM. The feedback may include specific failure modes observed on the robot or code execution errors (e.g., "the robot grasp was too far from door handle", "the pre-grasp motion should've approached the handle vertically"). After receiving the feedback, the LLM writes improved policy code, and this process is repeated until the task is solved. At the end of each trial, we obtain a sequence of (task, code, feedback) tuples.

Step 3: LLM-aided Prompt Improvement. While human-in-the-loop feedback can reduce errors for a specific task, it is desirable to modify the prompt so these errors are not repeated in the future. We do this by giving the LLM the original prompt and the history of (task, code, feedback) tuples, then asking the LLM "How would you modify the initial prompt to avoid making this mistake in the future while keeping existing constraints?" and "How would you add a general constraint to avoid making this mistake in the future?" These modifications are then incorporated in the initial prompt to improve performance on similar tasks.

With this procedure, we are able to efficiently construct PromptBook prompts across multiple robot task domains.

IV. Experiments

Our experiments present a series of case studies that: (a) investigate how different prompting methods compare to each other across different language models, (b) show that CaP on more complex robot systems can improve by interleaving linguistic descriptions of system states in between lines of robot code, (c) study how instruction-based prompting may benefit from human language feedback, and (d) demonstrate that a combination of both instructions and examples in the prompt—and integrating takeaways from aforementioned (a) to (c)—give rise to generating code that can express entirely new motion primitives for low-level skills (i.e., LLM-scripted policies). These empirical observations and discussions in aggregate are meant to assist future practitioners of Code as Policies in discovering more effective design choices for code-writing LLM-based systems, environments, and applications.

A. Example vs. Instruction Prompting across LLMs

In this section, we compare the planning success rates of various language models using (i) example-based prompting, (ii) instruction-based prompting, and (iii) prompting with a combination of both (instructions followed by examples).

Robot Platforms. We perform these experiments on a collection of pick and place sorting tasks across 2 platforms:

- Single arm (UR5): consists of a UR5 equipped with a suction gripper overlooking a tabletop surface with objects that span kitchenware and plastic food items. A RealSense d435 camera is mounted on the wrist that captures an overhead view of the tabletop scene.
- Bi-arm Kukas (Kuka2x): a more challenging setting that consists of two Kuka IIWA 7 arms equipped with two-finger grippers overlooking a large surface area with a bin next to each arm (reachable only by that arm), as well as a shared tabletop zone that is reachable by both arms. Objects in this setting include plastic food items, wooden blocks, plush toys, and soda cans. A RealSense d435 is mounted above the shoulders of the bi-arm setup to capture an overhead view of the scene.

Task Domains. Both robots are tasked with 100 natural language instructions to sort multiple objects (randomly chosen and positioned) in the scene by their varying properties e.g., "Put the vegetables on the green plate." or "Move the soft objects to the right bin." (see Appendix for a full description of the environment setting and objects). Given the language instructions and a description of the scene (dictionary of objects and poses) as input, the language model outputs code to call motion primitive APIs that sequence pick, place, or handover actions (bi-arm only) conditioned on a target location (or object name), and arm to use (bi-arm only). Both models use open-vocabulary object detectors (e.g., OWL-ViT [50]) to locate objects in the scene.

These are simple task domains, but they are sufficient to raise key challenges behind LLM-based planning. The distinction between a single arm and bi-arm setup makes for an interesting comparison because we can evaluate an LLM's capacity to reason over reachability constraints – not only does the LLM need to reason that each arm can only reach the table or the bin nearest to it, but that moving objects from one bin to another requires taking an additional action in between (i.e., handover). Reasoning about these constraints can be *implicit* via rich examples of handing over objects, or explicit by including a language description of the constraints (e.g., "you can only reach the bin nearest to you and you must hand over objects that you cannot reach") in the instruction prompt.

Evaluated LLMs. We evaluate the different prompting methods across three LLMs: (i) in-context pre-trained vanilla PaLM

2-L (340B), (ii) instruction-tuned PaLM (Instruct-PaLM 2-L), and (iii) and a smaller code-writing language model PaLM 2-S* (24B) specifically fine-tuned on code-related tokens. We chose to use PaLM-2 for a side-by-side comparison of pre-trained LLMs with and without instruction-tuning, as well as a smaller pre-trained code-writing model (trained with the same infrastructure), which we expect should provide systems-level advantages including faster inference speeds (i.e., lower planning latency) at the cost of reduced performance.

Table I: Prompting with both instructions and examples (instr. + ex.) yields stronger success rates (%) across robot settings and language models.

	Single Arm UR5			Bi-Arm Kuka2x		
Model	instr.	ex.	instr. + ex.	instr.	ex.	instr. + ex.
PaLM 2-L	42	83	**89**	71	72	**93**
Instruct-PaLM 2-L	72	**82**	80	66	82	**94**
PaLM 2-S* (Code)	47	78	**82**	5	50	**64**

Results. See Tab. I for results showing average planning success rates between prompting methods across all tasks and LLMs. There are three main takeaways: (i) Prompting with only examples yields better performance than with only instructions. This holds true for both base LLMs and instruction-tuned LLMs. (ii) Prompting with both instructions and example generally yields the best performance. The improvement difference is more pronounced for the Bi-Arm Kuka2x domain, where it is more difficult to correctly reason about reachability constraints (see failure mode analysis below). (iii) There is no clear advantage of instruction-tuned models over base models, even for prompts that include instructions. This is surprising, but we do not observe consistent improvements for Instruct-PaLM 2-L, which is better on instruction-only prompts for the UR5 domain, but worse on the Kuka domain.

To better characterize failure modes, we additionally categorized LLM planning errors into 1 of 3 types:

- Feasibility: generated code does not respect robot constraints and either: (i) attempts to move a robot other than itself, (ii) pick up an object not there, (iii) pick up multiple object simultaneously when the gripper can only hold one, or (iv) does not respect reachability constraints (bi-arm only) in that each arm can only reach objects in its nearest bin.
- Syntax: code does not execute due to a syntax error.
- Semantic: code executes, but the task failed (i.e., objects not sorted correctly accordingly to given instructions).

Tab. II shows the ratio of planning errors categorized by types. Most errors are task planning errors (semantic) e.g., objects sorted incorrectly, or the task was incomplete. The next largest source of errors is reasoning over action feasibility. In particular, example-based prompting tends to struggle on reasoning over reachability constraints, but improves when the LLM is instruction-tuned. By contrast, instruction-based prompting performs similarly across models. There is a clear improvement when both instructions and examples are provided in the prompt, in which case both instruction-tuned and non-instruction-tuned models perform comparably – with non-instruction-tuned models still struggling more in reasoning on reachability.

Table II: Failure modes (% error, categorized by type, sum of columns per language model is total % error) across prompting methods and robot settings.

		Single Arm UR5			Bi-Arm Kuka2x		
Model	Categories	instr.	ex.	instr. + ex.	instr.	ex.	instr. + ex.
PaLM 2-L	Feasibility	1	4	0	9	16	4
	Semantic	57	13	11	20	12	3
	Syntax	0	0	0	0	0	0
Instruct-PaLM 2-L	Feasibility	0	0	0	12	6	0
	Semantic	28	18	20	21	12	6
	Syntax	0	0	0	1	0	0
PaLM 2-S (Code)*	Feasibility	6	1	0	16	28	13
	Semantic	47	21	18	79	22	23
	Syntax	0	0	0	0	0	0

Real robot execution. We also directly run the robot policy code generated by the best performing LLMs and prompting methods on both platforms for all tasks. Instr. + ex. with PaLM 2-L on the single arm UR5 yields an average execution success rate of 83%, while instr. + ex. with Instruct-PaLM 2-L on the more challenging Bi-arm Kuka2x setup yields a success rate of 59%. Common execution failure modes on the UR5 setup include handling objects dynamics e.g., (i) slipping from gripper during picking (22% of errors), rolling away on contact (67% of errors), or colliding with another object during placing (11% of errors). On the other hand, for the Bi-arm Kuka2x setup, common failure modes include: (i) perception detection failures (47% of errors), (ii) objects dropping during handover (41% of errors), (iii) grasp failures (6% of errors), (iv) or other systems errors (6% of errors). See Varley et al. [51] for a more detailed analysis of the Bi-arm Kuka2x setup.

B. Interleaving State Predictions with Policy Code

To evaluate the impact of prompting the LLM to explicitly update the robot state when writing policy code, we test two LLMs (Instruct-PaLM 2-L and GPT-4) in a mobile robot trash sorting domain (similar to [17]). Here, the robot can move among three different trash bins (landfill, recycle, and compost), and it needs to sort the trash already placed in these bins to their correct bins (e.g., plastic bottles should go in recycle).

See Tab. III for results. Without interleaved state predictions, we observe that the LLMs frequently struggle when reasoning about reachability constraints (e.g., whether the robot bin is near a trash bin before dropping off a grasped waste item). With interleaved state predictions, we see large reductions in this type of errors, and task success rate improves across both LLMs, and achieves a high of 74%. Importantly, we see larger improvements for combined instruction and example prompts over just prompting with examples.

Table III: Prompting with interleaved robot state information improves task success rate (%) for the trash sorting task across two LLMs. Improvements are most substantial when prompting with instructions and examples.

Model	ex. only	instr. + ex.
Instruct-PaLM 2-L w/o State	4	–
Instruct-PaLM 2-L w/ State	5	16
GPT-4 w/o State	8	30
GPT-4 w/ State	17	74

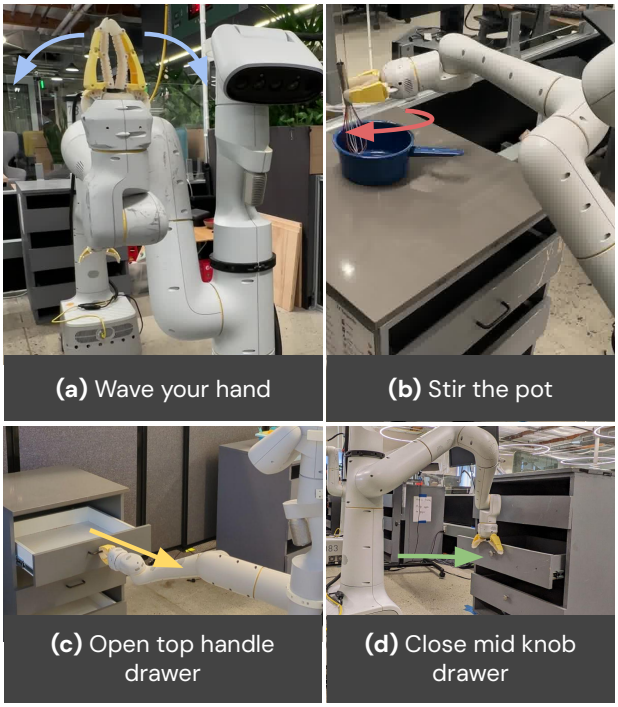

Figure 2: Examples of diverse motions generated with PromptBook. Notably, the same instruction-based prompt is re-used with only the specific low-level task instruction being swapped out for (a) waving, (b) stirring, (c) opening the top drawer by the handle, and (d) closing the middle drawer with a knob.

C. Improving Instruction-Based Prompts with Human Feedback

To evaluate the impact of iterative code improvement by human feedback, we evaluated two LLMs (Instruction PaLM 2-L and GPT-4) on the Bi-arm Kuka2x platform detailed above. Results in Tab. IV show that with only 2-3 rounds of feedback and prompt iteration, instruction-based prompting can substantially improve Bi-arm Kuka2x planning success. This improvement is consistent across the two different language models we evaluated on.

Table IV: Instruction-based prompting benefits from iterative prompt improvement with improved success rates on the Bi-arm Kuka2x planning task.

Model	instr. only	instr. + feedback	instr. + ex. + feedback
Instruct-PaLM 2-L	66	80	**93**
GPT-4	10	**99**	**99**

D. Building Low Level Motion Primitives On-The-Fly

Bringing our findings from previous sections to a practical real world low level robot control setting, we evaluate whether a single expressive PromptBook can generate novel motion primitives on the fly just by changing the input task instruction. Specifically, combining instructions and examples in the prompt (Sec. IV-A), integrating interleaved states between examples (Sec. IV-B) and using human feedback to improve the prompts (Sec. IV-C), can be thought of re-usable building blocks that can be applied across different robots, tasks, and LLMs. In this section, we demonstrate that these components together can produce expressive prompts that can generate entirely new motion primitives in a zero-shot fashion, without any robot data collection or LLM fine-tuning.

On a mobile manipulator, we build a PromptBook with a description of the robot, its constraints, robot APIs including `move_to()`, `detect_object()`, and `gripper()` functions, as well as an example of robot code for grasping an object on a countertop. This prompt can be queried to generate new motion primitives (code in Appendix) for multiple tasks, some of which are shown in Figure 2. The generated code exhibits "motion commonsense" knowledge from the LLM which are required to solve these low-level control tasks, including understanding of how a gripper should be oriented with respect to objects (e.g., vertically for a drawer handle, or horizontally for a knob). In quantitative evaluations of motion primitives generated by PromptBook on the fly, we find that policies can achieve reasonable success rates as detailed in Table V.

Table V: Real robot execution success rate of LLM-generated motion primitives zero-shot for a mobile manipulator, evaluated across 50 trials.

		Top Drawer with Handle		Middle Drawer with Knob	
Model	Pick	Open	Close	Open	Close
GPT-4	83	71	100	50	100

V. Discussions and Future Work

Our work proposes PromptBook, a guide for creating and improving prompts for new robot task domains through human and LLM feedback. We demonstrate PromptBook across three robot domains: UR5, Bimanual arms, and mobile manipulator, improving LLM robot task planning performance and synthesizing novel motion primitives. While our work investigates the trade-offs between prompting methods, the space of LLM prompting strategies and models is vast. For more complicated tasks and systems, teaching complex concepts through examples can dominate the input context to the LLM, and so does providing large APIs and instructions. We can enhance PromptBook by leveraging LLMs with longer context lengths, or efficiently selecting prompt tokens.

Our strategy to synthesize novel motions rely on robot motion primitives and objection poses information obtained from vision models. Errors in pose estimations or in the motion primitives affect the success rates of the novel motions when executed on real robots. This limitation is not specific to our method; that is, any method that relies on object pose estimation will face similar challenges. In the future, it would be interesting to consider how LLM re-planning can autonomously compensate for failures of upstream models to improve success rate.

Visit robot-promptbook.github.io for more information.

References

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv:2204.01691*, 2022.

[2] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *arXiv:2201.07207*, 2022.

[3] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman *et al.*, "Grounded decoding: Guiding text generation with grounded models for robot control," *arXiv preprint arXiv:2303.00855*, 2023.

[4] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke *et al.*, "Socratic models: Composing zero-shot multimodal reasoning with language," *arXiv:2204.00598*, 2022.

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *arXiv:2201.11903*, 2022.

[6] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *arXiv:2205.11916*, 2022.

[7] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou *et al.*, "Challenging big-bench tasks and whether chain-of-thought can solve them," *arXiv preprint arXiv:2210.09261*, 2022.

[8] A. Creswell, M. Shanahan, and I. Higgins, "Selection-inference: Exploiting large language models for interpretable logical reasoning," *arXiv preprint arXiv:2205.09712*, 2022.

[9] 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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[10] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *NeurIPS*, 2020.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[13] S. C. Chan, I. Dasgupta, J. Kim, D. Kumaran, A. K. Lampinen, and F. Hill, "Transformers generalize differently from information stored in context vs in weights," *arXiv preprint arXiv:2210.05675*, 2022.

[14] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz, "Generalized planning in pddl domains with pretrained large language models," *arXiv preprint arXiv:2305.11014*, 2023.

[15] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[16] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[17] A. Herzog, K. Rao, K. Hausman, Y. Lu, P. Wohlhart, M. Yan, J. Lin, M. G. Arenas, T. Xiao, D. Kappler *et al.*, "Deep rl at scale: Sorting waste in office buildings with a fleet of mobile manipulators," *arXiv preprint arXiv:2305.03270*, 2023.

[18] I. Drori, S. Zhang, R. Shuttleworth, L. Tang, A. Lu, E. Ke, K. Liu, L. Chen, S. Tran, N. Cheng, R. Wang, N. Singh, T. L. Patti, J. Lynch, A. Shporer, N. Verma, E. Wu, and G. Strang, "A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level," 2021.

[19] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. hsin Chi, F. Xia, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *ArXiv*, vol. abs/2201.11903, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246411621

[20] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," *arXiv preprint arXiv:2303.12153*, 2023.

[21] P. Parashar, V. Jain, X. Zhang, J. Vakil, S. Powers, Y. Bisk, and C. Paxton, "SLAP: Spatial-language attention policies," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=7Pkzm2FgUmq

[22] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," *arXiv preprint arXiv:2303.00001*, 2023.

[23] H. Hu and D. Sadigh, "Language instructed reinforcement learning for human-ai coordination," in *40th International Conference on Machine Learning (ICML)*, 2023.

[24] L.-H. Lin, Y. Cui, Y. Hao, F. Xia, and D. Sadigh, "Gesture-informed robot assistance via foundation models," in *7th Annual Conference on Robot Learning*, 2023.

[25] V. Jain, Y. Lin, E. Undersander, Y. Bisk, and A. Rai, "Transformers are adaptable task planners," in *6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=Eal_lL08v_l

[26] 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.

[27] C. Raffel, N. M. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *ArXiv*, vol. abs/1910.10683, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:204838007

[28] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," 2021.

[29] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy, "Lima: Less is more for alignment," *ArXiv*, vol. abs/2305.11206, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258822910

[30] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *AAAI*, 2011.

[31] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *Human-Robot Interaction*, 2010, pp. 259–266.

[32] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in *International Symposium on Experimental Robotics (ISER)*, 2012.

[33] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," *TACL*, 2013.

[34] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. Czarnecki, M. Jaderberg, D. Teplyashin, M. Wainwright, C. Apps, D. Hassabis, and P. Blunsom, "Grounded language learning in a simulated 3d world," *arXiv preprint arXiv:1706.06551*, 2017.

[35] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov, "Gated-attention architectures for task-oriented language grounding," in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018.

[36] F. Hill, S. Mokra, N. Wong, and T. Harley, "Human instruction-following with deep reinforcement learning via transfer-learning from text," *arXiv preprint arXiv:2005.09382*, 2020.

[37] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *CoRL*, 2022.

[38] C. Lynch and P. Sermanet, "Language conditioned imitation learning over unstructured data," *arXiv:2005.07648*, 2020.

[39] S. Yenamandra, A. Ramachandran, K. Yadav, A. S. Wang, M. Khanna, T. Gervet, T.-Y. Yang, V. Jain, A. Clegg, J. M. Turner, Z. Kira, M. Savva, A. X. Chang, D. S. Chaplot, D. Batra, R. Mottaghi, Y. Bisk, and C. Paxton, "Homerobot: Open-vocabulary mobile manipulation," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=b-cto-fetlz

[40] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. C. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. S. Ryoo, G. Salazar, P. R. Sanketi, K. Sayed, J. Singh, S. A. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. H. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich, "Rt-1: Robotics transformer for real-world control at scale," *ArXiv*, vol. abs/2212.06817, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254591260

[41] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. hsin Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," *Trans. Mach. Learn. Res.*, vol. 2022, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249674500

[42] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *ArXiv*, vol. abs/2005.14165, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218971783

[43] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng, "Large language models as general pattern machines," in *Proceedings of the 7th Conference on Robot Learning (CoRL)*, 2023.

[44] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, p. 20, 2023.

[45] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," 2023.

[46] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," *arXiv preprint arXiv:2309.03409*, 2023.

[47] Y. Ding, X. Zhang, S. Amiri, N. Cao, H. Yang, A. Kaminski, C. Esselink, and S. Zhang, "Integrating action knowledge and llms for task planning and situation handling in open worlds," *ArXiv*, vol. abs/2305.17590, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258960174

[48] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic chain of thought prompting in large language models," *arXiv preprint arXiv:2210.03493*, 2022.

[49] K. Gandhi, D. Sadigh, and N. D. Goodman, "Strategic

reasoning with language models," *arxiv*, 2023.

[50] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen *et al.*, "Simple open-vocabulary object detection with vision transformers. arxiv 2022," *arXiv preprint arXiv:2205.06230*.

[51] J. Varley, S. Singh, D. Jain, K. Choromanski, A. Zeng, S. B. R. Chowdhury, A. Dubey, and V. Sindhwani, "Embodied ai with two arms:zero-shot learning, safety and modularity," *arXiv preprint arXiv:2305.11014*, 2023.