Learning to Learn Faster from Human Feedback with Language Model Predictive Control

Jacky Liang*, Fei Xia*, Wenhao Yu*, Andy Zeng*
Montserrat Gonzalez Arenas, Maria Attarian, Maria Bauza, Matthew Bennice, Alex Bewley, Adil Dostmohamed, Chuyuan Kelly Fu
Nimrod Gileadi, Marissa Giustina, Keerthana Gopalakrishnan, Leonard Hasenclever, Jan Humplik, Jasmine Hsu, Nikhil Joshi,
Ben Jyenis, Chase Kew, Sean Kirmani, Tsang-Wei Edward Lee, Kuang-Huei Lee, Assaf Hurwitz Michaely, Joss Moore, Ken Oslund
Dushyant Rao, Allen Ren, Baruch Tabanpour, Quan Vuong, Ayzaan Wahid, Ted Xiao, Ying Xu, Vincent Zhuang
Peng Xu†, Erik Frey†, Ken Caluwaerts†,Tingnan Zhang†, Brian Ichter†, Jonathan Tompson†, Leila Takayama†, Vincent Vanhoucke†
Izhak Shafran†, Maja Mataric†, Dorsa Sadigh†, Nicolas Heess†, Kanishka Rao†, Nik Stewart†, Jie Tan†, Carolina Parada†
*corresponding authors in alphabetical order, †advising leads, all other authors in alphabetical order

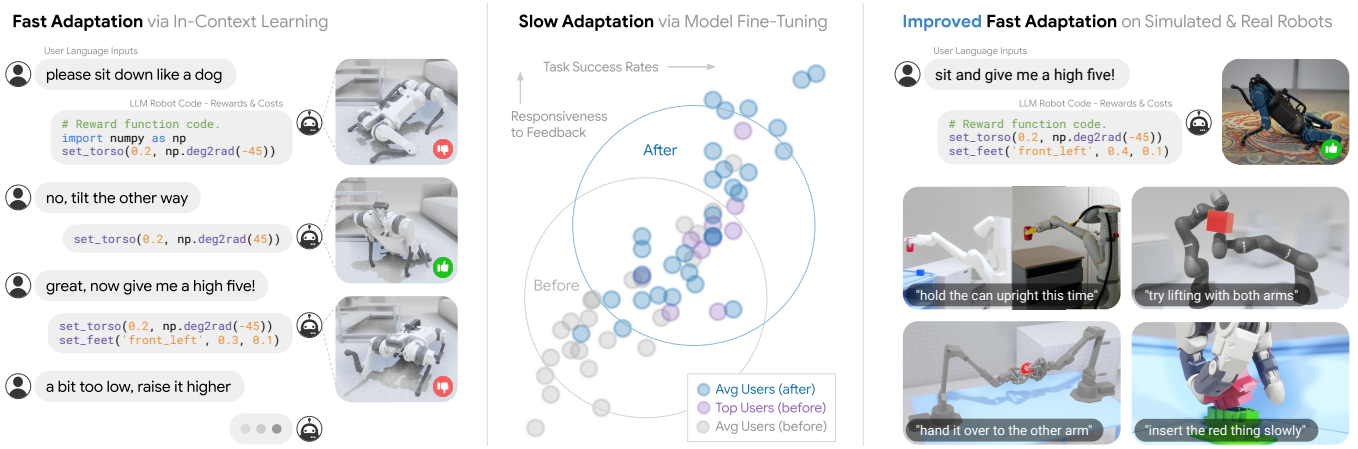

Fig. 1: Code-writing large language models (LLMs) present opportunities for non-experts to teach robots new tasks with language – enabled by fast adaptation via in-context learning (left). In this work, we fine-tune the underlying LLMs to further accelerate fast adaptation and improve their teachability (right). Results with human-robot interactions from non-experts teaching 5 robot embodiments on 78 tasks (gray) show that our framework (middle*) can identify top performing users (purple), and leverage their interactions (only 14% of task coverage) to drive LLM performance improvements for all users (blue) – measured in terms teaching success rates on unseen tasks, responsiveness to user feedback, and number of user corrections. Experiments show that these improvements generalize to new robot embodiments and APIs.

*visualizations from real data: before and after circles are centered on mean good rating rates (i.e., responsiveness to feedback metric, described in Sec. IV-D) vs task success rates over all users.

***Abstract*—Large language models (LLMs) have been shown to exhibit a wide range of capabilities, such as writing robot code from language commands – enabling non-experts to direct robot behaviors, modify them based on feedback, or compose them to perform new tasks. However, these capabilities (driven by in-context learning) are limited to short-term interactions, where users' feedback remains relevant for only as long as it fits within the context size of the LLM, and can be forgotten over longer interactions. In this work, we investigate fine-tuning the robot code-writing LLMs, to remember their in-context interactions and improve their *teachability* i.e., how efficiently they adapt to human inputs (measured by average number of corrections before the user considers the task successful). Our key observation is that when human-robot interactions are formulated as a partially observable Markov decision process (in which human language inputs are observations, and robot code outputs are actions), then training an LLM to complete previous interactions can be viewed as training a transition dynamics model – that can be combined with classic robotics techniques such as model predictive control (MPC) to discover shorter paths to success. This gives rise to Language Model Predictive Control (LMPC), a framework that fine-tunes PaLM 2 to improve its teachability on 78 tasks across 5 robot embodiments – improving non-expert teaching success rates of unseen tasks by 26.9% while reducing the average number of human corrections from 2.4 to 1.9. Experiments show that LMPC also produces strong meta-learners, improving the success rate of in-context learning new tasks on unseen robot embodiments and APIs by 31.5%. See videos, code, and demos at: https://robot-teaching.github.io/.**

I. Introduction

Natural language provides a rich and accessible interface for teaching robots – with the potential to enable anyone with minimal training to direct behaviors, express preferences, and provide feedback. Recent works show that large language models (LLMs), pretrained on Internet-scale data, exhibit out-of-the-box capabilities that can be applied to robotics – from planning a sequence of steps given language commands [1, 30], to writing robot code [41, 59, 71, 45]. Language inputs can also be sequenced in a multi-turn setting for example, to generate and modify reward function code from human feedback to compose new quadruped behaviors via real-time motion control [71] (example in Fig. 1).

LLM-based robot teaching (as shown in Fig. 1) can be driven by in-context learning [9] (e.g., on code and dialogue data), where previous interactions are kept as input context for subsequent ones. In-context learning occurs during inference without gradient updates to model weights, enabling fast adaptation to language instructions (via exemplar-based compositional generalization [11, 32]). However, this adaptation is limited to short-term reactive interactions where the users' feedback remains relevant for only as long as it fits within the context size of the LLM. As a result, if human instructions accumulate over longer multi-step interactions that fall outside the receding

context horizon, previous instructions can simply be forgotten.

We are interested in improving LLMs' *teachability* for robot tasks, i.e., how efficiently they adapt to human feedback, by enabling LLMs to remember their in-context interactions. Teachability in multi-turn language-based human-robot interaction (HRI) can be measured as the average number of human inputs (e.g., corrections) n before the robot succeeds at the task. For instance, $n=1$ refers to the standard zero-shot instruction following setting [33, 44]. Prior works propose to improve teachability by generating linguistic summaries of human feedback [75] or preferences [68] that can be indexed into memory and later retrieved in-context to guide future interactions. However, such methods are often constrained by in-context learning generalization (observed to be more "exemplar-based" i.e., on the basis of similarity to in-context examples [11, 57]), as opposed to generalization from in-weights learning via fine-tuning (which tends to be more "rule-based" i.e., on the basis of minimal features that support category boundaries in the training data [11, 6]). Subsequently, prior methods excel at overfitting to training tasks, but offer limited generalization (e.g., domain-level adaptation) to unseen tasks. Is it possible to leverage both forms of learning to address these shortcomings?

In this work, we investigate improving the teachability of robot code-writing LLMs via *in-context learning* (fast adaptation) by day, and model *fine-tuning* (slow adaptation) by night, to accelerate fast adaptation the next day.[1] Given a setting where non-experts teach robots new tasks with language, our goal is to study which methods of improvement (e.g., via fine-tuning) can best leverage data collected from in-context learning to improve future teachability (as measured on unseen tasks). Our key observation is that when human-robot interactions are formulated as a partially observable Markov decision process (POMDP – in which human language inputs are observations, and robot code outputs are actions), then training an LLM to autoregressively complete previous interactions can be viewed as training a transition dynamics model – that can be combined with classic robotics techniques such as model predictive control (MPC) to discover shorter paths to success. This gives rise to Language Model Predictive Control (LMPC), where we train the LLM to predict imagined future rollouts of human-robot interactions – and at inference time, sample multiple futures (with non-zero decoding temperature) to search for the best one and take the next action (i.e., receding horizon control as a decoding strategy). Classically challenging HRI problems (such as modeling individual user preferences) become more straightforward e.g., by simply conditioning LMPC rollouts on usernames ("user __ might say..."), with the intuition that different users cover different areas of the POMDP.

Extensive experiments (via blind A/B evaluations) show that fine-tuning with LMPC improves the teachability of PaLM 2 [3] on 78 tasks across 5 robot embodiments (on simulated and real platforms) – enabling non-experts to teach robots to achieve higher success rates on unseen tasks by 26.9%, and reduces average number of human corrections from 2.4 to 1.9. In particular, LMPC produces strong meta-learners – teachability improvements generalize to unseen embodiments, improving the success rate of in-context learning new tasks with new robot APIs by 31.5%. Interestingly, we observe substantial gains from top-user conditioned LMPC, which (i) autonomously identifies top users (by performance on training tasks), (ii) groups their data together with a special username "top-user," then (iii) conditions inference-time LMPC rollouts on this special username (i.e., assume everyone is a top-user). Despite top users having seen only 14% of tasks, experiments show this conditioning mechanism drives performance improvements for all users on all tasks, including unseen ones by 10.5%. LMPC also outperforms retrieval baselines [75], and user studies affirm that performance improvements are likely the result of changes in model capability, rather than user teaching proficiency. Our approach is not without limitations – we discuss these and areas for future work in Section V. Videos, code, and datasets will be released.

[1] In-context learning by day & fine-tuning by night analogy is loosely inspired by the role of circadian rhythms in the learning and memory of biological systems.

II. Related Work

Language and Robotics. A large body of work integrates language and robotics, including mapping language to planning primitives [63, 37, 46, 5, 35], imitation learning from demonstrations along with language instructions [33, 44, 58, 60, 47], learning language-conditioned reward functions [48, 34, 21, 14, 50, 2], and using language as corrective feedback to adapt or define new behaviors [75, 16, 15]. We refer the reader to comprehensive surveys for a more complete review of prior work in this area [64, 43].

Recently, LLMs trained on Internet-scale data have been shown to exhibit profound capabilities ranging from step-by-step planning [1, 30, 18, 69, 17, 42, 68, 54], writing robot code [41, 59, 73, 70, 4, 49], commonsense reasoning [62, 39], and acting as a proxy reward function capturing human preferences [38, 29, 71]. In this work, we are also interested in leveraging the power of LLMs for adapting and teaching new behaviors via language feedback [56, 74, 54, 75, 31] – but in contrast to prior work, we focus on not only evaluating online adaptation via in-context learning (e.g., prompting LLMs), but also on how we can improve that adaptation via offline model fine-tuning.

In-Context Learning for Robot Adaptation. In-context learning is a form of supervised meta-training [9], where multiple examples and instructions [51] from the same dataset are packed sequentially into a context buffer that is fed as input to an LLM with an unsupervised autoregressive completion objective [66]. The instructions and examples specify tasks (extending the concept of "task prefixes", i.e., predefined token sequences [53, 52]), where the model is expected to complete further instances of the task by predicting what comes next.

In robotics, in-context learning (via prompting) has been used to elicit a wide-range of capabilities – responding to feedback [74, 31], modifying low-level behaviors [71, 55, 49, 4, 38], remembering and applying user preferences [68], and asking for help [54]. Most related to our work is Zha et al. [75], which investigates robot teaching by summarizing human feedback, and indexing it in memory to be used again as in-context examples for similar future interactions via retrieval (e.g., retrieval augmented generation (RAG) [40]). In contrast, we focus on directly fine-tuning the underlying LLM to improve in-context learning from human language inputs (which can be multi-round contextual). We find finetuning exceeds the performance of retrieval-based methods for teaching unseen tasks – without additional external modules.

Improving LLM Alignment to User Feedback. Our work builds on an active area of research to *align* LLMs with user intent [51]. A common approach is to supervised fine-tune (SFT) the model on expert human inputs and outputs, then use non-expert labeled rankings (preferences) from model outputs to train reward models for reinforcement learning from human feedback (RLHF) [13, 61, 7]. However, these works often focus on mapping from single user inputs to preferred outputs (e.g., single-turn dialogue). In this work, we also investigate learning from human feedback, but we focus on improving the teachability of LLMs that write and improve robot code based on multi-turn, interactive human feedback. Our LMPC approach uses SFT to model human-robot interaction dynamics, and uses inference-time search and receding horizon control to discover shorter paths (with fewer rounds of corrections) to task success.

III. Language Model Predictive Control

We investigate teachability in the context of language-based human-robot interactions, where users communicate with robots via text messages through a chat-like interface next to a simulated visualization of the robot and its surroundings using the MuJoCo simulation engine [65] (see Fig. 3, more details in Appendix VI-L). User messages are free-form and up to users' discretion; they may include instructions, preferences, feedback, etc. In response to each message, the system outputs robot code, which is directly sent to a real-time motion controller on a simulated or real robot (Section III-B). Users then provide subsequent feedback based on the observed robot behavior.

Each human-robot conversation (i.e., chat session) is goal-driven: users are asked to teach one task per session and at the end of each session label "success" or "failure" conditioned on whether they believe the robot to have completed the task. Chat sessions can consist of multiple chat turns (i.e., human-robot input-output pairs) before success. On average, successful sessions run for 2-3 chat turns, while failure sessions run for 5-6 chat turns (see Fig. 3; bar plot shown in bottom left). User messages can be corrections or broken-up step-by-step sub-tasks to piece together more complex ones, and they are usually multi-round contextual. During data collection, users rate individual robot responses as 'good' or 'bad' — good if the robot responded correctly to the most recent human feedback (although it may not be successful at completing the entire task yet), and bad otherwise. We find that the ratio of good chat turn ratings correlates with task success (Fig. 3, bottom right).

A. Problem Statement

Our goal is to improve the teachability of LLMs that follow human instructions and feedback to write robot code. Teachability is defined as the average number of human inputs (chat turns) n before the robot succeeds at the task. This metric measures how efficiently the robot adapts to human inputs, and $n = 1$ is equivalent to a standard zero-shot instruction following setting [33, 44]. To *improve* teachability is to reduce the number of chat turns n before a desired success rate, and can be viewed as a meta-learning objective – i.e., learning to learn faster from human feedback [26]. Intuitively, improving teachability of a model should encourage its responsiveness to feedback, as a means to maximize the likelihood of generating the correct behavior (according to the user). Teachability can also reflect

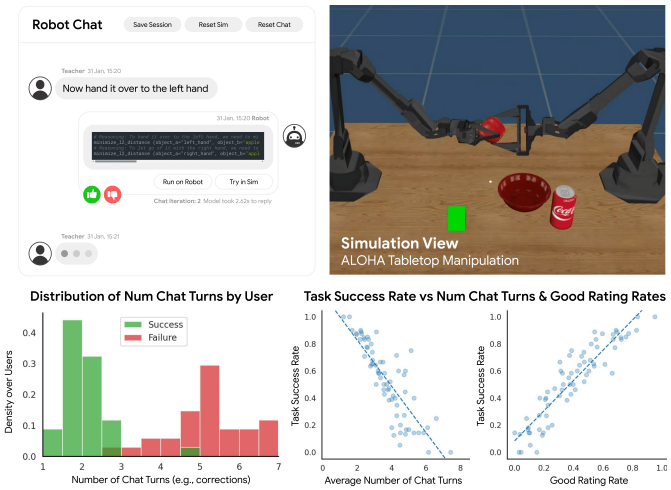

Fig. 3: Our chat interface (top left) allows non-experts to use language to teach robots new behaviors (visualized in simulation, top right). Our LLM responds with reward code, to drive real-time motion control of a simulated or real robot. Statistics show that base model data meets expectations: successful teaching sessions take fewer chat turns than failures (bottom left), and task success rates correlate with fewer chat turns ($r = -0.85$, bottom middle) and higher good rating rates (i.e., responsiveness to feedback, $r = 0.92$, bottom right).

how well a model adapts to preferences. For instance, user input "move a bit to the left" might yield different robot behavior modifications depending on the user – a strong meta-learner (with respect to teachability) is one that can learn this difference to minimize the number of interactions n, conditioned on with whom it interacts.

We formulate language-based human-robot interaction as a partially observable Markov decision process, in which the policy interacts with the human teacher (part of the environment) through code that is executed via motion control on the robot (the action), and the human gives natural language feedback (the observation) and indicates the success of the teaching session (the reward). The policy's goal is to produce code that leads the robot to behave as intended by the human, however, this target behavior has to be inferred from the human feedback. Mathematically, $s_t \in \mathcal{S}$ is the unobserved human (user) state at time t, human text inputs are observations $o_t \in \mathcal{O}$ of the state, and the agent (LLM) generates code as actions $a_t \in \mathcal{A}$ according to a policy $\pi(o_t|o_0,\ldots,o_{t-1})$ that is then executed on the robot. $\mathcal{P}(o_{t+1},r_t|o_{\leq t},a_{\leq t},r_{<t})$ is the transition probability of human interactions, and if at any time the user determines the task is a "success", the episode terminates with a sparse reward $r = 1$, and the chat session terminates. If the robot struggles to improve for more than 7 timesteps (chat turns, or (o_t,a_t) tuples), then the episode terminates as a "failure" ($r = 0$). Improving teachability here can be defined as improving the ability of π to infer the target behavior intended by the human, i.e. to discover shorter paths in the POMDP to task success (i.e., human-labeled success $r = 1$).

While the true transition dynamics are unknown (as they depend on the human teacher), we are interested in modeling the POMDP with language models, with which the human text inputs o_t and robot code as actions a_t can both be represented as a sequence of symbols $(x_0,\ldots,x_n)$, modeled autoregressively with a decoder-only Transformer architecture [66]. Thus we learn $\hat{\mathcal{P}}(o_t,a_t,r_0\ldots,r_7)$ by factorizing the probability of the full interaction sequence into the product of conditional probabilities $\hat{\mathcal{P}}(o_{t+1},r_t,a_t|o_{\leq t},a_{<t},r_{<t}) =$

$\hat{\mathcal{P}}(o_{t+1}|o_{\leq t},a_{\leq t},r_{\leq t})\hat{\mathcal{P}}(r_t,|o_{\leq t},a_{\leq t},r_{<t})\hat{\mathcal{P}}(a_t|o_{\leq t},a_{<t},r_{<t})$ [2]

Our approach is driven by two complementary forms of LLM improvement: (i) in-context learning (fast adaptation) for users to teach the model new tasks *online* (Section III-B), and (ii) Language Model Predictive Control (LMPC) fine-tuning (slow adaptation) to update the model weights *offline* (Section III-C). Our primary objective is to evaluate which of the methods of LLM improvement can most accelerate fast adaptation (measured via teachability of unseen tasks).

B. Fast Adaptation with In-Context Learning

Fast adaptation involves users providing multi-turn language inputs to guide LLM outputs towards generating reward code to elicit desired robot behavior(s). This is an interactive process – users provide feedback based on observing robot behaviors online, rather than labeling offline LLM data. In this work, fast adaptation is driven by in-context learning, where the language model is conditioned on a prompt that provides the initial tokens in the sequence $x_{1:k} = (x_1, ..., x_k)$ and uses the model to complete $x_{k+1:n}$. Our in-context prompt uses PromptBook formatting [4], which contains a description of the embodiment, the available robot APIs, as well as 1-2 example episodes (chat sessions) between the user and LLM, followed by the current chat session (full prompts in Appendix VI-N):

```
# You are a stationary robot arm with a 3-fingered hand.
class Robot:
   def reach(self, obj):
   def min_L2_dist(self, obj1, obj2):
# Example Session.
# Chat Turn #1: move the red and green things together.
reach(obj='red', weight=1.0)
min_L2_dist(obj1='red', obj2='green', weight=1.0)
...
```

We use an existing pre-trained LLM PaLM 2 [3], with which using the above prompts yields non-zero initial task success rates given feedback from the user. The code generated within each turn can either be a single reward function, or a sequence of multiple reward functions. Upon terminating a chat session, the interaction data is saved into a cached dataset to be used for slow adaptation. Note that even with human inputs, the model may struggle to perform certain tasks – experiments in Section IV-D show that slow adaptation is needed to unlock fast adaptation on these tasks.

Fast adaptation requires fast LLM inference runtime speeds, so that latencies do not negatively influence the human-robot interactions. Our model inference runs at 100 tokens per second, and returns robot reward code expressed with 200 - 300 tokens on average (which amounts to roughly 10-15 lines of code). The median duration for each chat turn is 56s, and the majority of user time is spent observing the robot performing the task in simulation.

Real-Time Motion Control with MJPC. We use robot reward code as an interface between LLM and robot actions. This leverages the effective high-level reasoning capabilities of LLMs to translate user intent into semantically meaningful reward functions, which are then used to drive low-level motion control for the robot in real-time, providing immediate visual feedback to the user. Specifically, our approach builds on Yu et al. [71] where given a reward function generated by the LLM, we use MuJoCo Model Predictive Control (MJPC)[3][27] to synthesize robot motion. MJPC runs a receding horizon trajectory optimization algorithm to find an action sequence that maximizes the reward in real-time. This enables near real-time interactions between users and robots.

Our code format extends Yu et al. [71] with 2 notable changes to expand the expressiveness of behaviors across embodiments: (i) Yu et al. [71] relied on two prompts to respond to task commands - one to generate high-level motion descriptions in natural language, and another to convert those into reward code. In our approach, we only use one prompt that embeds motion descriptions as comments interspersed between the lines of the reward code. This Chain-of-Thought style prompting simplifies reward code writing and enables more flexible code generation. (ii) Yu et al. [71] can only specify one reward function (robot behavior) at a time. In our approach, the LLM can sequence multiple reward functions together by writing condition functions that signify when the robot should transition from one reward function to the next. Here is an example of an LLM response to a task that involves transferring an object from one arm to another:

```
# To pick up the apple, bring it close to the left gripper.
min_L2_dist(obj1='left_hand', obj2='apple', weight=5.0)
# To lift up the apple, get its position and increment along z.
pos = get_obj_pos(obj='apple')
set_target_pos(obj='apple', (pos[0], pos[1], pos[2] + 0.25))
# Wait until the apple is in the air.
def condition_fn():
   return get_obj_pos(obj='apple')[2] >= 0.25
wait_until_condition(condition_fn)
# To hand over the apple, bring it close to the right gripper.
min_L2_dist(obj1='apple', obj2='right_hand', weight=5.0)
# Now let go of the apple with the left gripper.
min_L2_dist(obj1='left_hand', obj2='apple', weight=0.0)
```

Functions such as `min_L2_dist` and `set_target_pos` directly set reward terms for real-time MJPC, which returns high-rate low-level action trajectories that maximize rewards.

Sim2Real Transfer. Despite MJPC working well in simulation, it requires precise state estimation, which is computationally expensive, and poses notable challenges when deploying it directly in the real-world. To apply MJPC to real robotic control, prior work used MJPC as a motion planner (MJPC-as-planner) and deployed it to a real mobile manipulator robot [71]. However, the same approach does not work for robots that require high-frequency feedback such as quadruped robots. To enable real-world teaching on quadruped robots, we develop a policy distillation pipeline to train low-level end-to-end policies that are conditioned on cost terms generated by reward code. This can be thought of as training multi-task policies conditioned on latent task descriptors, which serve as an interface between high and low level control. In our experiments, we use policy distillation for real results on the quadruped robot, and the MJPC-as-planner approach [71] for the mobile manipulator robot. We defer additional details in policy distillation and MJPC-as-planner deployment to the Appendix VI-G.

C. Slow Adaptation with Model Fine-Tuning

Gathering interaction data from in-context learning (fast adaptation) allows us to fine-tune the underlying LLM (slow adaptation) to improve its ability to both write useful robot reward code and respond to human feedback, and subsequently improve teachability.

[2]Note o_t, a_t are themselves sequences of symbols also modeled autoregressively.

[3]https://github.com/google-deepmind/mujoco_mpc

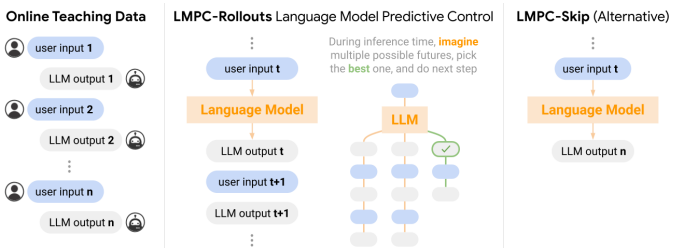

Fig. 4: Given a dataset of users teaching robots new tasks with language (represented as text inputs and code outputs from online in-context learning – left), LMPC-Rollouts is trained to predict subsequent inputs and outputs conditioned on the current chat history (middle), and uses MPC (receding horizon control) for inference-time search to return the next best action (with fewest expected corrections before success). LMPC-Skip is an alternate variant that is trained to directly predict the last action (right). Both LMPC variants accelerate fast robot adaptation via in-context learning.

In this work, we apply supervised fine-tuning (SFT) on our LLM, using Language Model Predictive Control (LMPC). LMPC formulates the human-robot exchange as a POMDP (see Section III-A), with human text inputs as observations o_t, and robot code outputs as actions a_t. Each chat session terminates with a success or failure reward r. We implement two variants of LMPC (depicted in Fig. 4): **(i) LMPC-Rollouts.** Given the current chat session (system prompt and current chat history), the LLM is trained to autoregressively predict the rest of the chat session (sequence of o_t and a_t's, until receiving a reward r at the end of the episode). For training, the input to the LLM is the system prompt with the initial user instruction; both are included because different robot embodiments have different system prompts (robot APIs), and this allows the LLM generation to support different robot APIs at inference time. The target is for the LLM to predict any remaining portions of a chat session, conditioned on the current portion. We only train LMPC-Rollouts on successful trajectories (training on both successes and failures yielded much worse performance. See Appendix VI-B). Training Transformers with causal attention on entire chat sessions (time-ordered sequences of observations and actions, one session per context) directly amounts to training a sequence-conditioned transition dynamics model of the POMDP. This dynamics model is used for search during inference.

A key aspect of LMPC is that at inference time, the fine-tuned LLM is used as a transition model together with model predictive control (MPC) to discover optimal paths to success. MPC can be thought of as a sequence-level decoding strategy [19], but differs from standard ones used in modern language models as it generates multiple episodic rollouts to search for the next best action, and repeats the process at every decision-making step. To do so, we sample from the LLM 8 rollouts with non-zero temperature sampling (next-token decoding) for a max token length of 4096. If a sampled trajectory reaches termination within the max token length, we treat it as successful, since LMPC-Rollouts is only trained on successful data. From these terminated samples, we choose the trajectory with the fewest predicted timesteps (i.e., chat turns) and return its first action, as shown in Fig 4 (center). If no sample terminates, then we randomly pick a trajectory and return its a_{t+1}. This process is then repeated given new human input for every chat turn. Intuitively, LMPC-Rollouts can be thought of as training the LLM via human-robot interaction as a form of chain-of-thought [67] during both training and inference — rather than cloning successful code, LMPC learns the process of getting to the correct code, and accelerating it via search at inference time. **(ii) LMPC-Skip** is a variant of LMPC-Rollouts with the same training inputs, but different targets. LMPC-Skip is trained to predict only the last action, skipping predictions of the interim trajectory (see Fig 4, right). This encourages the finetuned model to predict the final correct code as soon as possible e.g., optimizing for 1-turn success. However, because LMPC-Skip is not trained on nor does it model intermediate interactions with the user, it may be less responsive to corrective feedback. During inference, we condition LMPC-Skip's generation on the system prompt with the chat session so far, and only query the model once to generate a response. Like LMPC-Rollouts, LMPC-Skip is only trained on successful chat sessions.

Top-User Conditioning. To further improve LLM teachability with fine-tuning, we propose conditioning LLM generations, during both training and inference, on the user. For training, we modify the input prompt to include which user generated the following chat session using a unique ID label. Top-users are autonomously identified from the training dataset and are given a special ID "top-user." During inference, we always condition LLM generations on the "top-user" label. We identify top-users as the top 25% of users by their user performance score. This score is the average of a user's task success rate weighted by task difficulty, which is the task's failure rate across all users. See Appendix VI-C for more details.

Top-user conditioning, in the context of LMPC, can be interpreted as conditioning the LLM to generate the distribution of observations o_t (expected human inputs) and actions a_t (expected code outputs) closest to the top 75th percentile of users. Intuitively, if observations are viewed as a partial noisy representation of the true (user) state (or intent, during teaching), then different user proficiency levels can correspond to a varying amount of noise (i.e., higher proficiency is less noise), to which conditioning on top-users prompts the LLM to generate rollouts with less noise. Top-user conditioning draws similarity to performance conditioning with Decision Transformers [12], albeit (in the absence of dense rewards) using inference-time search via MPC. Note that top-user conditioning can broadly index distributions that represent a wide range of user-related attributes (e.g., preferences, user-specific styles, etc.), expanding beyond the scope of what performance conditioning on rewards alone can provide.

IV. Experiments

Our experiments evaluate how much the various proposed finetuning strategies (slow adaptation) improve online in-context learning (fast adaptation) for humans interactively teaching robots via natural language feedback. Evaluations are performed on 78 robot tasks, across 5 robot embodiments in simulation and 2 on real hardware. We specifically explore the following questions:

- How much does fine-tuning improve teachability, especially on test tasks?
- How do LMPC-Rollouts and LMPC-Skip compare?
- What are the benefits of Top-User Conditioning?
- Does finetuning enable cross-embodiment generalization?
- Can iterative finetuning further improve teachability?

All data collection and most evaluations were performed in simulations. All models were trained on data obtained with simulation. We separately evaluate finetuned models on real robots, but we have not experimented with training on data from teaching real robots.

A. Data Collection and Evaluation

To collect human teaching data and evaluate teaching performance, we worked with 35 non-expert users, who were able to collect 350 chat sessions per day. These users are non-experts: they are not researchers or engineers, and they are not familiar with the underlying LLMs or robot code. We instruct users to give natural language feedback on the behavior of the robot for each chat turn, instead of giving technical feedback or giving feedback on the code written by the LLM. Data collection protocol details are in Appendix VI-A. When a user interacts with a new chat session, a random robot embodiment and task is sampled, and the user is asked to teach the robot that task. Data collection is separated into two phases: 1) initial data collection with the base model and 2) subsequent data collection (evaluations) with finetuned models. In phase 2, we randomly sample which model the user interacts with, and the user does not know which model they are currently engaging with. This allows for blind A/B evaluations to minimize biases. For a given model, the data collected can be used for both downstream finetuning and evaluating the model.

Out of the 78 tasks, 51 are train tasks (65%), while 27 are test (35%). While separating tasks into train and test splits allows us to measure model generalization performance, it also means there are less data available for training. To address this and also to make the data distribution robust to user teaching noise, data collection and evaluation of models are typically aggregated across 2 days. Additional data filtering were performed to remove invalid and incorrect data ($< 4\%$). In total, 299 successful chat sessions from the initial data collection were made available for fine-tuning. Across chat sessions, the max total token length is 3900, with 1800 as the median. Given the limited amount of data, and to make LLM responses more robust to small differences in user feedback, we perform data augmentation on the collected data. This is done by generating 5 variations of user instructions (as well as intermediate feedbacks for training LMPC-Rollouts) using PaLM 2-L. We do not generate variations of the robot code. Combining the augmented and the original data, the training set contains about 3M tokens.

For evaluations, we collect approximately 350 chat sessions for all model variants we evaluate, split across all platforms and tasks. We observe minimal user performance drift over time (see Appendix VI-I), so differences in model performance are likely due to changes in model capabilities, and not in users' teaching proficiency.

B. Robot Embodiments and Tasks

In this section we give a brief overview of the 5 robot embodiments in our experiments. We chose these embodiments to explore teaching a diverse set of robot capabilities, from tasks that require a single arm, to bi-manual tasks, and to dexterous and locomotion tasks. See Fig. 1 for illustrations. We include the full list of tasks each embodiment in the Appendix VI-M.

1. Robot Dog. The first embodiment is a small custom quadruped robot [10] with comparable dimension and weight to both Unitree A1 and MIT Mini-Cheetah quadrupeds. Robot Dog has a total of 12 actuated degrees-of-freedom (DoF), 3 on each leg. Our tasks for the Robot Dog range from stationary posing tasks, like sitting and high-five, to more dynamic tasks, like trotting and door opening. We perform Robot Dog experiments in both simulation and the real world.

2. Mobile Manipulator In this embodiment, we use a mobile manipulator [24] with a 7 DoF arm and parallel jaw grippers. We explore tabletop manipulation tasks with rigid objects, such as flipping and stacking objects. The Mobile Manipulator is also available both in simulation and the real world.

3. Aloha. This is a bi-manual embodiment with two 6 DoF arms, each attached with a parallel jaw gripper [76]. The two arms sit directly opposite of each other on a table that has a set of rigid household objects. We explore tasks that require coordination with both arms, such as object transfers.

4. Bi-arm Kuka. Bi-arm Kuka consists of two 7 DoF Kuka LBR IIWA14 arms without end-effectors. The omission of end-effectors allows us to explore whole-body manipulation tasks (e.g. manipulating objects with any part of the robot arm) with this embodiment. We populate the workspace with boxes of different sizes and colors, and the robot needs to manipulate individual or sets of objects to desired goal locations (which may be on the workspace surface or in the air) and in a given order.

5. Kuka+Hand. This embodiment comprises a 7 DoF Kuka arm attached with a custom three-fingered hand. The full system is controlled via torque control. Along with the arm, a set of rigid objects is provided in the workspace. With Kuka+Hand, we explore dexterous manipulation tasks that are difficult to perform with the other manipulation embodiments, such as lifting multiple objects in-hand and plug insertion.

C. Compared Methods

We compare performances across the base model (PaLM 2-S), the two finetuned variants LMPC-Rollouts and LMPC-Skip, and a Retrieval-Augmented Generation (RAG) [40] baseline. Comparing LMPC-Rollouts and LMPC-Skip captures the difference between finetuning the LLM to leverage and predict the entire human-robot chat interaction, versus skipping to predicting the final robot-code response. Comparing to RAG captures if the LLM's improvement in our domain is possible if we do not have access to model weights or the resources needed for finetuning. For RAG, we use a pretrained embedding model to retrieve relevant examples from the training data then inserting them into the LLM context, similar to other RAG applications for adapting robot behavior [75]. See implementation details in the Appendix VI-D.

D. Experiment Results

We evaluate the LLM's *teachability* as task-success for $< N$ user interactions or "chat turns". This is visualized in a curve in Fig. 5, where each point indicates the proportion of chat sessions that achieved success (y-axis) with equal to or less than a certain number of chat turns (x-axis). Models that have better teachability would have a curve that is higher and to the left.

Fig. 5 reports the main teachability results aggregated over all embodiments for the base PaLM 2-S model, the finetuned models LMPC-Rollouts and LMPC-Skip, and the base model with RAG. Through finetuning, models are able to exceed teachability performance of the base model. On train tasks, LMPC-Skip performs the best. On test tasks, LMPC-Rollouts perform the best, improving success rate over the base model by 27%. Both models

Tasks	Model	Success Rate	Num Chat Turns	Good Rating Rate	Successful Tasks Rate	1 Turn Success Rate	2+ Turn Success Rate
Train	PaLM 2-S	34.8%	2.3	16.7%	74.0%	13.0%	21.7%
	RAG	46.4%	2.2	21.4%	**83.3%**	25.1%	21.2%
	LMPC-Skip	**56.0%**	**1.7**	**25.6%**	**83.3%**	**34.6%**	21.4%
	LMPC-Rollouts	51.9%	2.2	21.8%	74.0%	23.5%	**28.4%**
Test	PaLM 2-S	39.4%	2.4	18.1%	81.5%	17.5%	21.9%
	RAG	51.9%	2.0	20.9%	75.0%	27.9%	24.0%
	LMPC-Skip	59.4%	**1.6**	24.7%	**88.9%**	**41.7%**	17.8%
	LMPC-Rollouts	**66.3%**	1.9	**26.5%**	**88.9%**	34.8%	**31.5%**

TABLE I: Comparing base and finetuned models across all embodiments. *Success*: overall success rate on all tasks. *Num Chat Turns*: mean number of chat turns for successful chat sessions. *Good Rating*: proportion of positively rated chat turns after the turn. *Successful Tasks*: proportion of tasks with at least one successful chat session. *1 turn Success*: the proportion of chat sessions that were successful with just one chat turn. *2+ turn Success*: the proportion of chat sessions that were successful with two or more chat turns.

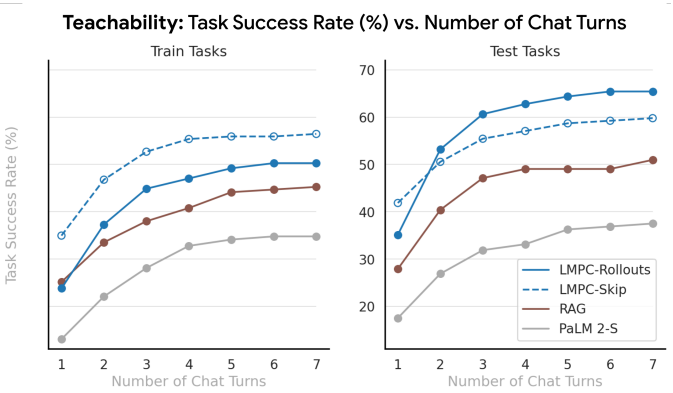

Fig. 5: Our fine-tuned LLMs with LMPC-Rollouts and LMPC-Skip improve the teachability of the base model (PaLM 2-S), and outperforms a RAG [40] baseline across all embodiments. LMPC-Skip overfits to train tasks (left), while LMPC-Rollouts generalizes better (i.e., more teachable and responsive to feedback) on unseen test tasks (right) for multi-turn sessions (with more than one chat turn).

also reach high success rates faster than the base model - matching or exceeding the final success rate of the base model after just one chat turn. While LMPC-Skip achieves the higher 1-turn success rate than LMPC-Rollouts on test tasks, the order flips starting at 2 chat turns. This suggests that LMPC-Rollouts is more amenable to improvements from user feedback. RAG performs competitively over the base model, but it trails behind the finetuned methods in both train and test tasks. See these results separated by embodiment in Table VIII in Appendix VI-B.

Table I provides additional quantitative comparisons across all models evaluated, including:

- *Success Rate*: overall success rate on all tasks and embodiments
- *Num Chat Turns*: mean number of chat turns for successful chat sessions
- *Good Rating Rate*: proportion of positively rated chat turns after the first chat turn (captures responsiveness to corrective feedback)
- *Successful Tasks Rate*: the proportion of tasks with at least one successful chat session
- *1 turn Success Rate*: proportion of chat sessions that were successful with just one chat turn (1st instruction)
- *2+ turn Success Rate*: proportion of chat sessions that were successful with > 1 chat turns. This is the difference between the overall success rate and 1 turn Success Rate

For both train and test tasks, LMPC-Skip achieves the lowest Num Chat Turns for successful chat sessions, as well as the highest 1-turn Success Rate. These reflect how LMPC-Skip is trained to predict the final code as fast as possible. However, LMPC-Rollouts has the highest 2+ turn Success Rate, suggesting it is most amenable to corrective feedback given an incorrect first response. To maximize performance in practice, these results suggest that one should use LMPC-Skip for responding to the initial user instruction, then LMPC-Rollouts for responding to subsequent user feedback. For RAG, while the method improves upon the base model on overall success rate, it achieves lower Successful Task Rate than the base model on test tasks. This suggests that while RAG may be proficient at increasing the success rate of tasks similar to the retrieved examples, it struggles to perform well on novel tasks.

Data	Model	Train Tasks	Test Tasks
All Users	LMPC-Rollouts	-8.4%	-10.5%
	LMPC-Skip	-16.3%	-26.1%
Only Top Users	LMPC-Rollouts	-23.8%	-21.7%
	LMPC-Skip	-9.6%	-13.6%

TABLE II: Changes in success rate without Top-User Conditioning. We evaluate two variants of LMPC-Rollouts and LMPC-Skip that do not apply top-user conditioning: training on data from all users and training on data from only top users. Success rates degrade significantly for both variants, suggesting that 1) focusing LLM generation on the style of top-users is important and 2) top-user data alone is insufficient, and training on the wider data distribution of all users is still important.

Effects of Top-User Conditioning. In Table II, we show the change in task success when training without top-user conditioning on 1) data from all users and 2) data from only top users. These ablations were only performed on the Robot Dog and Mobile Manipulator embodiments due to time constraints. From the initial data collected on the base model, 10 out of 35 users were identified as top-users, and they only covered 11 out of 50 train tasks. However, despite this small coverage, top-user conditioning significantly outperforms both variants of no top-user conditioning, across model types (LMPC-Rollouts and LMPC-Skip) and task types (train and test). This suggests that with top-user conditioning, models can learn to transfer the style of responses induced by top-users teaching to novel tasks. It also highlights the importance of training the LLM to mimic generations from a high-quality data distribution as well as across a diverse data distribution. See Appendix VI-C for analysis on the teaching styles of top-users.

Cross-Embodiment Generalization. Beyond evaluating generalization towards test tasks, we also evaluate whether training on a subset of embodiments would lead to improved performance on new embodiments that the finetuned models were not trained on. To the LLM, the difference in embodiment is captured through the prompt, which contains different robot descriptions and APIs for each embodiment. We performed an experiment where we train the LMPC models on data from Robot Dog, Mobile Manipulator,

Embodiment	Task	PaLM 2-S		LMPC-Rollouts	
		Success	Num Chat Turns	Success	Num Chat Turns
Robot Dog	"downward dog"	100%	1.3	100%	2.8
	"hop"	25%	2.0	100%	2.3
	"high-five with left hand"*	75%	2.3	75%	3.0
	"walk forward in a trotting gait"*	25%	2.0	100%	2.8
	"hop while turning counterclockwise"*	25%	5.0	25%	4.0
Mobile Manipulator	"knock over coke can"	20%	5.0	20%	3.0
	"open top drawer half-way"*	100%	3.4	100%	3.2
	"push coke can from right to left"*	60%	2.0	80%	2.0
	Average	53.8%	**2.9**	**75%**	**2.9**

TABLE III: LMPC-Rollouts has higher success than PaLM 2-S on real robots. Test tasks are starred*. Robot Dog tasks are performed 4 times, Mobile Manipulator tasks 5 times.

Model	Train Embodiments		Test Embodiments
	Train Tasks	Test Tasks	
LMPC-Skip	+28.8%	+19.0%	+18.6%
LMPC-Rollouts	+17.2%	+23.8%	+31.5%

TABLE IV: Finetuned models can generalize to new robot embodiments and APIs not seen during training. Higher improvements in test tasks and embodiments are caused by the train:test split not being explicitly selected for uniform task difficulty and baseline performance; doing so is infeasible as the split needs to be chosen before starting evaluations, when task difficulty and baseline performance were unknown.

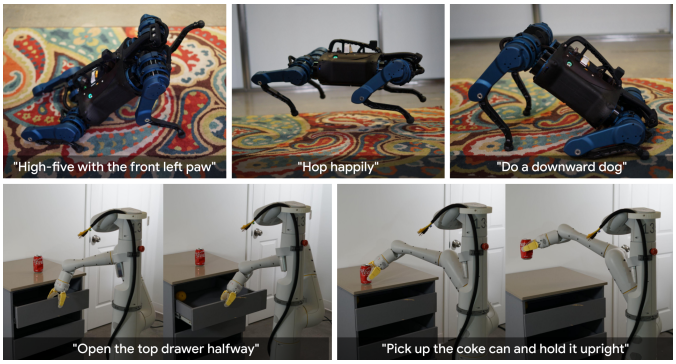

Fig. 6: Tasks evaluated in the real-world Mobile Manipulator and Robot Dog.

and Aloha, omitting Bi-arm Kuka and Kuka+Hand. See results in Table IV, where we report success rate differences between the finetuned models and the base model. We see improvements in test embodiments of 18.6% for LMPC-Skip and 31.5% for LMPC-Rollouts, suggesting that finetuned models generalize not only to test tasks, but test robot embodiments as well. This generalization is non-trivial as the embodiments have very different APIs from each other, and the test embodiments require writing robot reward code that can induce complex dexterous manipulation behaviors.

Real-world Evaluations. We evaluate our approach on a subset of tasks for the Mobile Manipulator and the Robot Dog in the real world (Fig. 6). For each task, we ask users to perform four teaching sessions on the real-robot directly. See results that compare PaLM 2-S and LMPC-Rollouts in Table III. LMPC-Rollouts achieves higher success rate than PaLM 2-S across all tasks. While Num Chat Turns for successful sessions is about the same for PaLM 2-S and LMPC-Rollouts on these tasks, LMPC-Rollouts achieves much higher success rates. See more detailed comparisons between sim and real executions in the Appendix VI-G.

Multiple Fine-tune Iterations. Given that the finetuned models exhibit improved teachability performance over the base model,

Model	Success Rate Diff from Iter 1	
	Train Tasks	Test Tasks
LMPC-Skip Iter 2	+5.1%	-4.7%
LMPC-Rollouts Iter 2	-5.5%	-1.9%

TABLE V: Further finetuning on data generated from both the base model and the first finetuned models models does not yield performance improvements.

additional, iterative training with data collected with the finetuned models could potentially further improve performance. We tested this hypothesis by training Iteration 2 LMPC models with data collected by the Iteration 1 models. Results are shown in Table V. Currently, we do not observe further improvements from the second iteration of finetuning. This implies that the data distribution or data amount used to train the second iteration of models do not differ significantly from that of the first iteration, so the resultant model behaviors remain largely unchanged. While recent works have demonstrated iterative self-improving finetuning for LLMs [72, 25, 22], enabling LLM iterative improvement with human feedback and grounded on robot code executions remain promising but under-explored, and we defer this topic to future research.

V. Discussions

We introduce a method that improves the teachability of LLMs by 1) formulating human-robot interaction as a POMDP and 2) performing Language Model Predictive Control with LLMs finetuned to predict the dynamics of human-robot interactions. LMPC can learn to learn faster from human feedback, and we observe performance gains on test tasks and test robot embodiments. Despite the promising results, there are several limitations to our work that can point to potential future research. Some limitations stem from resource availability. We assume access to sufficient computational resources both for MJPC (e.g. 128 CPU cores) and for LLM finetuning. More efficient MPC and finetuning techniques (e.g. LoRA [28]) would help.

Other limitations relate to the foundation model. We assume that the base LLM can generate some positive chat sessions for bootstrapping the learning process. We also only use language models; future work on expanding feedback modality (e.g. to video/audio inputs) by using multimodal foundation models can expand the richness of feedback as well as improve finetuned models' ability to predict human reactions to robot behavior. Lastly, our approach currently observes no benefit from learning across multiple in-context learning and fine-tuning cycles. Adapting the data distribution, through methods like active task exploration or synthetic data generation, may unlock additional performance gains.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding language to autonomously-acquired skills via goal generation. *arXiv:2006.07185*, 2020.

[3] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[4] Montserrat Gonzalez Arenas, Ted Xiao, Sumeet Singh, Vidhi Jain, Allen Z Ren, Quan Vuong, Jake Varley, Alexander Herzog, Isabel Leal, Sean Kirmani, et al. How to prompt your robot: A promptbook for manipulation skills with code as policies. In *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*, 2023.

[5] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)*, 1:49–62, 2013.

[6] F Gregory Ashby and James T Townsend. Varieties of perceptual independence. *Psychological review*, 93(2):154, 1986.

[7] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[8] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[10] Ken Caluwaerts, Atil Iscen, J Chase Kew, Wenhao Yu, Tingnan Zhang, Daniel Freeman, Kuang-Huei Lee, Lisa Lee, Stefano Saliceti, Vincent Zhuang, et al. Barkour: Benchmarking animal-level agility with quadruped robots. *arXiv preprint arXiv:2305.14654*, 2023.

[11] Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

[12] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[13] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[14] Geoffrey Cideron, Mathieu Seurin, Florian Strub, and Olivier Pietquin. Self-educated language agent with hindsight experience replay for instruction following. *DeepMind*, 2019.

[15] John D. Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[16] Yuchen Cui, Siddharth Karamcheti, Raj Palleti, Nidhya Shivakumar, Percy Liang, and Dorsa Sadigh. No, to the right – online language corrections for robotic manipulation via shared autonomy. In *Proceedings of the 2023 ACM/IEEE Conference on Human-Robot Interaction (HRI)*, 2023.

[17] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

[18] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023.

[19] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.

[20] Edith Galy, Julie Paxion, and Catherine Berthelon. Measuring mental workload with the nasa-tlx needs to examine each dimension rather than relying on the global score: an example with driving. *Ergonomics*, 61(4):517–527, 2018.

[21] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. *arXiv:2007.15543*, 2020.

[22] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

[23] S. G. Hart. Nasa-task load index (nasa-tlx); 20 years later. *Proceedings of the 50th HFES Conference*, pages 904–908, 2006.

[24] Alexander Herzog, Kanishka Rao, Karol Hausman, Yao Lu, Paul Wohlhart, Mengyuan Yan, Jessica Lin, Montserrat Gonzalez Arenas, Ted Xiao, Daniel Kappler, Daniel Ho, Jarek Rettinghouse, Yevgen Chebotar, Kuang-Huei Lee, Keerthana Gopalakrishnan, Ryan Julian, Adrian Li, Chuyuan Kelly Fu, Bob Wei, Sangeetha Ramesh, Khem Holden, Kim Kleiven, David Rendleman, Sean Kirmani, Jeff Bingham, Jon Weisz, Ying Xu, Wenlong Lu, Matthew Bennice, Cody Fong, David Do, Jessica Lam, Yunfei Bai, Benjie Holson, Michael Quinlan, Noah Brown, Mrinal Kalakrishnan, Julian Ibarz, Peter Pastor, and Sergey Levine. Deep rl at scale: Sorting waste in office buildings with a fleet of mobile manipulators, 2023.

[25] Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689*, 2022.

[26] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[27] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with mujoco. *arXiv preprint arXiv:2212.00541*, 2022.

[28] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[29] Hengyuan Hu and Dorsa Sadigh. Language instructed reinforcement learning for human-ai coordination. In *40th International Conference on Machine Learning (ICML)*, 2023.

[30] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[31] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[32] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

[33] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning, 2022.

[34] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *NeurIPS*, 2019.

[35] Siddharth Karamcheti, Edward C. Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson L. S. Wong, and Stefanie Tellex. A tale of two draggns: A hybrid approach for interpreting action-oriented and goal-oriented instructions. In *First Workshop on Language Grounding for Robotics @ ACL*, 2017.

[36] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[37] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Human-Robot Interaction*, pages 259–266, 2010.

[38] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[39] Minae Kwon, Hengyuan Hu, Vivek Myers, Siddharth Karamcheti, Anca Dragan, and Dorsa Sadigh. Toward grounded commonsense reasoning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024. URL arXivpreprintarXiv:2306.08651.

[40] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir

Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[41] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[42] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[43] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language, 2019.

[44] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data, 2021.

[45] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[46] 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.

[47] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

[48] Suvir Mirchandani, Siddharth Karamcheti, and Dorsa Sadigh. Ella: Exploration through learned language abstraction, October 2021.

[49] Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large language models as general pattern machines. *arXiv preprint arXiv:2307.04721*, 2023.

[50] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv:1704.08795*, 2017.

[51] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[52] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018. URL https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[53] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[54] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners, 2023.

[55] Hao Sha, Yao Mu, Yuxuan Jiang, Li Chen, Chenfeng Xu, Ping Luo, Shengbo Eben Li, Masayoshi Tomizuka, Wei Zhan, and Mingyu Ding. Languagempc: Large language models as decision makers for autonomous driving, 2023.

[56] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. *arXiv preprint arXiv:2204.05186*, 2022.

[57] Roger N Shepard and Jih-Jie Chang. Stimulus generalization in the learning of classifications. *Journal of Experimental Psychology*, 65(1):94, 1963.

[58] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *CoRL*, 2021.

[59] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[60] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *NeurIPS*, 2020.

[61] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback, 2022.

[62] Alon Talmor, Ori Yoran, Ronan Le Bras, Chandra Bhagavatula, Yoav Goldberg, Yejin Choi, and Jonathan Berant. CommonsenseQA 2.0: Exposing the limits of AI through gamification. *arXiv preprint arXiv:2201.05320*, 2022.

[63] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[64] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):25–55, 2020. doi: 10.1146/annurev-control-101119-071628. URL https://doi.org/10.1146/annurev-control-101119-071628.

[65] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[67] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[68] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *arXiv preprint arXiv:2305.05658*, 2023.

[69] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

[70] Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R. Walter. Statler: State-maintaining language models for embodied reasoning, 2023.

[71] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

[72] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

[73] Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D Goodman, and Nick Haber. Parsel: A (de-) compositional framework for algorithmic reasoning with language models. *arXiv preprint arXiv:2212.10561*, 2023.

[74] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[75] Lihan Zha, Yuchen Cui, Li-Heng Lin, Minae Kwon, Montserrat G. Arenas, Andy Zeng, Fei Xia, and Dorsa Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024. URL https://arxiv.org/abs/2311.10678.

[76] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

AUTHORSHIP AND ACKNOWLEDGMENTS

Acknowledgements. We thank John Guilyard for his expert animations, and Giles Ruscoe for beautiful renderings. We thank Steven Bohez, Yuval Tassa, Tom Erez, Murilo Martins, Rugile Pevceviciute, David Rendleman, and Connor Schenck for their dedication to ensuring we had strong simulated environments. We thank Travis Armstrong, Noah Brown, Spencer Goodrich, Craig Hickman, Atil Iscen, Jerad Kirkland, Jason Powell, Stefano Saliceti, Ron Sloat, Sergey Yaroshenko, Eddie Yu, Grace Vesom, and Jake Varley for additional robot platform support and robot lab operations. Special thanks to Michael Ahn, Kendra Byrne, Aleksandra Faust, René Wagner, Yuheng Kuang, Yao Lu, Yansong Pang, and Zhuo Xu for supporting this project.

We thank all the users who volunteered to collect the robot teaching data. We also thank the Google DeepMind Visualization and Human Interaction teams for their help with the development and support of the chat interface. We also want to thank the entire Google DeepMind Robotics team whose tireless efforts can be traced to additional support on this paper. This includes Administrative, Product, Programs, and Strategy teams whose contributions impact all of the team's successes. We also want to thank our friends in Google DeepMind and Google Research for their guidance, inspirational research, and even direct contributions.

Program Leads

This project is part of the Google DeepMind 2023 program "ApprenticeBots," an interactive embodied AI moonshot with the mission statement: "anyone can teach a robot, and a robot that can learn from anyone."

Carolina Parada, *Director*
Nik Stewart, *Technical Program Manager*
Jie Tan, *Team Lead*

Technical Leads

Andy Zeng, *Research Lead*
Wenhao Yu, Fei Xia, *Data Collection & Teaching Leads*
Jacky Liang, *Model Training & Improvement Lead*
Jasmine Hsu, *Data & Logging Lead*
Peng Xu, *Infrastructure Lead*
Ben Jyenis, *Operations Lead*
Erik Frey, *Simulation Lead*

Operations

Ben Jyenis, Travis Armstrong, *Head of operations*
Jasmine Hsu, Jacky Liang, *Data collection monitoring*
Wenhao Yu, *Pilot studies for Robot Dog*
Fei Xia, *Pilot studies for Mobile Manipulator*
Baruch Tabanpour, *Pilot studies for Aloha*
Maria Attarian, Jonathan Tompson, *Pilot studies for Bi-arm Kuka*
Joss Moore, Maria Bauza, *Pilot studies for Kuka+Hand*
Contributors: Maria Attarian, Ken Caluwaerts, Jasmine Hsu, Jacky Liang, Assaf Hurwitz Michaely, Jonathan Tompson, Fei Xia, Wenhao Yu, Andy Zeng, Tingnan Zhang

Data Logging Infrastructure

Jasmine Hsu, Ken Caluwaerts, *Datasets and dashboards*
Peng Xu, Assaf Hurwitz Michaely, Jacky Liang, *Materialization*
Contributors: Adil Dostmohamed, Marissa Giustina, Nikhil Joshi, Jacky Liang, Quan Vuong, Tingnan Zhang

Model Serving Infrastructure

Assaf Hurwitz Michaely, Ying Xu, *Core contributors*
Jasmine Hsu, Ken Caluwaerts, Adil Dostmohamed, *LLM Chat UI*
Contributors: Jacky Liang, Allen Ren, Andy Zeng, Tingnan Zhang

Model Training Infrastructure

Core contributors: Assaf Hurwitz Michaely, Jacky Liang, Peng Xu, Andy Zeng, Jasmine Hsu, Edward Lee
Contributors: Quan Vuong, Tingnan Zhang

Evaluations & Analysis

Jacky Liang, *Technical Lead*
Leila Takayama, *Human-Robot Interaction Lead*
Contributors: Alex Bewley, Keerthana Gopalakrishnan, Jasmine Hsu, Jacky Liang, Assaf Hurwitz Michaely, Dorsa Sadigh, Fei Xia, Ted Xiao, Andy Zeng, Tingnan Zhang

Prompt Engineering

Maria Bauza, Marissa Giustina, Kuang-Huei Lee, Jacky Liang, Joss Moore, Dushyant Rao, Baruch Tabanpour, Fei Xia, Wenhao Yu, Andy Zeng

Simulation & MJPC

Maria Attarian, Ken Caluwaerts, Erik Frey, Chuyuan Kelly Fu, Nimrod Gileadi, Leonard Hasenclever, Jan Humplik, Nikhil Joshi, Ben Jyenis, Joss Moore, Dushyant Rao, Baruch Tabanpour, Fei Xia, Ted Xiao, Wenhao Yu, Tingnan Zhang

Robot-Specific Infrastructure

Robot Dog: Ken Caluwaerts, Marissa Giustina, Chase Kew, Ken Oslund, Wenhao Yu Tingnan Zhang,
Mobile Manipulator: Fei Xia, Chuyuan Kelly Fu
Aloha: Baruch Tabanpour, Jonathan Tompson, Erik Frey
Bi-arm Kuka: Maria Attarian
Kuka+Hand: Maria Bauza, Joss Moore, Dushyant Rao, Nimrod Gileadi

Real Robot Deployment & Policy Distillation

Ken Caluwaerts, Chuyuan Kelly Fu, Leonard Hasenclever, Jan Humplik, Chase Kew, Sean Kirmani, Kuang-Huei Lee, Ken Oslund, Allen Ren, Jonathan Tompson, Quan Vuong, Fei Xia, Ted Xiao, Zhuo Xu, Wenhao Yu, Tingnan Zhang

Advising

Alex Bewley, Erik Frey, Leonard Hasenclever, Jasmine Hsu, Jan Humplik, Brian Ichter, Kuang-Huei Lee, Jacky Liang, Carolina Parada, Dushyant Rao, Dorsa Sadigh, Nik Stewart, Leila Takayama, Jie Tan, Fei Xia, Ted Xiao, Peng Xu, Wenhao Yu, Andy Zeng, Tingnan Zhang

Additional Contributions

Authorship and Acknowledgments: Nik Stewart
Paper Content and Web Posts: Carolina Parada, Andy Zeng, Wenhao Yu, Jacky Liang, Fei Xia, Tingnan Zhang
Steering: Carolina Parada, Nik Stewart, Izhak Shafran, Vincent Vanhoucke, Maja Mataric, Leila Takayama, Jie Tan, Dorsa Sadigh, Andy Zeng, Wenhao Yu, Jacky Liang, Fei Xia, Tingnan Zhang

VI. Appendix

We organize the appendix as follows:

- Details on data collection (e.g., chat UI), and evaluation protocol (e.g., task sampling). Section VI-A
- Additional results and evaluations in Section VI-B.
- Details on top-users conditioning (Section VI-C): how they are autonomously selected, quantitative and qualitative analysis on how top-user teaching data differs from other users.
- Retrieval baseline details (Section VI-D) and data augmentation (Section VI-E).
- Quantitative analysis of chat feedback embeddings (Section VI-F).
- Real robot experiments details (Section VI-G) including model training and deployment (Section VI-H).
- User studies and performance drift Section VI-I.
- Failure mode analysis Section VI-J.
- Model performance on existing code-writing benchmarks Section VI-K.
- Robot-specific embodiment details (Section VI-L), tasks (Section VI-M), and prompts (Section VI-N).

A. Data Collection and Evaluation Details

During data collection, non-expert users interact with the robot using natural language through a browser-based chat UI (shown in Fig. 3). The chat UI displays a user input box, the message history, and a visualization of the simulated robot and its surroundings using MuJoCo [65]. The human provides textual input and the LLM replies to each subsequent user query with executable code. The user can then select a button to either run the code in the simulator to observe the resulting motion, or run it on a real robot. The user can continue to provide feedback (which can be multi-turn contextual) and continue modifying the behavior through text inputs in the chat UI until the desired robot behavior is achieved. Each user is remotely connected (via Remote Desktop) to one machine, drawn from a shared pool of high performance machines (128 cores) in the cloud. Machines with high core counts are necessary for Mujoco's MJPC [27] to synthesize robot motion at an interactive rate – leading to better low-level robot behaviors and subsequently user feedback data.

For each chat session, the user teaches the robot one task specified via language e.g., teach the robot-dog to "sit down and give a high-five." For each chat turn (user-input, LLM-output pair), the user has the option to rate the individual robot response as 'good' or 'bad'. These single turn good rating rates (while not used during training) help us evaluate responsiveness to feedback across individual responses, and we find they are strongly correlated to task success (see Fig. 3). Finally, users can label the entire chat session as "success" by clicking a success button if the robot succeeded at the task during the conversation, or "failure" if the task does not succeed within 7 rounds of human input. Variation in success is expected, and users are encouraged to rate success based on the observed behaviors of the robot (as opposed to the accuracy of the code). After labeling a chat session, the chat history UI refreshes, the robot simulator is reset, and a new sampled task and embodiment is presented to the user. Users are able to flag chat sessions in case of technical difficulties.

Our UI backend uses a Task Sampler, which is configured to (i) randomly sample tasks from the set of 78 tasks across 5 embodiments (platforms illustrated in Fig. 1), and (ii) randomly sample an LLM model to connect to. Users do not know which model they speak to, which allows us to perform fair blind A/B evaluations. All experiment numbers are computed with data collected using this sampler.

From the perspective of users, our data collection protocol is equivalent to our evaluation protocol – we train on data collected from users interacting with the model(s) through the UI, and we measure whether users believe the model(s) to have improved (via statistics on good rating rates and session success labels) through the UI with blind A/B evaluations. This deviates from the standard norm in robot learning pipelines (e.g., 3-stage pipeline of collect data, train, and evaluate), and presents practical infrastructure/operations advantages (predominantly around simplicity).

To operationalize data collection, we started off with running multiple pilot sessions with the users for each embodiment. These pilot sessions were focused on introducing these 35 non-expert users to the the chat UI, the type of tasks they are expected to teach, and the MuJoCo simulation environment. After the pilot sessions conclude, the users were tasked to contribute 10 chat sessions per day on all embodiments through the task sampler, amounting to 350 chat sessions every day. The users were also asked to fill out a brief questionnaire for a feedback after each day about their experience on the data collection and the overall teaching session. To meet the daily target of 350 chat sessions per day, it was important to maintain participation from all of the users equally to obtain the expected level of diversity in the data. We maintained consistent distribution of number of chat sessions across multiple embodiments i.e. Robot Dog, Mobile Manipulator, Aloha, Bi-arm Kuka, Kuka+Hand.

The users who participated in these experiments were 23-43 years of age (M=30.5, SD=5.6), including 11 who identified as cisgender women, 17 who identified as cisgender men, and 1 as non-binary. They had a range of educational degrees (9 Associates degrees or some college, 6 Bachelor of Arts, 11 Bachelor of Sciences, and 3 Masters degrees) – 14 non-technical and 15 technical. When asked about their familiarity with the ML models on a scale of 1 (zero familiarity) to 5 (most familiar), 15 users reported 1 (zero familiarity), 11 users reported 2, and 3 users reported 3; none of the users reported to have more familiarity with ML models (4 or 5).

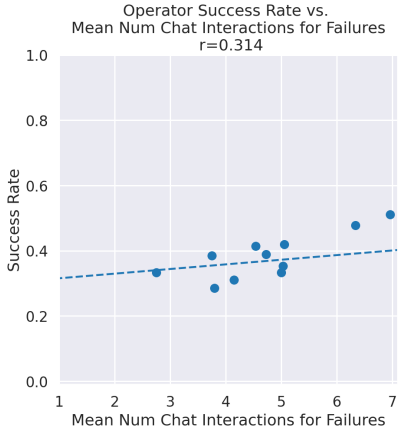

Fig. 7: Correlation of Operator Success Rate and Num Chat Turns until Failures

	Train Tasks	Test Tasks
LMPC-Rollouts-with-Failures	−11.5%	−14.0%

TABLE VIII: Success Rates of Training LMPC-Rollouts on both Success and Failure chat sessions.

User Persistence Analysis. We plot the success rate of each user against the mean number of chat turns in failed sessions across each user in Fig. 7. The higher the mean number of chat turns for failure, the more persistant the user was in teaching the robot (i.e. the user did not give up early). These two quantities exhibit a slight positive correlation, suggesting that on average, users who were more persistant at teaching achieved slightly higher success rates.

B. Additional Results

Per Embodiment Evaluation. Fig. 8 shows our main teachability result (Fig. 5) separated by embodiments. On test tasks, models improved upon the base model the most in Aloha and Bi-arm Kuka, while LMPC-Rollouts improved much higher on Kuka with Hand than the other model.

Embodiment	Chat Session Duration (s)	Chat Turn Duration (s)
Kuka+Hand	429	97
Bi-arm Kuka	406	88
Aloha	200	66
Mobile Manipulator	238	65
Robot Dog	138	41

TABLE VI: Median Chat Session and Chat Turn Durations across Embodiments

Model	Chat Session Duration (s)	Chat Turn Duration (s)
LMPC-Rollouts	187	60
LMPC-Skip	158	49

TABLE VII: Median Chat Session and Chat Turn Durations across Models

Chat Duration Analysis. We measured and analyzed the duration of chat sessions and chat turns and compared them across different embodiments and models. Chat turn duration measures the total time it took for the model to respond, for the user to run the robot code in simulation, for the user to observe the resulting robot behavior, and for the user to input the subsequent language feedback. Fig. 9 shows the distribution of chat turn durations across both models and embodiments. While there are no obvious differences in these distributions, some are more long-tailed than others, and we see this in the median statistics. In Table VI, we show the median durations for chat sessions and chat turns across different embodiments. Kuka+Hand and Bi-arm Kuka have significantly higher durations than other embodiments. This reflects that these embodiments were likely more difficult to teach (it took longer for users to respond) as well as taking longer to simulate (they had tasks that had longer horizons than the other embodiments). In Table VII, we compare the median durations for LMPC-Rollouts and LMPC-Skip. LMPC-Rollouts has slightly higher chat turn and chat session durations, and this difference reflects how inference (decoding the LLM for entire chat sessions) for LMPC-Rollouts takes slightly longer than inference for LMPC-Skip. Lastly, in Fig. 10, we show a small negative correlation between task success rate and chat duration — the longer it takes for users to complete a chat turn, the less likely it is for that task to be successful.

Training LMPC on Both Success and Failures. In principle, LMPC-Rollouts (when viewed as a dynamics model) can be trained on both success and failure data (since all chat turns are valid transitions, regardless of whether the session ended in task success). In this version, LMPC-rollouts also predicts (on trajectory termination) whether the predicted rollout would lead to a success or failure. Inference-time search would then be adjusted accordingly to disregard sampled rollouts that ended in predicted failure. While this remains an interesting aspect of LMPC-Rollouts, our main experiments report results from training LMPC-Rollouts on success data only (as a fair comparison with LMPC-Skip, which can only be trained on success data), with which we do observe performance improvements over mixing failure sessions into the training data (results in Table VIII). We hypothesize that training LMPC-Rollouts only on sessions that ended in task success yields more efficient inference-time search, since the alternative of training on both success and failure sessions leads to more unused predicted rollouts that terminate with failure.

C. Top-Users and Details on Autonomous Top-Users Selection

Model	Top Users	Other Users
LMPC-Skip	+15.1%	+14.2%
LMPC-Rollouts	+26.3%	+18.9%

TABLE IX: Success rate improvements by user group for test tasks.

We identify top-users by evaluating how well they perform on training tasks (Appendix Section VI-M), weighted by task difficulty. Let there be N tasks and K users. Let $s(n,k)$ denote the self-reported success rate of the nth task for the kth user, $c(n,k)$ denote the number of times the kth user taught the nth task, and $\bar{c}(n,k) = \mathbb{1}(c(n,k) \geq 1)$ to indicate whether or not the kth user has taught the nth task. Due to practical constraints, $\bar{c}(n,k) = 0$ for many user-task pairs. We define the task difficulty rating $d(n)$ as the average task failure rate across all users: $d(n) = 1 - \frac{1}{K_n}\sum_{k=1}^{K} s(n,k)\bar{c}(n,k)$, where $K_n = \sum_{k=1}^{K}\bar{c}(n,k)$. Then, we define a user performance score as a user's average success rate weighted by the task difficulty rating: $h(k) = \sum_{n=1}^{N_k} d(n)s(n,k)\bar{c}(n,k)$, where $N_k = \sum_{n=1}^{N}\bar{c}(n,k)$. We define top-users as those who are in the top 75th percentile by this performance score. We refer to the remaining users as "other users".

Table IX shows the average performance improvements of user-conditioned LMPC over the base model split by top users and other users. We observe largest performance improvements when LMPC-Rollouts (conditioned on top-users) is served to top users directly, and this is less evident with LMPC-Skip, suggesting that inference-time search (via MPC) over future interactions performs better at catering to improving the teachability of top users (e.g., satisfying their criterion for success).

Our experiments in the main paper (Table II) demonstrate that conditioning LMPC on top-users can drive performance improvements for all users – but what makes top-user teaching data different from other users? To explore this question, we define 4 axes

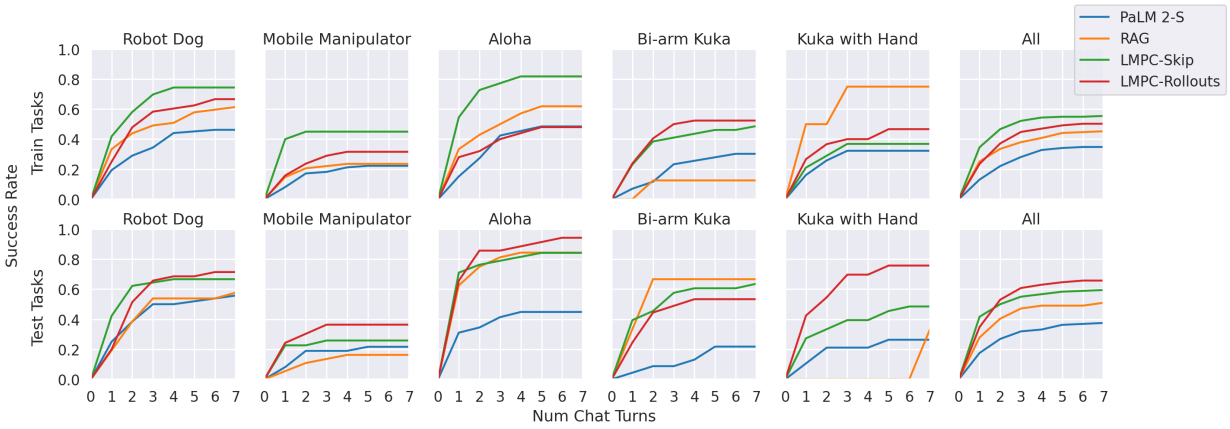

Fig. 8: Task Success vs. Number of Chat Turns. across embodiments

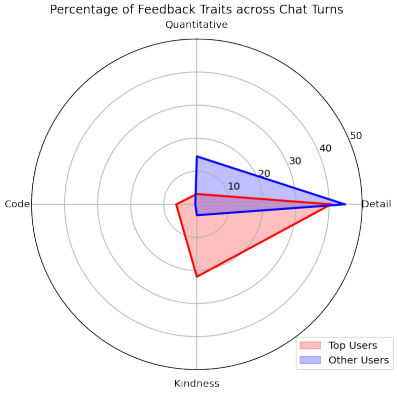

Fig. 11: Analysis of feedback traits across all chat turns for Top Users and Other Users

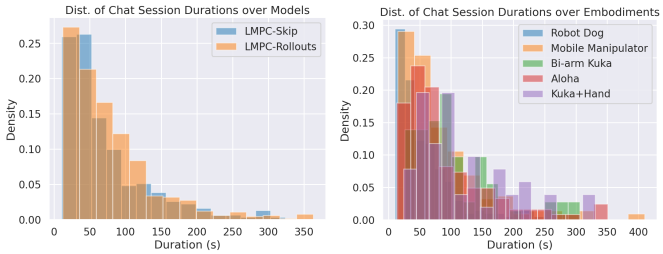

Fig. 9: Distribution of Chat Turn Duration over Models and Embodiments

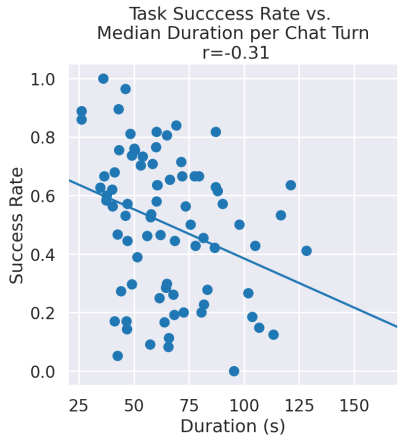

Fig. 10: Correlation of Success Rate vs. Median Chat Turn Durations across Tasks.

to categorize the feedback: (1) Quantitative, (2) Related to Code, (3) Detailed, and (4) Kind. Classification is done via GPT-4 with a few-shot prompt. See Section VI-C1 for the prompts used to do the trait classification. Each message of feedback is classified with these traits. If a given chat session has a trait for any message in the session, we say the entire session had that trait. For example, if one question is “detailed", we say the session had “detailed" feedback.

As can be seen in the analysis of feedback for all users in Fig. 11, both the top-users and other users provide quite detailed feedback. One interesting trait that stood out is that top users are more “kind", which might imply that they have more patience for errors in the code generated. This may lead to more thoughtful responses that end up in successful policies. Another surprising finding was that the other-users were more quantitative in their responses than the top users, indicating that it’s preferred to give softer feedback signals than precise numbers. These insights are preliminary and further investigation is needed to understand what makes top users successful.

To better understand how top-users teach, we asked them about their teaching strategies and what advice they would give to others. Many top-users started out with simple, natural language instructions. Then they would review the robot’s performance. If that performance was not satisfactory, then they would provide more detailed feedback. One top-user summarized this approach quite well: “Think of talking to a toddler, sentences are a couple words long and are easy to understand; however, this toddler knows words or terms from a university physics textbook (e.g. rotational velocity, perpendicular, yaw, pitch, roll).”

1) Trait Classification Prompts: **Code Feedback Prompt**

```
# INSTRUCTIONS: Given an instruction teacher gives to a student, rate this instruction based on "Code Feedback". After considering the instruction carefully, output one of three following ratings along with a justification:
* NEGATIVE: There is *no* feedback related to the code in the instruction.
* NEUTRAL: There is a *fuzzy* feedback related to the code in the instruction.
* POSITIVE: There is *clear* feedback related to the code in the instruction.
# EXAMPLE 1:
QUERY: grasp apple and place it on top of the cube
JUSTIFICATION: There is *no* feedback related to the code in the instruction.
RATING: NEGATIVE
# EXAMPLE 2:
QUERY: set your turning_speed equal 0
JUSTIFICATION: There is *clear* feedback related to the code in the instruction.
RATING: POSITIVE
# IMPORTANT: Always output justification first, then the rating.
# INPUT
QUERY: {query}
JUSTIFICATION:
```

Model	Success Rate Diff w/o Data Augmentation	
	Train Tasks	Test Tasks
LMPC-Skip w/o Aug	-7.1%	+0.6%
LMPC-Rollouts w/o Aug	+2.8%	-7.0%

TABLE X: Success rate differences between models that do not use data augmentation and models that do.

Quantitative Feedback Prompt

```
# INSTRUCTIONS: Given an instruction teacher gives to a student, rate this instruction based on "Quantitative
Feedback". After considering the instruction carefully, output one of three following ratings along with a
justification:
* NEGATIVE: There are *no* numerical and quantitative information in the instruction.
* NEUTRAL: There is a *fuzzy* indication of numerical and quantitative information in the instruction.
* POSITIVE: There is *clear* numerical and quantitative information in the instruction.
# EXAMPLE 1:
QUERY: move the left arm towards green cube and push it to the right 20cm
JUSTIFICATION: There is *clear* numerical and quantitative information in the instruction.
RATING: POSITIVE
# EXAMPLE 2:
QUERY: pick up the connector
JUSTIFICATION: There are *no* numerical and quantitative information in the instruction.
RATING: NEGATIVE
# IMPORTANT: Always output justification first, then the rating.
# INPUT
QUERY: {query}
JUSTIFICATION:
```

Kindness Prompt

```
# INSTRUCTIONS: Given an instruction teacher gives to a student, rate this instruction based on "Kindness".
After considering the instruction carefully, output one of three following ratings along with a justification:
* NEGATIVE: This instruction is *not* kind.
* NEUTRAL: This instruction is neither kind nor unkind.
* POSITIVE: This instruction is kind.
# EXAMPLE 1: QUERY: move the cube a little bit to the left please
JUSTIFICATION: This instruction is kind.
RATING: POSITIVE
# EXAMPLE 2:
QUERY: did you forget how to walk? Please reposition yourself heading south
JUSTIFICATION: This instruction is *not* kind.
RATING: NEGATIVE
# EXAMPLE 3:
QUERY: close the door by pushing it.
JUSTIFICATION: This instruction is neither kind nor unkind.
RATING: NEUTRAL
# IMPORTANT: Always output justification first, then the rating.
# INPUT
QUERY: {query}
JUSTIFICATION:
```

Detail Prompt

```
# INSTRUCTIONS: Given an instruction teacher gives to a student, rate this instruction based on "Detail". After
considering the instruction carefully, output one of three following ratings along with a justification:
* NEGATIVE: This instruction is *not* detailed.
* NEUTRAL: This instruction is neither detailed nor undetailed.
* POSITIVE: This instruction is quite detailed.
# EXAMPLE 1:
QUERY: very good, now extend your front left paw as far forward as possible without losing balance on the rest
of your legs. Change the angle of your torso as needed to maintain balance
JUSTIFICATION: This instruction is quite detailed.
RATING: POSITIVE
# EXAMPLE 2:
QUERY: stand up
JUSTIFICATION: This instruction is *not* detailed.
RATING: NEGATIVE
# IMPORTANT: Always output justification first, then the rating.
# INPUT
QUERY: {query}
JUSTIFICATION:
```

D. RAG Implementation Details

To implement our RAG baseline, we first construct an embedding dataset from the same data used to train LMPC-Skip. This dataset includes each data point's initial user instruction, its Gecko embedding (obtained via an embedding model based on PaLM 2), and the final successful response code. During inference, we use the embedding of the initial user instruction of the current chat session to find the 5 most relevant data points of the same robot embodiment from the dataset. This is done by first selecting the closest 30% of data by cosine similarity (embeddings are normalized), then applying the farthest point sampling algorithm among this set to ensure diversity of the selected data points. Finally, the retrieved data points are re-ordered from lowest to highest relevancy, such that the most relevant example is closest to the current instruction. This ordered list of (instruction, code) pairs is then inserted into the context of the LLM prompt.

E. Data Augmentation Details

We augment user inputs, including 1st input (task request) and subsequent feedback and corrections with PaLM 2-L. The prompt for augmentation asks PaLM 2-L to rewrite original text in K different ways by replacing words with synonyms, rephrasing, changing grammatical structure, sentence lengths, punctuation, etc. We also specifically prompt the model to output the K ways in one batch with variations in the batch and use relatively high generation temperature (0.8) to ensure the output is sufficiently diverse.

For example, a user's request of "pick up the cube" is rewritten into "grab the cube and raise it ", "lift up the cube", "raise the cube", etc; user's correction "wrong direction, keep hopping but turn the opposite direction" for the "hop while turning counterclockwise" task is rewritten into "that is the incorrect direction, maintain hopping but go the opposite way", "you are going the wrong way, keep hopping but turn in the opposite direction", "wrong direction, maintain hopping but turn the opposite way", etc.

The augmented data is used for training both LMPC-Skip and LMPC-Rollout models. We report the success rate differences when training models without data augmentation in Table X. Without data augmentation, LMPC-Skip performs worse on train tasks, but on par on test tasks. By contrast, without data augmentation, LMPC-Rollouts performs on par on train tasks, but much worse on test tasks. This suggests that the generalization capabilities of LMPC-Rollouts benefits more from data augmentation than does LMPC-Skip. We hypothesize this is due to that data augmentation makes LMPC-Rollouts' chat session predictions more robust to compounding errors, leading to better predictions of feedback dialogue.

F. Analysis of Chat Feedback Embeddings

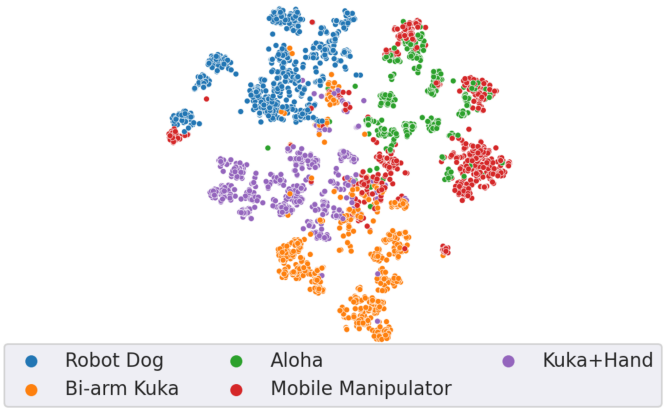

Fig. 12: T-SNE plot of embeddings of human feedback across embodiment.

What kinds of feedback do users provide to steer robot behaviors? To study this, we compute language embeddings on all individual chat turn user queries, using a finetuned T5 XL model [53]. Then,

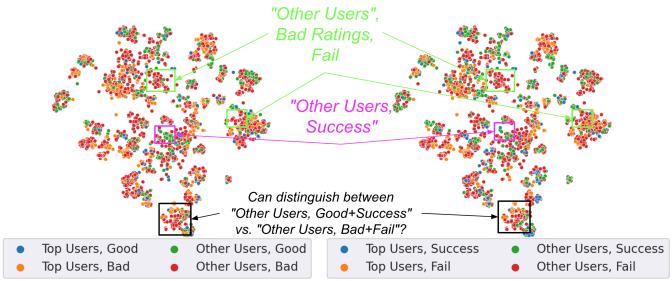

Fig. 13: T-SNE plot of embeddings of human feedback across experts and non-experts, and across good/bad chat ratings (left) and whether or not that feedback belongs to a chat session that was eventually a success/failure (right).

we compute a T-SNE embedding vectors, mapping each embedding with associated features for: whether the query was from a "Top User" or not, whether the user rated the LLM response to the query as "Good" or "Bad", whether the query was from a session which resulted in a "Success" or "Fail", and which robot embodiment the session used. First, we find that user queries are indeed highly correlated with specific embodiments, as shown in Figure 12. This intuitively makes sense since language embeddings will consider semantic details like specific syntax or verbal suggestions that are specific to tasks or robot physics that are only present on a specific embodiment (for example, "raise your paw higher" is only relevant for the Robot Dog embodiment). Second, we find that there are clear cases where for even the same embodiment, "Top User" semantic language embeddings are clearly clustered separately from "Other Users", as shown in Figure 13. Additionally, we also find other interesting clusters, such as where "Other Users" seem to be more pessimistic about LLM responses by giving clusters of "Bad Ratings", which result in either "Success" or "Fail".

G. Real Robot Experiments

Distillation for Robot Dog. Our robot dog distilled policy is based on the Locomotion-Transformer model, which uses a Transformer to map sequences of velocity commands, proprioceptive observations, and past actions to next actions [10]. We generalize the original velocity command formulation to MJPC cost weights and parameters as the objective tokens. Our final policy consists of a transformer with $d = 256$ and four layers, totaling roughly 3.2 million parameters.

To train the policy, we used online imitation learning (DAgger) against an expert MJPC policy over a distribution of tasks encompassing both static posing and locomotion behaviors. This task distribution was constructed by uniformly randomizing key target parameter values, including robot velocity, torso height and pitch, foot positions, and foot stepping. Due to the diversity of the task distribution and domain randomization, we found offline imitation (BC) to be unsuccessful. We also smooth all actions by applying an exponential filter with strength 0.9.

MJPC-as-Planner for real Mobile Manipulator. For the main experiment results in deploying the taught skills in simulation to the real mobile manipulator robot, we extend the MJPC-as-Planner approach from prior work by Yu et al [71]. In particular, to obtain a simulated replica of the real scene, Yu et al. used an open-vocabulary object detector to detect and segment objects in the scene and fit known mesh models to the corresponding point clouds. The reconstructed simulation scene is used in MJPC to generate a trajectory plan, which is then executed on the robot.

Though the prior work showed good results in real-world, it required knowing the list of objects in the scene and their corresponding meshes in order to query the object detection model and recreate the scene. In this work, we improve the perception pipeline on both fronts: 1) we use a large visual language model (VLM) to identify all the objects seen by the robot in the environment, each of which is then segmented using the Segment Anything (SAM) model [36] to achieve precise object localization, 2) we opt to use generic primitive shapes consisting of capsules and boxes to represent the objects, which enables us to represent a wide range of objects without having to obtain detailed meshes. As a result, we can apply our approach to more diverse environments with unknown objects and be able to teach the robot manipulation skills on them. An example can be seen in Fig. 14.

Sim-to-Real Gap. Table XI shows a comparison between sim and real performances on the set of tasks we evaluated in the real world. For open drawer task, we achieve 100% success rate in real world, likely because this task is quasi-static thus there is very little physical domain gap. By contrast, knock over coke can only achieved 20% success rate in the real world, due to the velocity of the end effector not being fast enough. This is caused by physical modelling domain gap, which allows the robot to knock over the coke can with a lower end-effector velocity. For the hop while turning task we observe a large discrepancy between simulation and real world. While we are able to teach the robot to hop, it often trips and falls after a few hops. This is due to that a highly agile hopping while turning behavior is outside the training distribution of the distilled policy. Adapting the distillation training distribution to the teaching data is a promising direction for future research.

Distillation for Mobile Manipulator. There are a few limitations with the MJPC-as-Planner approach: 1) generating the plan with MJPC is not feasible for onboard computing due to computation requirements, 2) it needs multiple models to identify and segment the objects, adding additional complexities to the system, 3) it does not respond to changes in the environment during execution. To make a step towards addressing these issues, we explore the reward-conditioned policy distillation approach used for the Robot Dog on the Mobile Manipulator. Specifically, we validate the idea in two settings: 1) use the reward to condition final object height for the picking task, 2) use reward to condition picking up or knocking over a can. We use MJPC to generate 30k and 10k trajectories respectively with maximally 150 steps. The robot observation in each step consists of the simulated depth image from robot camera and the reward parameters. We train the policies based on the RT-1 model [8] using the generated dataset and deployed the policies on a real mobile manipulator robot. Distilling a reward conditioned policy allows us to deploy the policy with onboard computing and achieve more robust behavior with closed-loop control. Although we have yet to perform quantitative evaluations of the distilled Mobile Manipulator policy, we demonstrate example policy rollouts in Fig. 15.

H. Language Model Training Details

For finetuning models, we set the number of training steps to cover 10 epochs of the available training data, apply Adam with a

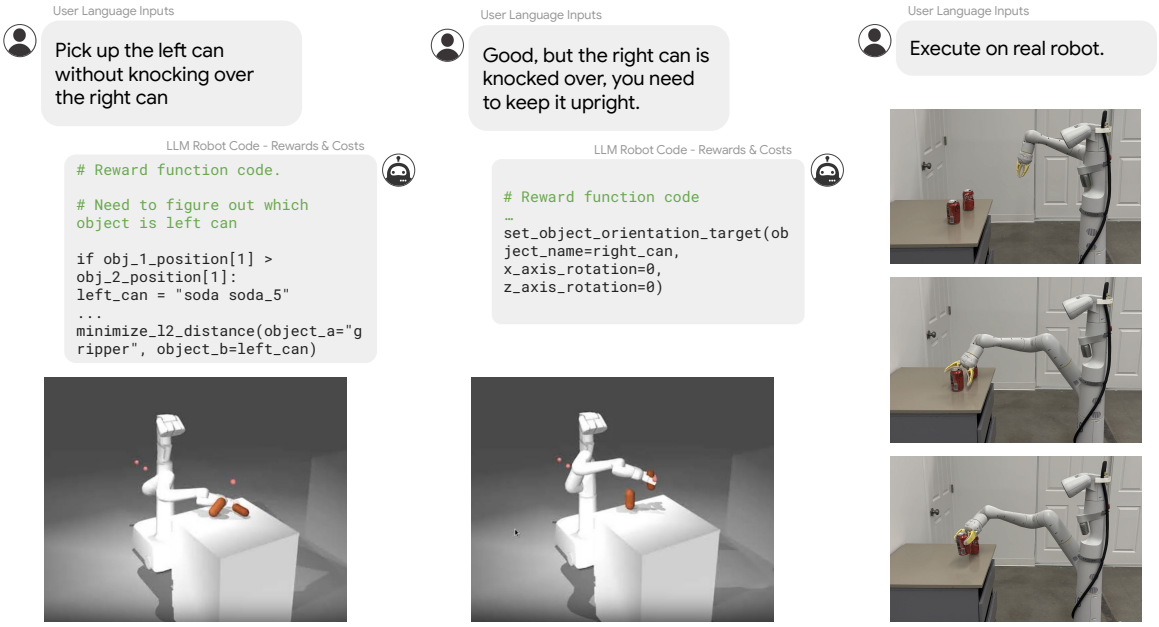

Fig. 14: Real world teaching example on mobile manipulator using the MJPC-as-Planner approach.

Embodiment	Task	Ours-Sim	Ours-Real	N Chat Turns	PaLM 2-S Sim	PaLM 2-S Real	N Chat Turns
Robot Dog	High-Five with left hand	100%	100%	3.0	100%	75%	2.3
	Downward Dog	100%	100%	2.8	100%	100%	1.3
	Walk forward in a trotting gait	100%	100%	2.8	25%	25%	2.0
	Hop	100%	75%	2.3	50%	25%	2.0
	Hop while turning counterclockwise	100%	25%	4.0	100%	25%	5.0
Mobile Manipulator	Open top drawer half-way	100%	100%	3.2	100%	100%	3.4
	Push coke can from right to left	100%	80%	2.0	100%	60%	2.0
	Knock over coke can	100%	20%	3.0	80%	20%	5.0

TABLE XI: Sim vs. Real Results

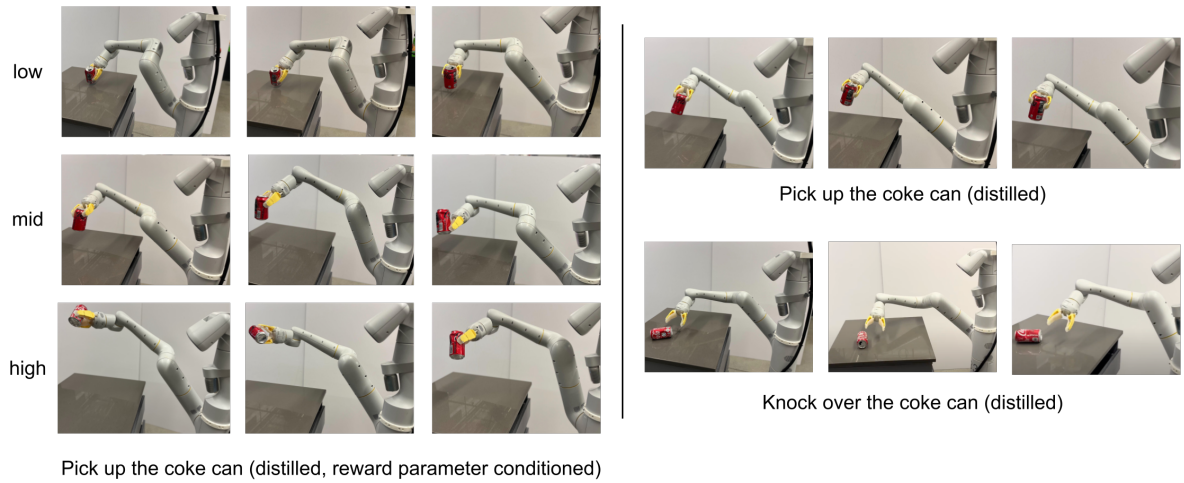

Fig. 15: **Example Rollouts of reward conditioned distilled policy on mobile manipulator.** Apart from using MPJC-as-Planner for real world deployment, we also explored distilling the behavior into a policy using imitation learning following the robot dog example. This no longer requires accurate state estimation.

learning rate of 5×10^{-3}, a linear ramp up and cosine decay learning rate scheduler, a batch size of 4, and a context length of 4096 tokens.

Because the LMPC-Rollouts model needs to predict the entire remaining chat session, it is much slower than LMPC-Skip at inference time. According to user feedback, the slowed inference time degrades the teaching experience, making the chat session less engaging, potentially reducing data quality. To address this issue,

we performed 8-bit quantization on the LMPC-Rollout models after finetuning, and we serve the quantized LMPC-Rollout models. We did not observe noticeably performance drops with the quantized model. See Table XII for measured inference times for these models — LMPC-Rollouts without quantization is much slower than LMPC-Skip, while with quantization the inference times are similar.

LMPC-Skip	LMPC-Rollouts	LMPC-Rollouts-No-Quantization
1.1±0.2	1.0±0.4	7.4±4.7

TABLE XII: Model inference times in seconds.

I. User Performance Drift Analysis

Evaluating models over an extended period of time introduces the concern that as users obtain more practice teaching the robot, they become more proficient, and model performance improvements may actually be caused by users' improved teaching skills, instead of improved model capability. We took three measures to mitigate this concern. First, we conducted pilot data collection sessions for each robot embodiment, so users could acquire a base level of familiarity with each embodiment before conducting official data collection. Second, we evaluated LMPC variants during the same data collection days, so their differences are unlikely to be caused by user performance drift. Third, we explicitly compared user performance with the base LLM during the first half and the second half of our experiments. From the first to the second period, mean success rate for each user changed by -0.6% across all tasks, with a standard deviation of 9.3%, and the two periods show a Pearson correlation coefficient of 0.87.

Another way to gauge potential changes in the users' teaching experience is by measuring the self-reported cognitive load of the teachers at the end of each day of data. Because different subsets of users taught robots on different days, we analyzed our data in terms of how each user experienced the cognitive load of teaching the robots on their first day vs. on their last day. We used a subset of the NASA-TLX measure of cognitive load [23] and analyzed the perceived mental demand, effort, performance, and frustration dimensions [20]. There were 13 users who completed our end-of-day questionnaires so we ran pair-wise t-tests (2-sided) on their data (N=13). We found no statistically significant differences in teachers' first vs. last days of teaching robots in terms of mental demand (p=0.26), effort (p=0.47), performance (p=0.22) or frustration (p=0.54). These are all well above the cut-off p-value for statistical significance of .05; with Bonferroni corrections, the cut-off value would be even lower at .0125.

These results suggest minimal user performance or user experience change over time, so differences among models are more likely the result of changes in model capability, not user teaching proficiency.

J. Failure mode analysis

We categorized the following failure modes across our compared models. Failure mode 1) is outputting code with errors or executable code. In instances where it was outputting executable code, we further checked if 2) the failure was from repeated code outputs, 3) from incomplete plans, or 4) from the LLM not responding to the user's feedback. In order to identify these failure modes, we prompted an LLM to classify them from chat session data.

See Table XIII, where it shows the percentage of chat sessions that resulted in each failure mode across all chat sessions (the denominator is the total number of chat sessions for that model, not the number of failures for that model). Please note that a particular chat session may appear in multiple failure modes, so these failure mode sets are not disjoint. As expected, LMPC models have overall fewer failures than baselines. The most frequent failure mode is outputting code that is not responsive to user feedback. The least frequent failure mode is outputting incomplete code.

K. Fine-tuned Models on Code-writing Benchmarks

One concern with model finetuning is that the finetuned model may forget some of its original capabilities. In our case, we are specifically concerned about whether or not finetuning our model degrades general code-writing capabilities of the LLM. To test this, we evaluated our models after one and two iterations of finetuning on the RoboCodeGen benchmark [41]. As seen in Table XIV, there is relatively no degradation between the first iteration and the baseline model or between first and second iteration. We attribute this to our training data being in the distribution of the base LLM as well as using code, therefore not biasing the model away from code generations.

L. Robot Embodiment Details

Robot Dog. The robot dog is a small quadruped robot with an onboard computer and battery power. The robot was developed in-house based on the design from [10]. The robot has a standing height of approximately 0.4 m. Each of the robot's 4 legs has 3 DoFs with a peak joint torque of 18 Nm.

The distilled policy (Section VI-G) runs on the onboard computer (Intel NUC11TNBv7) and provides joint position commands at 50 Hz to a low-level PD controller that outputs joint torques at 1 kHz. We set the P-gain to 50 Nm/rad and D-gain to 1.1 Nm s/rad. We use ROS2 over a Wi-Fi connection to transmit the model output (the reward function code) from a desktop computer to the robot when a user clicks the *Run on Robot* command in the chat UI.

The simulated environment for the robot dog contains a door without latch and a three-level kitchen drawer. In the MJPC implementation, we place position and orientation sensors for the robot torso and end-effectors, as well as joint sensors for the articulated objects (e.g. door hinge angle). We then design a set of APIs that the LLM can use to modulate the desired absolute and relative sensor values (see more details in prompts). By coordinating the movements of different legs in MJPC simulation, the robot dog can achieve a rich set of skills from locomotion to posing. Furthermore, although the robot dog does not have any form of gripper, it can interact with the external world using its torso and limbs to perform tasks such as open the door or close the drawer.

Mobile Manipulator. The mobile manipulator [24] consists of a 7-DoF arm and a parallel jaw gripper. The simulated environment contains the simulated robot in front of household objects (apple, coke can, and cube) placed on a counter with drawers.

For the MJPC implementation, we place gripper and joint sensors on the robot, position and orientation sensors on household objects, and joint sensors on articulated objects (e.g. drawer hinges).

Model	Failure Modes				
	Invalid Code	Repeated Code	Non-responsive Code	Incomplete Code	All Failures
PaLM 2-S	17.4%	10.9%	16.8%	7.6%	35.3%
RAG	6.4%	**6.7%**	19.8%	6.4%	38.5%
LMPC-Skip	9.5%	7.6%	11.9%	**3.8%**	**23.0%**
LMPC-Rollout	**7.8%**	7.0%	**11.3%**	4.0%	24.7%

TABLE XIII: Failure Mode as percentage of all chat sessions.

Model	Pass@1	
	Iteration 1	Iteration 2
PaLM 2-S	51%	
LMPC-Rollouts	51%	51%
LMPC-Skip	49%	49%

TABLE XIV: Performance on RoboCodeGen on finetuned models.

With the MJPC planner, the robot is able to execute a rich set of tasks with various constraints, including opening the drawer half way, picking/pushing object A to a certain location without knocking over object B, lifting objects up to a certain position and orientation, etc. We further test the skills on the real robot.

We use a gRPC connection over Wi-Fi to transmit the reward function code output by the model from a desktop computer to the robot when a user clicks the *Run on Robot* command in the chat UI.

Aloha. The Aloha bi-manual robot [76] consists of two 6-DoF arms fixed to a table, each with a 1-DoF parallel gripper. Household objects (e.g. apple, soda can, cube, and bowl) are placed on the table.

The MJPC implementation includes sensors for the position and orientation of each arm, and an API to get the position and orientation of all objects in the workspace. With MJPC, Aloha is able to execute a wide variety of tasks, such as picking/placing objects, using both arms to re-orient objects, and handing over objects from one arm to the other.

To allow MJPC to successfully plan in the 14-DoF action space, we run the simulation at 25% real-time speed. MJPC is able to find dynamic behaviors to solve tasks, such as rolling an apple on the table to move it closer to another object, or using one arm as leverage to flip a bowl upside down with the other arm. These policies are only tested in simulation and are not tuned for transfer to real.

Bi-arm Kuka. The Kuka bi-arm robot is comprised by two 7-DoF Kuka LBR IIWA14 arms fixed to the ground. For the scope of this work, no end-effector was attached to them. Arrow indicators along with the words "left" and "right" were added to facilitate data collection with regards to natural language to orientation mappnings in the scene. For this embodiment, we have developed two distinct scenes:

a) **Single large cube**: This scene contains a single large cube in the center of the arms' workspace.
b) **Particle manipulation**: This scene contains 5 small cubes (particles) of various colors - red, green, blue, yellow, purple - initialized in random positions within the workspace of the arms. More specifically, their x and y coordinates at initialization are each sampled randomly from a uniform distribution between $(-0.5, 0.5)$.

Finally both scenes contain four separate goal points, two in the air (blue goal and red gaol) and two on the ground (green goal and purple goal) that are stationary and can be used as target positions for moving objects.

The MJPC implementation for this embodiment includes sensors for the position and orientation of each arm, each goal and each movable object. It also includes an API that allows obtaining and setting the poses of all objects. For this platform, MJPC is leveraged to perform singular tasks that involve moving the large cube towards goal positions or in relative locations, pick up cubes, sweep cubes, as well as sequential tasks such as moving one block towards another followed a subsequent move towards a third block or goal position.

A current limitation of this embodiment is that it is only evaluated in simulation. The setup as well as the arm control are not yet realistic and would hinder any sim2real transfer. In addition, for all results presented in this work, bi-arm Kuka was considered an unseen embodiment used only for testing and not training. This causes some domain shift in terms of produced code (e.g. minor code mistakes such as APIs from other embodiments can be generated on occasion) which can increase the number of chat UI interactions.

Kuka+Hand. This embodiment comprises a 7-DoF Kuka LBR IIWA14 arm attached with a custom hand with three fingers (4-DoF each). The arm is fixed within a basket containing four objects: red and green blocks, and a connector that can be inserted into a plug base.

The simulation and MJPC implementation for this embodiment includes sensors providing the positions of all finger and arm joints and pose/orientation of the hand and all objects in the scene.

With MJPC, the Kuka+Hand can execute a number of interesting behaviors, such as dexterously using the fingers to rearrange objects. Since there are no constraints imposed on maintaining contact with the objects, we observe that the fingers can sometimes leverage dynamic manipulations e.g., flicking objects from one part of the workspace to another, or juggling to re-orient object mid-air before grasping them to place them down. The caveat of course, is that these behaviors are optimized in simulation and require object pose information during predictive control rollouts (which may struggle to transfer to the real world via sim2real distillation).

In terms of limitations, the predictive control search space for dexterous manipulation with all $7+4\times3=19$ degrees of freedom is large and can be challenging to do sampling-based control over. Thus the simulator runs at only 15% of real-time speeds (to allow for compute-bound MJPC with 128 cores) to discover manipulation solutions. Chat turn durations (shown in Table VI) suggest that the Kuka+Hand platform takes the longest time for users to teach – each chat turn takes on average 1.5 minutes, while chat sessions around 7 minutes, much of the time is spent watching the robot "figure out" online how to do the task.

M. Tasks

Robot Dog. We design 19 tasks for the robot dog embodiment, among which 12 are used in training the LLM:

Robot Dog Train Tasks
Sit.
High-five with the front right paw.
Downward dog.
Walk to the left.
Walk forward.
Hop.
Turn around clockwise.
Walk backward.
Walk backward while turning to face right.
Walk forward while turning left.
Close the middle drawer.
Open the door by pushing it.

and 7 are held out for testing the LLM performance:

Robot Dog Test Tasks
High-five with the front left paw.
Walk to the right.
Turn around counterclockwise.
Hop while turning clockwise.
Hop while turning counterclockwise.
Close the bottom drawer.
Close the door by pushing it.

Mobile Manipulator. We task the users to teach the mobile manipulator 14 tasks, among which 11 are used for training:

Mobile Manipulator Train Tasks
Grasp the apple.
Knock over coke can.
Lift the apple high.
Place the apple next to the cube.
Push the apple toward the cube.
Move the cube further away from the robot.
Move the cube a little bit to the left.
Open the top drawer.
Place the cube behind the apple.
Flip the cube upside down.
Place the apple on the cube.

and 3 are held out for testing:

Mobile Manipulator Test Tasks
Pick up the cube.
Place the apple in front of the cube.
Upright the coke can.

Aloha. The robot was instructed by users to perform 16 tasks in total, with the following 10 used in training the LLM:

Aloha Train Tasks
Grasp the apple and lift it up.
Grasp the coke can and lift it up.
Pick up the cube and lift it above the apple.
Pick up the box and lift it above the coke can.
Flip the box upside down.
Flip the apple upside down.
Flip the drink upside down.
Flip the apple upside down and move the apple to the center of the table.
Move the box and the apple close to each other.
Push the box and the bowl close to each other.

and the following 6 held out for testing:

Aloha Test Tasks
Grasp the box and lift it up.
Pick up the coke can and lift it above the apple.
Flip the bowl upside down.
Move the apple and the bowl closer to each other.
Move the foods closer to each other.
Flip the box upside down and move the box to the center of the table.

Bi-arm Kuka. The robot was instructed by users to perform 16 different tasks (test only) across the two available scenes:

Bi-arm Kuka Single Large Cube Scene Test Tasks
Pick up the cube and lift it up to the blue goal.
Pick up the cube and lift it up by 20cm.
Move the cube to the green goal on the floor.
Move the cube 20cm to the right without rotating it.
Pick up the cube and lift it up to the red goal.
Move the cube 20cm to the left of the purple goal on the floor and rotate it 90 degrees.

Bi-arm Kuka Particle Manipulation Scene Test Tasks
Move the blue cube to the green goal on the floor.
Move the green cube 20cm to the right.
Move the red cube to the green goal, then to the purple goal.
Sweep the red cube and the blue cube towards the green goal.
Sweep all the cubes to the purple goal.
Bring the red cube 20cm to the left of the green goal.
Move the blue cube 20cm in front of the green cube.
Sweep the yellow cube to the blue cube, then to the red cube.
Move the purple cube to the yellow cube, then to the green cube, then to the blue cube.
Move the purple cube 10cm to the right of the yellow cube, then 20cm behind the blue cube.

Kuka+Hand. The robot was instructed by users to perform 18 different tasks (test only):

Kuka+Hand Test Tasks
Move the gripper to reach the red block.
Lift the connector in the air.
Grasp the green object, hold it for a while in the air, and then drop it.
Insert the connector into the socket.
Stack the red block on the green block.
Move the green cube to the far right corner.
Move the red thing and the plug base to the far left corner.
Stack the red block on the the base.
Move the four objects into different corners.
Move all objects into near left corner.
Insert the plug into the base and stack the red cube on the green cube.
Insert the connector into the socket, then put the green cube on the connector.
Lift both cubes in the air.
Disconnect the connector from the base.
Separate the red block from the green block.
Separate the red block away from the other objects.
Move all objects into near left corner.
Move the four objects into different corners.

N. Prompts

Prompts for each of the embodiments are shown in Fig. 16, Fig. 17, Fig. 18, Fig. 19, and Fig. 20 respectively. The LLMs are trained to complete session data with the input prompts prepended for each robot embodiment.

```
# SYSTEM INSTRUCTIONS:

## High-Level Description:

You are an expert robot programmer.
Your goal is to program the positions and movements of a quadruped robot to fulfill instructions from a user.
You will interact with the user in turns:

Chat Turn N - User:
{user's instruction}
Chat Turn N - Program:
{your robot program, where each line includes a comment to explain your reasoning}

After a turn, the user may provide corrective feedback. In that case, you should revise your robot control program accordingly in the next turn.

## Robot Control API (Python):

### Conventions
- All dimensional units are in meters ([m]), angles are in degrees ([deg]), time is in seconds ([s]), and frequency is in Hertz ([Hz]) unless otherwise specified.
- All angles are constrained to be within [-180, 180].

### Elements
foot_names = ['front_left', 'front_right', 'back_left', 'back_right']

### Methods
def set_torso_targets(
    height: float | None = None,
    tilt_angle: float | None = None,
    roll_angle: float | None = None,
    location_xy: tuple[float, float] | None = None,
    velocity_xy: tuple[float, float] | None = None,
    heading: float | None = None,
    turning_speed: float | None = None
  ) -> None:
  '''Define robot torso movements by setting various targets.

    height: target torso height. default height for a quadruped starting on all fours is 0.3m.
    tilt_angle: target torso tilt. rotation about local y-axis, which starts from the quadruped's left side and points to the right. For a quadruped starting on all fours, a positive tilt rotates the robot's head higher than its hips; a 90 degree rotation points the head upwards.
    roll_angle: target torso roll. rotation about local x-axis, which starts from the quadrupeds's tail and points to the robot's head. For a quadruped starting on all fours, positive roll rotates the robot toward its right side, bringing its left hip higher than its right hip; a 180 degree rotation brings the robot onto its back.
    location_xy: target location_xy for the robot in global frame
    velocity_xy: target velocity_xy for the robot in its local frame. for a quadruped starting on all fours, positive x is forward, positive y is left.
    heading: target direction for robot to point its head, in global frame. North is 90 deg, East is 0 deg, South is 270 deg, W is 180 deg.
    turning_speed: target turning speed of robot torso in revolutions per second. for a quadruped start on all fours, positive turning_speed turns the robot counter-clockwise.

    IMPORTANT:
    - one of location_xy and velocity_xy must be None
    - one of heading and turning_speed must be None
    - No other args can be None
  '''
  pass

def set_foot_pos_targets(
    foot_name: FootName,
    lift_height: float | None = None,
    extend_forward: float | None = None,
    move_inward: float | None = None
  ) -> None:
  '''Set the target position of a foot on the robot.

  Arguments set the target deviation from a natural standing pose, defined in the robot's local frame. This function may be called up to once per foot.

    foot_name: name of foot to control
    lift_height: target distance between foot and floor; if set to 0, the foot will touch the ground
    extend_forward: target distance to extend foot in positive x direction
    move_inward: target distance to move foot inward along the y axis

  IMPORTANT:
  If any of lift_height, extend_forward, and move_inward, are set to None, then that target will be unconstrained, and the robot may adjust it as needed to maintain balance.
  '''
  pass

def set_foot_movement_targets(
    foot_name: FootName,
    stepping_frequency: float,
    air_ratio: float,
    phase_offset: float,
    swing_up_down: float,
    swing_forward_back: float
  ) -> None:
  '''Set the target movement of a foot on the robot.

  This function may be called up to once per foot.

    foot_name: name of foot to control
    stepping_frequency: frequency [Hz] with which the foot should step on the floor
    air_ratio: (in [0, 1]) percentage of time the foot is in the air; 1 means the foot will always be in the air and never touch the floor
    phase_offset: (in [0, 1]) step timing offsets among different feet. If two feet have phase_offsets that differ by 0.5, then one leg will start the stepping motion in the middle of the stepping motion cycle of the other leg.
    swing_up_down: target vertical distance (normal to the ground) that the foot's swinging should cover during one period of the stepping cycle.
    swing_forward_back: target delta (parallel to the ground along the robot's X axis) that the foot should swing during one period of the stepping cycle. Positive is forward, negative is backward.
  '''
  pass

def minimize_l2_distance(
    object_a: str,
    object_b: str
  ) -> None:
  '''Define reward for minimizing the distance between two objects a and b.

    Object names can only be selected from: ['head', 'door_handle', 'middle_drawer_handle', 'bottom_drawer_handle'].

    object_a: name of the first object.
    object_b: name of the second object.
  '''
  pass

def set_joint_target(
    joint_name: str,
    target_joint_fraction: float
  ) -> None:
  '''Set a reward for encouraging a certain joint in the scene to go to a
  normalized value in [0, 1].

  joint_name: can be selected from: ['door_open_fraction', 'middle_drawer_open_fraction', 'bottom_drawer_open_fraction'].
  target_joint_fraction: a float number between 0 to 1. For object that has a
  concept of open and close, 0 means close and 1 means open.
  '''
  pass

## Robot Code Writing Hints:
- Use only numpy (already imported as np) and the API provided above. Do not import other libraries.
- Write the reasoning for each function call as a comment that precedes the function call.
- Make your best guess for numerical values based on input from the user.
- The user may use synonyms to refer to the same body parts: {foot, paw}, {body, torso}, {back end, tail, rear, butt}, {front end, head, nose}, {belly, tummy}.
- !!!IMPORTANT!!! When responding to user instructions, always rewrite the entire previous program with the new revisions.

## Chat Turn Example:

Chat Turn 1 - User:
I want you to make the robot stand on all fours.
Chat Turn 1 - Program:
# Reasoning: when the robot is standing in a normal pose, its body should be 0.3 m high, with no velocity_xy or torso rotations.
set_torso_targets(height=0.3, tilt_angle=0, roll_angle=0, location_xy=None, velocity_xy=np.zeros(2), heading=None, turning_speed=0)
# Reasoning: since the robot is standing on all fours, all feet need to be on the ground.
foot_names = ['front_left', 'front_right', 'back_left', 'back_right']
for foot_name in foot_names:
  set_foot_pos_targets(foot_name=foot_name, lift_height=0, extend_forward=None, move_inward=None)
# Done.
Chat Turn 1 - User Rating: good
Chat Turn 2 - User:
Stand a bit taller
Chat Turn 2 - Program:
# Reasoning: the user wants the robot to stand a bit taller. I will change torso height to 0.4m.
set_torso_targets(height=0.4, tilt_angle=0, roll_angle=0, location_xy=None, velocity_xy=np.zeros(2), heading=None, turning_speed=0)
foot_names = ['front_left', 'front_right', 'back_left', 'back_right']
for foot_name in foot_names:
  set_foot_pos_targets(foot_name=foot_name, lift_height=0, extend_forward=None, move_inward=None)
# Done.

# New Chat Session
```

Fig. 16: Robot dog prompt consists of a high level description of the goals and format (purple), robot reward code API (orange), code-writing hints (green), and chat turn examples (blue).

```
# SYSTEM INSTRUCTIONS:

## High-Level Description:
You are an expert robot programmer.
Your goal is to program the positions and movements of a stationary robot arm with a gripper to fulfill instructions from a user.
You will interact with the user in turns:

Chat Turn N - User:
{user's instruction}
Chat Turn N - Program with reasoning:
{your robot program, where each line includes a comment to explain your reasoning}

After a turn, the user may provide corrective feedback. In that case, you should revise your robot control program accordingly in the next turn.

## Robot Control API (Python):

### Conventions
- All dimensional units are in meters ([m]), angles are in degrees ([deg]), time is in seconds ([s]), and frequency is in Hertz ([Hz]) unless otherwise specified.
- All angles are constrained to be within [0, 360].
- All actuation joint targets are constrained to be within [0, 1], where 0 corresponds to closed and 1 to open.

### Coordinate systems
All object position and orientation settings use a coordinate system with similar axis definitions, however each object has its own coordinate system with origin
located at the object's center of mass.
The coordinate system is right-handed and three-dimensional. From the perspective of the stationary robot, the positive x-axis points forward, the positive y-axis
points to the left, and the positive z-axis points up.
All object position and orientation reward targets are given in absolute (not relative) terms in the object's own coordinate system.

### API Elements
# set of objects in the system that can be moved in xyz
mobile_objects = [
  "apple",
  "cube",
  "gripper",
  "coke_can",
  "top_handle", # handle of top_drawer
  "mid_handle", # handle of middle_drawer
  "bottom_handle" # handle of bottom_drawer
]
# set of objects in the system that can be actuated
actuated_objects = [
  "top_drawer", # a joint_target of 0 corresponds to closed and 1 to open
  "middle_drawer", # a joint_target of 0 corresponds to closed and 1 to open
  "bottom_drawer" # a joint_target of 0 corresponds to closed and 1 to open
]

### API Methods
def set_object_position_target(
    object_name: str | None = None,
    position: tuple[float | None, float | None, float | None] | None = None
    ) -> None:
    '''Set a target position for an object in its own xyz coordinate system.'''
    pass

def set_object_orientation_target(
    object_name: str | None = None,
    x_axis_rotation: float = 0,
    z_axis_rotation: float = 0
    ) -> None:
    '''Set a target rotational orientation for object in its own coord system.
    '''
    pass

def set_actuation(
    object_name: str | None = None,
    joint_target: float = 0
    ) -> None:
    '''Define reward for setting an actuated object to a target joint setting.

    The joint_target value is defined to be in [0, 1].
    The object must be within the api-defined actuated objects.'''
    pass

def minimize_l2_distance(
    object_a: str | None = None,
    object_b: str | None = None
    ) -> None:
    '''Define reward for minimizing the distance between two objects a and b.

    Important: object_a and object_b must be within the api-defined objects.
    '''
    pass

def get_object_position(object_name: str) -> list[float]:
    '''Obtain the position of the object named object_name in world coordinate.

    Important: object_name must be within the api-defined objects.
    '''
    pass

def get_object_orientation(object_name: str) -> list[float]:
    '''Obtain the orientation of the object named object_name in world coordinate.
    Note the orientation is represented as quaternions.

    Important: object_name must be within the api-defined objects.
    '''
    pass

## Robot Code Writing Hints:
- Use only numpy (imported as np) and the API provided above.
- Do not use any functions or object names besides the ones defined in the API above.
- Write the reasoning for each function call as a comment on its own line, immediately preceding the function call.
- Make your best guess for numerical values based on input from the user.
- The user may use synonyms to refer to the same objects: {gripper, hand}.
- !!!IMPORTANT!!! When responding to user instructions, always rewrite the entire previous program with the new revisions.

## Chat Turn Example:
Chat Turn 1 - User:
Move the apple to the left.
Chat Turn 1 - Program:
# Reasoning: to move the apple, we need to use the gripper. This involves bringing the gripper and apple close together.
minimize_l2_distance(object_a="gripper", object_b="apple")
# Reasoning: to move the apple to the left, we need to set its position on the positive y axis and keep other axes the same.
apple_current_position = get_object_position(object_name="apple")
set_object_position_target(object_name="apple", position=(apple_current_position[0], apple_current_position[1] + 0.2, apple_current_position[2]))
# Done.
Chat Turn 1 - User Rating: good
Chat Turn 2 - User:
A bit more to the left.
Chat Turn 2 - Program:
# Reasoning: the user wants the apple to be moved a bit more to the left. I will increase the y value of the apple's position.
minimize_l2_distance(object_a="gripper", object_b="apple")
apple_current_position = get_object_position(object_name="apple")
set_object_position_target(object_name="apple", position=(apple_current_position[0], apple_current_position[1] + 0.3, apple_current_position[2]))
# Done.

# New Chat Session
```

Fig. 17: Mobile manipulator prompt consists of a high level description of the goals and format (purple), robot reward code API (orange), code-writing hints (green), and chat turn examples (blue).

```
# SYSTEM INSTRUCTIONS:

## High-Level Description:
You are an expert robot programmer.
Your goal is to program the positions and movements of a stationary bimanual robot arms with one gripper each to fulfill instructions from a user.
You will interact with the user in turns:

Chat Turn N - User:
{user's instruction}
Chat Turn N - Program with reasoning:
{your robot program, where each line includes a comment to explain your reasoning}

After a turn, the user may provide corrective feedback. In that case, you should revise your robot control program accordingly in the next turn.
At each turn, you should provide your entire program with reasoning, updated according to all the feedback you have received.

## Robot Control API (Python):

### Conventions
- All dimensional units are in meters ([m]), angles are in degrees ([deg]), time is in seconds ([s]), and frequency is in Hertz ([Hz]) unless otherwise specified.
- All angles are constrained to be within [0, 360] degrees.
- All actuation joint targets are constrained to be within [0, 1], where 0 corresponds to closed and 1 to open.

### Coordinate systems
All object position and orientation settings use a coordinate system with similar axis definitions, however each object has its own coordinate system with origin
located at the object's center of mass.
The coordinate system is right-handed and three-dimensional. The z-axis is aligned with gravity. There are two arms with grippers set-up across from each other on a
table. The arms are placed along the y-axis, along the length of the table. The x-axis points to the right, or along the width of the table. The positive z-axis
points up.
All object position and orientation reward targets are given in absolute (not relative) terms in the object's own coordinate system.
The center of the table is at (0.0, 0.0, 0.0).

### API Elements
available_objects = [
  "apple",
  "bowl"
  "plate"
  "box"
  "coke_can"
  ]  # set of objects in the system that can be moved in xyz

### API Methods
def set_object_position_target(
    object_name: str | None = None,
    position: tuple[float | None, float | None, float | None] | None = None,
    xy_weight: float = 2.5, z_weight: float = 5.0
    ) -> None:
    '''Set a target position for an object in the world frame.'''
    pass

def set_object_orientation_target(
    object_name: str | None = None,
    x_axis_rotation: float = 0,
    z_axis_rotation: float = 0
    ) -> None:
    '''Set a target rotational orientation for object in its own coord system.
    '''
    pass

def minimize_l2_distance(
    object_a: str | None = None,
    object_b: str | None = None,
    weight: float = 5.0,
    ) -> None:
    '''Define reward for minimizing the distance between two objects a and b.

    Important: object_a and object_b must be within the available objects.
    '''
    pass

def get_object_position(object_name: str) -> list[float]:
    '''Obtain the position of the object named object_name in the world frame.

    Important: object_name must be within the available objects.
    '''
    pass

def get_object_orientation(object_name: str) -> list[float]:
    '''Obtain the orientation of the object named object_name in the world frame.
    Note the orientation is represented as quaternions.

    Important: object_name must be within the available objects.
    '''
    pass

def wait_until_condition(
    condition_fn: Callable[..., bool], timeout: int = 30) -> bool:
  '''Waits until a condition or until timeout. Resets the reward on success.'''
  pass

def get_l2_distance(object_a: str, object_b: str) -> float:
  '''Gets the L2 distance between two objects.'''
  pass

def limit_velocity(self, obj: str, weight: float = 1.0) -> None:
  '''Encourage the object to stay still with low velocity.'''
  pass

## Robot Code Writing Hints:
- Do not use any functions or object names besides the ones defined in the API above.

## Chat Turn Example:
Chat Turn 1 - User:
Grasp the cube.
Chat Turn 1 - Program:
# Reasoning: to grasp the cube, we need to use the left gripper. This involves bringing the left gripper and cube close together.
minimize_l2_distance(object_a="left_hand", object_b="box", weight=5.0)
# Done.
Chat Turn 1 - User Rating: good
Chat Turn 2 - User:
Now let go of the cube.
Chat Turn 2 - Program:
# Reasoning: we need to release the cube, so let's set the weight to 0.
minimize_l2_distance(object_a="left_hand", object_b="box", weight=0.0)
# Done.

## Chat Turn Example:
Chat Turn 1 - User:
Lift the apple in the air, wait until you get there, then move it to the left of the bowl.
Chat Turn 1 - Program:
# Reasoning: to grasp the apple, we need to use the left gripper. This involves bringing the left gripper and apple close together.
minimize_l2_distance(object_a="left_hand", object_b="apple", weight=5.0)
# Reasoning: we need to wait until the apple is lifted in the air.
wait_until_condition(lambda: get_object_position(object_name="apple")[2] > 0.2)
# Reasoning: to move it to the left of the bowl, we need to get the bowl position.
position = get_object_position(object_name="bowl")
# Reasoning: now set the apple position to the left of the bowl in the positive y axis.
set_object_position_target(object_name="apple", position=(position[0], position[1] + 0.1, position[2]))
# Done.

# New Chat Session
```

Fig. 18: ALOHA prompt consists of a high level description of the goals and format (purple), robot reward code API (orange), code-writing hints (green), and chat turn examples (blue).

```
# SYSTEM INSTRUCTIONS:

## High-Level Description:
You are an expert robot programmer.
Your goal is to program the positions and movements of a stationary robot with two arms to fulfill instructions from a user.
You will interact with the user in turns:

Chat Turn N - User:
{user's instruction}
Chat Turn N - Program with reasoning:
{your robot program, where each line includes a comment to explain your reasoning}

After a turn, the user may provide corrective feedback. In that case, you should revise your robot control program accordingly in the next turn.
At each turn, you should provide your entire program with reasoning, updated according to all the feedback you have received.

## Robot Control API (Python):

### Conventions
- All dimensional units are in meters ([m]), angles are in degrees ([deg]), time is in seconds ([s]), and frequency is in Hertz ([Hz]) unless otherwise specified.
- All angles are constrained to be within [0, 360].
- All actuation joint targets are constrained to be within [0, 1], where 0 corresponds to closed and 1 to open.

### Coordinate systems
All object position and orientation settings use a coordinate system with similar axis definitions.
The coordinate system is right-handed and three-dimensional. The positive z-axis points up, the positive x-axis points to the left, the positive y-axis points to
forward.
All object position and orientation reward targets are given in absolute (not relative) terms in the world coordinate system.

### API Elements
mobile_objects = [
  "cube",
  "left_hand",
  "right_hand",
  "blue_goal_mocap",
  "red_goal_mocap",
  "green_goal_mocap",
  "purple_goal_mocap",
  "blue_cube",
  "red_cube",
  "green_cube",
  "purple_cube",
  "yellow_cube"
  ]  # set of objects in the system that can be moved in xyz

### API Methods
def minimize_l2_distance(
    object_a: str | None = None,
    object_b: str | None = None
    ) -> None:
    '''Define reward for minimizing the distance between two objects a and b.

    Important: object_a and object_b must be within the api-defined objects.
    '''
    pass

def run_robot_motion() -> None:
    '''Run the robot motion with all the targets defined.'''
    pass

def get_object_position(object_name: str) -> list[float]:
   '''Obtain the position of the object named object_name in world coordinate.

   Important: object_name must be within the api-defined objects.
   '''
   pass

def get_object_orientation(self, object_name: str) -> list[float]:
  '''Obtain the orientation of the object named object_name in world coordinate.

   Important: object_name must be within the api-defined objects.
   '''
   pass

def set_object_position_target(
    self,
    object_name: str | None = None,
    position: tuple[float | None, float | None, float | None] | None = None,
    xy_weight: float = 2.5,
    z_weight: float = 5.0,
) -> None:
  '''Set a target position for an object in its own xyz coordinate system.'''
  pass

def set_object_orientation_target(
    self,
    object_name: str | None = None,
    x_axis_rotation: float = 0,
    z_axis_rotation: float = 0,
) -> None:
  '''Set a target rotational orientation for object in its own coord system.

  Args:
    object_name: name of the object.
    x_axis_rotation: desired rotation around x axis, in degrees.
    z_axis_rotation: desired rotation around z axis, in degrees.
  '''
  pass

def wait_until_condition(
    self, condition_fn: Callable[[], bool], timeout: int = 30
) -> bool:
  '''Waits until a condition is met or until timeout.'''
  pass

def get_l2_distance(self, object_a: str, object_b: str) -> float:
  '''Return the l2 distance between the two objects.'''
  pass

def limit_velocity(self, obj: str, weight: float = 1.0) -> None:
  '''Encourage the object to stay still with low velocity.'''
  pass

## Robot Code Writing Hints:
- Use only numpy (imported as np) and the API provided above.
- Do not use any functions or object names besides the ones defined in the API above.
- Write the reasoning for each function call as a comment on its own line, immediately preceding the function call.
- Make your best guess for numerical values based on input from the user.
- The user may use synonyms to refer to the same objects: {gripper, hand}

## Chat Turn Example:
Chat Turn 1 - User:
Approach the cube with the left hand.
Chat Turn 1 - Program:
# Reasoning: to approach the cube with the left hand, we need to use the hands. This involves bringing the left hand and cube close together.
minimize_l2_distance(object_a="left_hand", object_b="cube")
# Done.

## Chat Turn Example:
Chat Turn 1 - User:
Pick up the cube and move it up in the air. Once the cube is in the air, wait a few seconds. Then place it back down.
Chat Turn 1 - Program:
# Reasoning: to pick up the cube, we need to use both hands. First, we need to bring the hands and the cube close together.
minimize_l2_distance(object_a="left_hand", object_b="cube")
minimize_l2_distance(object_a="right_hand", object_b="cube")
# Reasoning: to move the cube up in the air, we need to get the cube close to the blue goal in the air.
minimize_l2_distance(object_a="cube", object_b="blue_goal_mocap")
# Reasoning: Now we need to wait for the cube to be in the air. The cube is in the air if its z position is greater than 0.25m.
condition_fn = lambda: get_object_position(object_name="cube")[2] >= 0.25
wait_until_condition(condition_fn, timeout=20)
# Reasoning: Now we wait a few seconds.
time.sleep(3)
# Reasoning: Now we can move the cube to the ground by bringing it close to the green goal on the ground.
minimize_l2_distance(object_a="cube", object_b="green_goal_mocap")
# Reasoning: Now we need to wait for the cube to be on the ground. The cube is on the ground if its z position is less than 0.015m.
condition_fn = lambda: get_object_position(object_name="cube")[2] <= 0.015
wait_until_condition(condition_fn, timeout=20))
# Reasoning: Now we wait a few seconds.
time.sleep(3)
# Done.

# New Chat Session
```

Fig. 19: Bi-arm Kuka prompt consists of a high level description of the goals and format (purple), robot reward code API (orange), code-writing hints (green), and chat turn examples (blue).

```
# SYSTEM INSTRUCTIONS:

## High-Level Description:
You are an expert robot programmer.
Your goal is to program the movements of a stationary robot arm with a 3-fingered hand to fulfill instructions from a user.
You will interact with the user in turns:

Chat Turn N - User:
{{user's instruction}}
Chat Turn N - Program with reasoning:
{{your robot program, where each line includes a comment to explain your reasoning}}

After a turn, the user may provide corrective feedback. In that case, you should revise your program accordingly in the next turn.
At each turn, you should provide your entire program with reasoning, updated according to all the feedback you have received.

The scene has:
- The robot.
- The workspace is 24cm by 24cm, raised by 5cm above the floor, surrounded by sloping sides that extend to 0.5m from the center.
- Workspace coordinates are in meters. The center of the workspace is at (0.6, 0, 0.05); the far right corner is at (0.72, 0.12, 0.05); the near right corner is at
(0.72, -0.12, 0.05).
- The robot's hand is initialized 35cm above the workspace center, at (0.6, 0, 0.4)
- There are 4 objects:
1. A plug: a purple plastic connector which is 7cm long, and can be inserted into a socket.
2. A socket: A dark green plastic base to which the plug can be inserted.
3. A green cube with 5cm sides.
4. A red cube with 5cm sides.

## Robot Control API (Python):

### API Methods
def wait(duration_seconds: float):
  """Executes the current policy for the given number of seconds.

  This function should be used to separate different stages of the task.
  """

def set_object_target(
    obj: str,
    position: np.ndarray,
    xy_weight: float = 1.0,
    z_weight: float = 1.0,
):
  """
  Sets a target position for the object, with separate weights for XY distance and Z distance.
  Args:
    obj: The name of an object in the scene from ['plug', 'base', 'red' 'green']
    position: XYZ object position. Positive X is farther away from the scene, Negative X is closer. Positive Y is moving to the right, negative Y is left. Positive Z
is up. Do not set Z to below 0.05.
    xy_weight: If 1, the object will be moved to the target position in the XY plane. If 0, the object motion in the XY plane will be unregulated.
    z_weight: If 1, the object will be will be moved to the target position in Z. If 0, the object motion in Z will be unregulated.
  !!!DO NOT USE THIS FUNCTION TO MOVE ONE OBJECT TO ANOTHER!!!
  """

def limit_velocity(obj: str, weight: float = 1.0):
  """
  Encourages the robot to keep the object stationary and stop it from moving.
  Args:
    obj: The name of an object
    weight: If 1, will encourage the robot to keep the object stationary and motionless. If 0, the object velocity will be unregulated.
  """

def set_hand_position(position: np.ndarray, weight: float = 1.0):
  """Sets the target position for the hand
  Args:
    position: a 3D hand position.
    weight: If 1, the robot will be encouraged to move its hand to the target position. If 0, the hand motion will be unregulated.
  """

def reach(obj: str, weight: float = 1.0):
  """Move the robot hand to the given object.
  Args:
    obj: The name of an object in the scene from ['plug', 'base', 'red' 'green']
    weight: If 1, the robot will be encouraged to move its hand to the object and move the object around with the hand. If 0, the hand and the object will move
independently of each other, and the object will no longer be attached to the hand.
  """

def match_orientations(source: str, target: str, weight: float = 1.0):
  """Sets the importance of matching the orientations from object source to object target.
  Args:
    source: The name of an object in the scene from ['plug', 'base', 'red' 'green']
    target: The name of an object in the scene from ['plug', 'base', 'red' 'green']
    weight: If 1, the object orientations will be matched. If 0, their orientations will be unregulated.
  """

def minimize_distance_2d(source: str, target: str, weight: float = 1.0):
  """
  Brings the source object and target object together in the horizontal plane.
  Args:
    source: The name of an object in the scene from ['plug', 'base', 'red' 'green']
    target: The name of an object in the scene from ['plug', 'base', 'red' 'green']
    weight: If 1, the objects will be brought together. If 0, their distance will be unregulated. If -1, the robot will try to move the objects apart from each other.
  """

## Robot Code Writing Hints:
- For APIs that accept an obj string (like set_object_target, reach, limit_velocity), the only object names allowed are {"plug", "base", "red", "green"}.
- Use only numpy (imported as np) and the API provided above.
- Do not use any functions or object names besides the ones defined in the API above.
- Write the reasoning for each function call as a comment on its own line, immediately preceding the function call.
- Make your best guess for numerical values based on input from the user.
- The user may use synonyms to refer to the same objects: {gripper, hand}, {socket, base}, {plug, connector}.
- Remember: The coordinates of the objects are unknown so do not use absolute coordinate for an object's position.
- You can UNDO previous motions by setting the corresponding weights to 0.
- !!!IMPORTANT!!! When responding to user instructions, always rewrite the entire previous program with the new revisions.

## Chat Turn Example:
Chat Turn 1 - User:
Lift the green object up.
Chat Turn 1 - Program:
# Reasoning: First we reach for the green cube and wait for completion.
reach(obj='green', weight=1.0)
wait(10.0)
# Reasoning: Then, we lift the green cube by setting its Z target.
set_object_target('green', position=(0, 0, 0.2), xy_weight=0, z_weight=1)
# Reasoning: We want the green cube to move up, but we don't want it to move arbitrarily in the XY plane. A limit velocity with a small weight should be used.
limit_velocity('green', 0.5)
# Done.

## Chat Turn Example:
Chat Turn 1 - User:
Move the red thing next to the green thing
Chat Turn 1 - Program:
# Reasoning: First we reach for the red cube and wait for completion.
reach(obj='red', weight=1.0)
wait(10.0)
# Reasoning: To move the red object to the green object, since we did not lift the red object, we can just minimize the planar distance between the two objects.
minimize_distance_2d('red', 'green', weight=1)
# Done.

# New Chat Session
```

Fig. 20: Kuka+Hand prompt consists of a high level description of the goals and format (purple), robot reward code API (orange), code-writing hints (green), and chat turn examples (blue).