# Language to Rewards for Robotic Skill Synthesis

**Anonymous Author(s)**
Affiliation
Address
email

**Abstract:** Large language models (LLMs) have demonstrated exciting progress in acquiring diverse new capabilities through in-context learning, ranging from logical reasoning to code-writing. Robotics researchers have also explored using LLMs to advance the capabilities of robotic control. However, since low-level robot actions are hardware-dependent and underrepresented in LLM training corpora, existing efforts in applying LLMs to robotics have largely treated LLMs as semantic planners or relied on human-engineered control primitives to interface with the robot. On the other hand, reward functions are shown to be flexible representations that can be optimized for control policies to achieve diverse tasks, while their semantic richness makes them suitable to be specified by LLMs. In this work, we introduce a new paradigm that harnesses this realization by utilizing LLMs to define reward parameters that can be optimized and accomplish variety of robotic tasks. Using reward as the intermediate interface generated by LLMs, we can effectively bridge the gap between high-level language instructions or corrections to low-level robot actions. Meanwhile, combining this with a real-time optimizer, MuJoCo MPC, empowers an interactive behavior creation experience where users can immediately observe the results and provide feedback to the system. To systematically evaluate the performance of our proposed method, we designed a total of 17 tasks for a simulated quadruped robot and a dexterous manipulator robot. We demonstrate that our proposed method reliably tackles $90\%$ of the designed tasks, while a baseline using primitive skills as the interface with Code-as-policies achieves $50\%$ of the tasks. We further validated our method on a real robot arm where complex manipulation skills such as non-prehensile pushing emerge through our interactive system.

**Keywords:** Large language model (LLM), Low-level skill learning, Legged locomotion, Dexterous manipulation

## 1 Introduction

The recent advancements in large language models (LLMs) pretrained on extensive internet data [1, 2] has revolutionized the ability to interpret and act on user inputs in natural language. These LLMs exhibit remarkable adaptability to new contexts (such as APIs [3], task descriptions [4], or textual feedback [5]), allowing for tasks ranging from logical reasoning [6, 7] to code generation [8] with minimal hand-crafted examples.

These diverse applications have extended to the field of robotics as well, where substantial progress has been made in using LLMs to drive robot behaviors [3, 5, 4, 9, 10, 11]: from step-by-step planning [4, 9, 12], goal-oriented dialogue [10, 11], to robot-code-writing agents [3, 13]. While these methods impart new modes of compositional generalization, they focus on using language to concatenate together new behaviors from an existing library of control primitives that are either manually-engineered or learned a priori. Despite having internal knowledge about robot motions, LLMs struggle with directly outputting low-level robot commands due to the limited availability of relevant training data (Fig. 1). As a result, the expression of these methods are bottlenecked by the breadth of the available primitives, the design of which often requires extensive expert knowledge or massive data collection [14, 15, 16].

To tackle these challenges, we need to operate at a level of abstraction that allows harnessing the intuitive and interactive capabilities offered by LLMs. Our key insight is to leverage reward functions as an interface that bridges the gap between language and low-level robot actions. This is motivated by the fact that language instructions from humans often tend to describe behavioral outcomes instead of low-level behavioral details

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

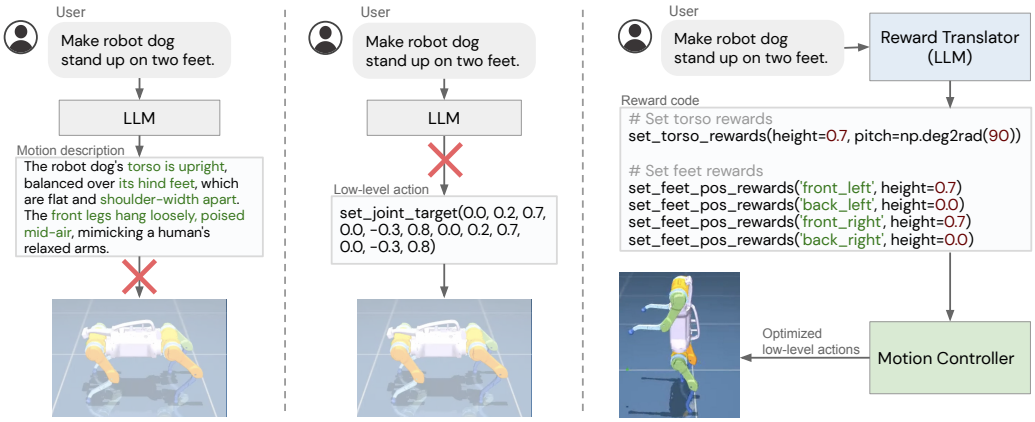



Figure 1: LLMs have some internal knowledge about robot motions, but cannot directly translate them into actions (left). Low-level action code can be executed on robots, but LLMs know little about them (mid). We attempt to bridge this gap, by proposing a system (right) consisting of the Reward Translatorthat interprets the user input and transform it into a reward specification. The reward specification is then consumed by a Motion Controller that interactively synthesizes a robot motion which optimizes the given reward.

(e.g. "robot standing up" versus "applying 15 Nm to hip motor"), and therefore we posit that it would be easier to connect instructions to rewards than low-level actions given the richness of semantics in rewards. In addition, reward terms are usually modular and compositional, which enables concise representations of complex behaviors, goals, and constraints. This modularity further creates an opportunity for the user to interactively steer the robot behavior. However, in many previous works in reinforcement learning (RL) or model predictive control (MPC), manual reward design requires extensive domain expertise [17, 18, 19]. While reward design can be automated, these techniques are sample-inefficient and still requires manual specification of an objective indicator function for each task [20]. This points to a missing link between the reward structures and task specification which is often in natural language. As such, we propose to utilize LLMs to automatically generate rewards, and leverage online optimization techniques to solve them. Concretely, we explore the code-writing capabilities of LLMs to translate task semantics to reward functions, and use MuJoCo MPC, a real-time optimization tool to synthesize robot behavior in real-time [21]. Thus reward functions generated by LLMs can enable non-technical users to generate and steer novel and intricate robot behaviors without the need for vast amounts of data nor the expertise to engineer low-level primitives.

The idea of grounding language to reward has been explored by prior work for extracting user preferences and task knowledge [22, 23, 24, 25, 26, 27]. Despite promising results, they usually require training data to learn the mapping between language to reward. With our proposed method, we enable a data efficient interactive system where the human engages in a dialogue with the LLM to guide the generation of rewards and, consequently, robot behaviors (Fig. 1).

Across a span of 17 control problems on a simulated quadruped and a dexterous manipulator robot, we show that this formulation delivers diverse and challenging locomotion and manipulation skills. Examples include getting a quadruped robot to stand up, asking it to do a moonwalk, or tasking a manipulator with dexterous hand to open a faucet. We perform a large-scale evaluation to measure the overall performance of our proposed method. We compare our method to a baseline that uses a fixed set of primitive skills and an alternative formulation of grounding language to reward. We show that our proposed formulation can solve $40\%$ more skills than baselines and is more stable in solving individual skills. We further deploy our approach to a real robot manipulator and demonstrate complex manipulation skills through language instructions.

## 2 Related Work

Here we discuss relevant prior work that reason about language instructions to generate robot actions, code, or rewards to ground natural language to low-level robotic control. We then discuss work focused on responding to interactive human feedback such as language corrections.

**Language to Actions.** Directly predicting low-level control actions based on a language instruction has been studied using various robot learning frameworks. Early work in the language community studied mapping templated language to controllers with temporal logic [28] or learning a parser to motion primitives [29], while more recent work utilize end-to-end models that produce actions conditioned on natural

language descriptions. One example is instruction following methods in navigation [30]. However, they often assume low-dimensional actions navigating from one node of the graph to another [30, 31]. To extend the end-to-end approaches to manipulation, a common approach is to utilize latent embeddings of language commands as multitask input context, and train with behavioral cloning [14, 32, 16], offline reinforcement learning [33], goal-conditioned reinforcement learning [34], or in a shared autonomy paradigm [35]. While end-to-end trained policies can be performant, they require significant amount of data in the form of offline datasets or online environment interaction. In contrast, we study a less data hungry approach where low-level actions are not directly produced by an end-to-end policy but instead by an optimal controller.

**Language to Code.** Code generation models have been widely studied both in and outside robotics context [36, 8, 37]. The capability of those models range from solving coding competition questions [38] and benchmarks [39], to drawing simple figures [40], generating policies that solve 2D tasks [41], and complex instruction following tasks [3]. In this work, we study LLMs for generating code for reward functions, and show that the expression of the rewards can lead to expressive low-level policies.

**Language to Rewards.** The idea of translating natural language instructions to rewards has been explored by several prior work [26, 23, 25, 42, 22, 43, 27]. A common strategy in this direction is to train domain-specific reward models that map language instructions to reward values [23, 22, 42] or constraints [25]. Although these methods can achieve challenging language conditioned robotic tasks such as object pushing [25], and drawer opening [42], they require considerable language-labeled data to train the reward model. Recent works investigated using LLMs directly as a reward function for inferring user intentions in negotiation games or collaborative human-AI interaction games [26, 27]. By leveraging LLMs to assign reward values during RL training, they demonstrate training agents that are aligned with user intentions and preferences without explicit reward modeling. However, these works receive reward values of rollouts when training RL policies, which requires a large number of queries to LLMs during training. In contrast, we levrage LLMs to produce a parameterized reward function that can then be optimized. A similar direction to this work is automated parameterization of reward functions, which had been explored in AutoRL [20], however, they don't provide a language interface.

**Incorporating Iterative Human Feedback.** Correcting plans with iterative language feedback has also been explored in the past. Broad et al. enable efficient online corrections using distributed correspondence graphs to ground language [44]. However, this work relies on a semantic parser with pre-defined mappings to ground language corrections. More end-to-end approaches have also demonstrated learning a language correction conditioned policy, but they are similarly data hungry and thus fall back to shared autonomy to reduce complexity [45]. Later work explore mapping language corrections to composable cost functions similar to our work by training a prediction model from demonstration and apply trajectory optimization to perform control [25]. Followup works further simplifies the system by integrating language corrections to directly modify the waypoints of a trajectory using extensive datasets of paired corrections and demonstrations [46, 47]. In contrast to these prior work, we demonstrate a flexible and data-efficient approach that leverages LLMs to allow for multi-step correction of reward functions based on human feedback.

# 3 Grounding Language to Actions Using Rewards

## 3.1 Background and Reward Interface

Our system takes user instruction in natural language and synthesizes corresponding robot motions by leveraging reward function as the interface to communicate with low-level controllers. We define the reward function in the context of Markov Decision Process (MDP), commonly used to formulate robot control problems: $(S,A,R,P,p_0)$, where $S$ is the state space, $A$ is the action space, $R: S\times A\mapsto\mathbb{R}$ is the reward function, $P: S\times A\mapsto S$ is the dynamics equation, and $p_0$ is the initial state distribution. Given a reward function $R$, an optimal controller finds a sequence of actions $\mathbf{a}_{1:H}=\{\mathbf{a}_1,...,\mathbf{a}_H\}$ that maximizes the expected accumulated reward: $J(\mathbf{a}_{1:H})=\mathbb{E}_{\tau=(\mathbf{s}_0,\mathbf{a}_0,...,\mathbf{s}_H)}\sum_{t=0}^{H}R(\mathbf{s}_t,\mathbf{a}_t)$, where $H$ is the rollout horizon.

In this work, we assume the reward takes a particular form, suitable for use with MJPC (see below). The reward is the sum of a set of individual terms:

$$R(\mathbf{s},\mathbf{a})=-\sum_{i=0}^{M}w_i\cdot\mathrm{n}_i\big(r_i(\mathbf{s},\mathbf{a},\psi_i)\big), \tag{1}$$

where $w\in\mathbf{R}_+$ is a non-negative weight, $\mathrm{n}(\cdot):\mathbf{R}\to\mathbf{R}_+$ is a twice-differentiable norm that takes its minimum at 0, $r\in\mathbf{R}$ is a residual term that achieves optimality when $r=0$, and $\psi_i$ is the parameters of

the $i_{th}$ residual term. For example, if we want to have the robot raise its body height $h$ to a desired height, we may design a residual term $r_h(h,\psi) = h - \psi$, where the reward parameter $\psi$ denotes the desired height, and use the l2 norm to construct the final reward function: $R_h = -w||r_h||_2$. In principle, one may design task-specific residual terms that can solve particular controller tasks. However, designing these residuals requires domain expertise and may not generalize to novel tasks. In this work, we use a set of generic and simple residual terms, and leverage the power of LLMs to compose different terms to generate complex behaviors. The full set of residual terms used in this work can be found in the Appendix A.5.

Our proposed system consists of two key components (Fig. 1 right): i) a *Reward Translator*, built upon pre-trained Large Language Models (LLMs) [10], that interacts with and understands user intents and modulates all reward parameters $\psi$ and weights $w$, and ii) a *Motion Controller*, based on MuJoCo MPC [21], that takes the generated reward and interactively optimize the optimal action sequence $\mathbf{a}_{1:H}$. Below we provide more details on the design of Reward Translatorand Motion Controller.

### 3.2 Reward Translator

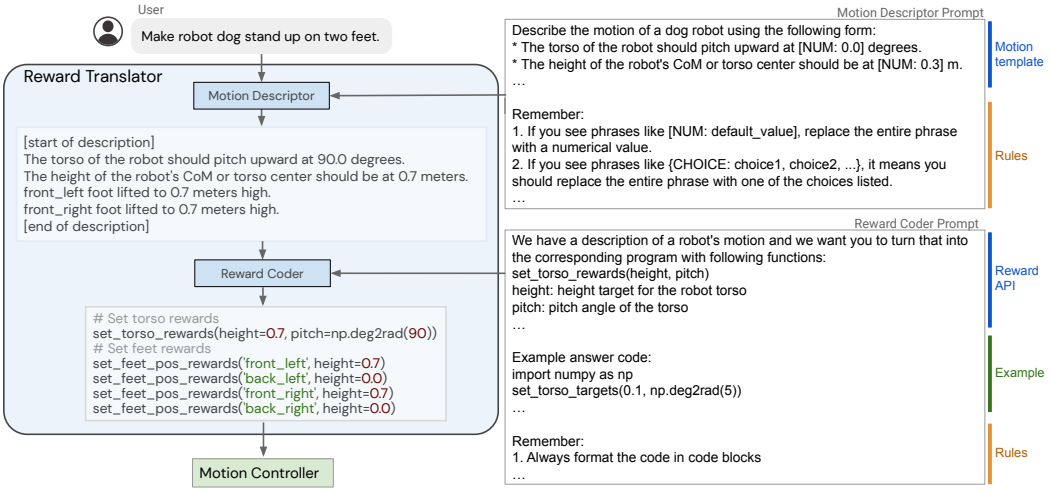



Figure 2: Detailed dataflow of the Reward Translator. A *Motion Descriptor* LLM takes the user input and describe the user-specified motion in natural language, and a *Reward Coder* translates the motion into the reward parameters.

Inspired by recent progress on Large Language Models (LLMs), we propose to build the Reward Translator-based on LLMs to map user interactions to reward functions corresponding to the desired robot motion. As reward tuning is highly domain-specific and requires expert knowledge, it is unsurprising that LLMs trained on generic language datasets (e.g. [1]) cannot directly generate a reward for a specific hardware. Instead, we explore the in-context learning ability of LLMs to achieve this goal, inspired by prior work that demonstrated a variety of in-context learning skills for LLMs [2, 48]. Furthermore, we decompose the problem of language to reward into two stages: motion description and reward coding task, as illustrated in Fig. 2.

**Motion Description** In the first stage, we design a *Motion Descriptor* LLM that interprets and expands the user input into a natural language description of the desired robot motion following a pre-defined template (see example in Fig. 2). Although it is possible for LLMs to directly generate reasonable reward functions for relatively simple task, it often fails for tasks that necessitates complex reasoning. On the other hand, as observed in Fig. 1 left, LLMs can describe complex motions in detailed natural language successfully.

Inspired by this observation, we design a template that describes common movements of a robot (see Fig. 2 top right for an example of the template and the prompt for the LLM) to effectively harness LLMs' internal knowledge about motions. The role of the *Motion Descriptor* is to complete the provided template (e.g., replacing certain elements such as `CHOICE` and `NUM` in the example. This helps the *Motion Descriptor* produce more structured and predictable outputs and improves stability of the overall system. In addition, as we are describing the motion in natural language, we do not need to provide any specific examples in the prompt and can rely entirely on LLMs to generate the result.

**Reward Coding** In the second stage, we translate the generated motion description into the reward function using a second LLM. We formulate the problem of language to reward function as a code-writing

task to benefit from the LLMs' knowledge of coding and code structure, thus we name the second LLM the *Reward Coder*. We design a prompt for instructing the LLM to generate reward specifying code (see example in Fig. 2 bottom right). The prompt consists of three parts: i) description of the reward APIs that the LLM can call to specify different parameters of the reward function, ii) an example response that we expect the *Reward Coder* to produce, and iii) the constraints and rules that the *Reward Coder* needs to follow. Note that the example is to demonstrate to the LLM how the response should look like, instead of teaching it how to perform a specific task. As such, the *Reward Coder* needs to specify the reward parameters based on its own knowledge about the motion from the natural language description.

### 3.3 Motion Controller

The Motion Controller needs to map the reward function generated by the Reward Translatorto low-level robot actions $\mathbf{a}_{1:H}$ that maximize the accumulated reward $J(\mathbf{a}_{1:H})$. There are a few possible ways to achieve this, including using reinforcement learning (RL), offline trajectory optimization, or, as in this work, receding horizon trajectory optimization, i.e., model predictive control (MPC). At each control step, MPC plans a sequence of optimized actions $\mathbf{a}_{1:H}$ and sends to the robot. The robot applies the action corresponding to its current timestamp, advances to the next step, and sends the updated robot states to the MJPC planner to initiate the next planning cycle. The frequent re-planning in MPC empowers its robustness to uncertainties in the system and, importantly, enables interactive motion synthesis and correction. Specifically, we use an open-source implementation based on the MuJoCo simulator [49], MJPC [21]. MJPC has demonstrated the interactive creation of diverse behaviors such as legged locomotion, grasping, and finger-gaiting while supporting multiple planning algorithms, such as iLQG and Predictive Sampling. Following the observation by Howell et al [21], second-order planners such as iLQG produces smoother and more accurate actions while zeroth-order planners such as Predictive Sampling is better at exploring non-smooth optimization landscape, we use iLQG for legged locomotion tasks, while use Predictive Sampling for manipulation tasks in this work.

## 4 Experiments

We design experiments to answer the following questions:

1) Is our proposed method, by combining LLMs and MJPC, able to generate diverse and complex robot motions through natural language interface?

2) Does interfacing with the reward function result in a more expressive pipeline than interfacing directly with low-level or primitive actions and is Motion Descriptor necessary for achieving reliable performance?

3) Can our method be applied to real robot hardware?

### 4.1 Experiment Setup

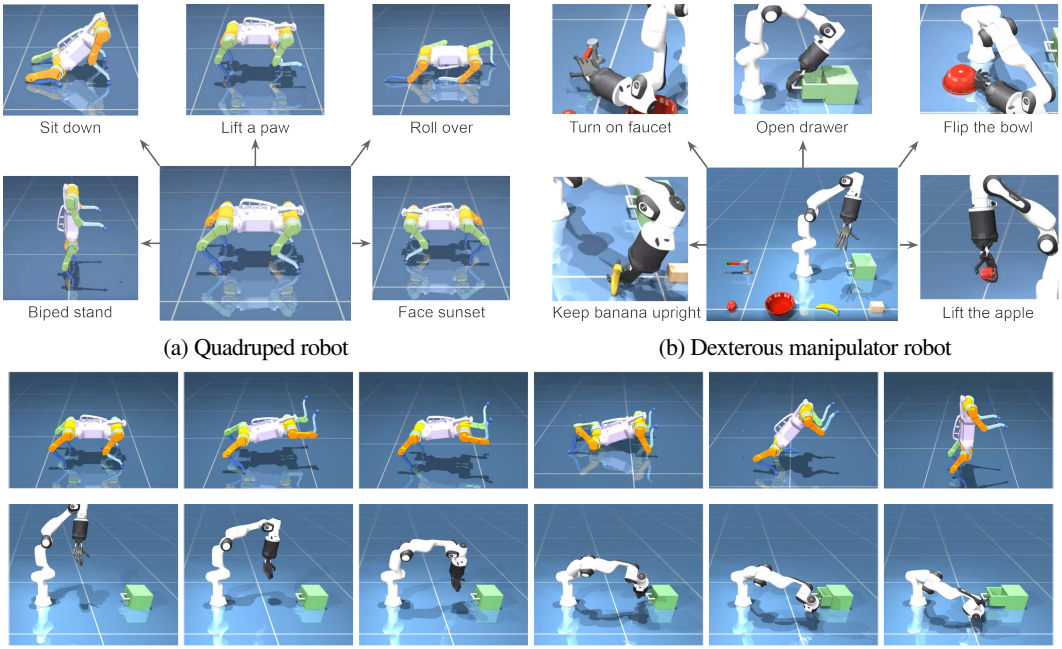
We evaluate our approach on two simulated robotic systems: a quadruped robot, and a dexterous robot manipulator (Fig. 3). Both robots are modeled and simulated in MuJoCo MPC [21]. In all experiments we use GPT-4 as the underlying LLM module [50]. Here we describe the key setups of each robot. More details regarding the full prompts and reward function can be found in Appendix A.4 and A.5.

**Quadruped Robot** In this example, we demonstrate using our system to command a four legged robot (Fig. 3 (a)) to perform a variety of motor skills. The quadruped robot has 12 joints, 3 on each leg. Quadruped robots have been demonstrated to perform a large variety of skills including locomotion [40], hopping [18], biped standing [51, 52], etc. We apply our system to the quadruped robot to perform a similar suite of skills while only using natural language as input.

**Dexterous Manipulator** In the second example, we demonstrate our system on a dexterous manipulator robot. The robot consists of a 7 DoF Franka Emika arm and a 20 DoF shadow hand as the end-effector (as shown in Fig. 3 (b)). This creates a large action space, making it challenging to manually train a controller to directly perform tasks using this robot.

### 4.2 Baselines

We compare our proposed system to two baseline methods: i) an ablation of our approach that only uses *Reward Coder* without having access to the *Motion Descriptor*, and ii) Code-as-Policies [3] where the




(a) Quadruped robot

(b) Dexterous manipulator robot

(c) Example rollout for the two robots.

Figure 3: The two robots used in our experiments and sampled tasks. (a) a Quadruped robot with 12 DoFs. (b) a dexterous manipulator robot with 27 DoFs. (c) example rollouts produced by our algorithm.

LLM generates a plan for the robot motion using a set of pre-defined robot primitive skills instead of reward functions. For the Code-as-Policies (CaP) baseline, we design the primitive skills based on common commands available to the robot. Due to limited space we put the full list of primitives in Appendix A.2.

### 4.3 Tasks

We design nine tasks for the quadruped robot and eight tasks for the dexterous manipulator to evaluate the performance of our system. Fig. 3 shows samples of the tasks. The full list of tasks can be found in Appendix A.1. Videos of sampled tasks can also be found in supplementary video and project website [1].

For the quadruped robot, the tasks can be categorized into four types: *1) Heading direction control*, where the system needs to interpret indirect instructions about the robot's heading direction and to control the robot to face the right direction (e.g., identify the direction of sunrise or sunset). *2) Body pose control*, where we evaluate the ability of the system to understand and process commands to have the robot reach different body poses, inspired by common commands issued by human to dogs such as sit and roll over. *3) Limb control*, where we task the system to lift particular foot of the robot. Furthermore, we also test the ability of the system to take additional instructions to modify an existing skill, such turn in place with lifted feet. *4) Locomotion styles*, where we evaluate our proposed system in generating different locomotion styles. In particular, we design a challenging task of having the quadruped stand up on two back feet to walk in a bipedal mode.

For the dexterous manipulator, we design tasks to test its ability to achieve different types of interactions with objects such as lifting, moving, and re-orienting. We test the system on a diverse set of objects with significantly different shapes and sizes (Fig. 3) for each task. We further include two tasks that involves interacting with articulated objects of different joint types.

### 4.4 Evaluation results

For each task and method considered, we generate 10 responses from Reward Translator, each evaluated in MJPC for 50 times, thus we measure the end-to-end stability of the full pipeline. Fig. 4 shows the results for both robots. Our proposed approach achieves significantly higher success rate for $11/17$ task categories and

[1] `language-to-reward-review.github.io`

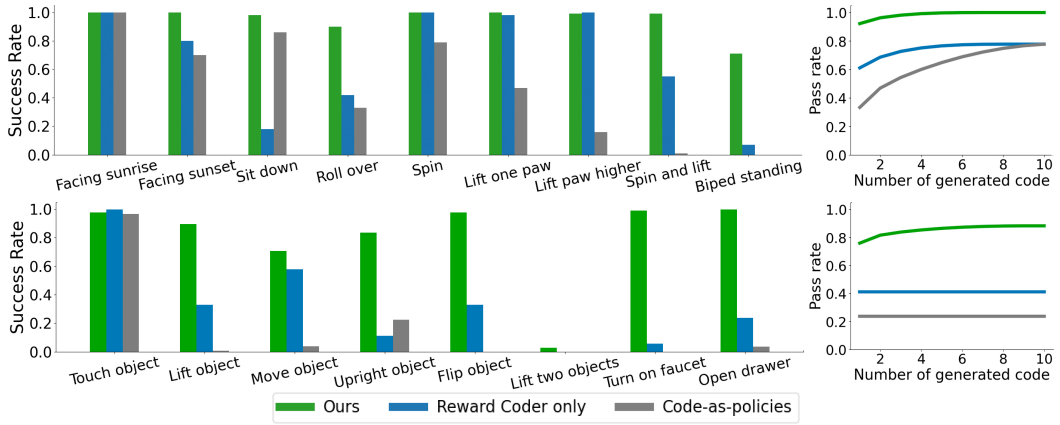



Figure 4: Comparison of our method and alternative methods in terms of pass rate: if we generate N pieces of code for each task and pick the best performing one, what's the percentage of tasks that the system can successfully tackle.

comparable performance for the rest tasks, showing the effectiveness and reliability of the proposed method. When compared to the CaP baseline, our method achieves better success rate in almost all tasks. This is due to that CaP can perform well on tasks that can be expressed by the given primitives (e.g. Touch object) or very close to the given examples in prompt (e.g. Sit down), but fails to generalize to novel low-level skills. On the other hand, using Reward Coder only can achieve success on some tasks but fails in ones that requires more reasoning. For example, when asked to open a drawer, the Reward Coder only baseline often forget to task the robot hand to get closer to the drawer handle and only design the reward for encouraging the drawer to be open. Sampled responses from different method can be found in Appendix A.7.

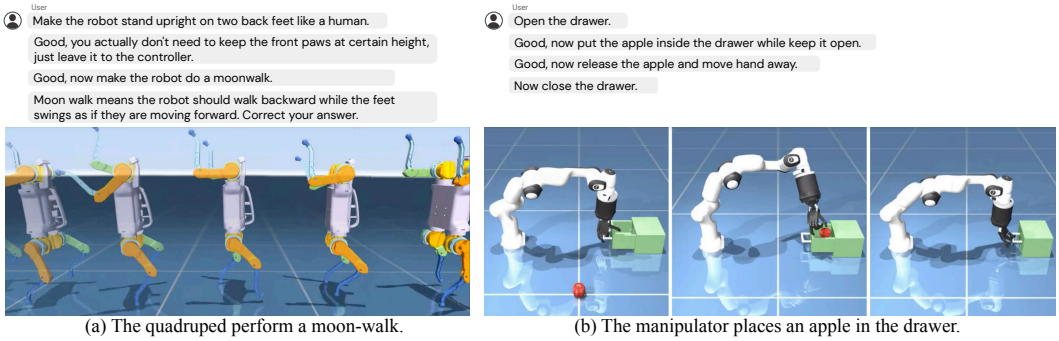
To further understand the overall performance of different systems, we also show the pass rate in Fig. 4 right, which is a standard metric for analyzing code generation performance [8]. For each point in the plot, it represents the percentage of tasks the system can solve, given that it can generate N pieces of code for each task and pick the best performing one. As such, the pass rate curve measures the stability of the system (the more flat it is, the more stable the system is) as well as the task coverage of the system (the converged point represents how many tasks the system can solve given sufficient number of trials). It is clear from the result that for both embodiments, using reward as the interface empowers LLMs to solve more tasks more reliably, and the use of Structured Motion Description further boosts the system performance significantly.

### 4.5 Interactive Motion Synthesis Results

One benefit of using a real time optimization tool like MJPC is that humans can observe the motion being synthesized in real time and provide feedback. We showcase two examples where we teach the robot to perform complex tasks through multiple rounds of interactions. In the first example, we task the quadruped robot to stand up and perform a moon-walk skill (Fig. 5a). We give four instructions to achieve the task, as shown in Fig. 5. Each instruction improves the behavior towards the desired behavior based on the interactively synthesized results. This showcase that users can interactively shape the behavior of the robot in natural language. In the second example, we showcase a different way of leveraging the interactivity of our system by sequentially commanding the dexterous manipulator robot to place an apple in a drawer, as seen in Fig. 5b. Results of the interactive results are best viewed in the supplementary video and full code output from our method can be found in Appendix A.8.

### 4.6 Real-robot experiments

We implement a version of our method onto a mobile manipulator, and tested it on nonprehensile manipulation tasks in the real world. In simulation, we have access to the ground-truth state for objects in the scene. In the real world, we detect objects in image-space using an open-vocabulary detector: F-VLM [53]. We extract the associated points from point cloud behind the mask and perform outlier rejection for points that might belong to the background. From a birds-eye view, we fit a minimum volume rectangle and take the extremes to determine the extent in the z-axis. We use this 3D bounding box as state estimation for corresponding object in simulation. To detect the surface of the table with proper orientation, we use an AprilTag [54]. In addition, as seen in the supplementary video, MJPC can discover




(a) The quadruped perform a moon-walk. (b) The manipulator places an apple in the drawer.

Figure 5: The two interactive examples using our proposed system.

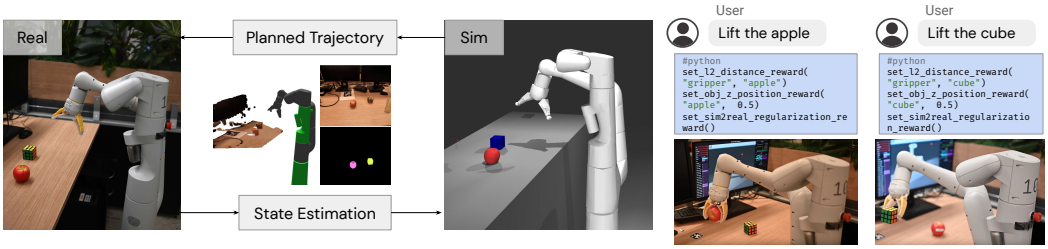



Figure 6: Implementation and rollouts of the proposed system in the real world.

highly dexterous and dyanmic maneuvers to accomplish the desired task. However, these movements are beyond the capabilities of current real hardwares. To mitigate this issue, we design a regularization residual term specific to encourage steady and stable robot movements when applying our system to the real robot (*set_sim2real_regularization_reward()* in Fig. 6, see Appendix A.5.3 for details for this term).

We demonstrate sim-to-real transfer on two tasks: object pushing and object grasping. Our system is able to generate relevant reward code and the Mujoco MPC is able to synthesize the pushing and grasping motion. For rollouts please refer to the supplementary video/website and Fig. 7.

## 5 Discussion and Conlusion

In this work, we investigate a new paradigm for interfacing an LLM with a robot through reward functions, powered by a low-level model predictive control tool, MuJoCo MPC. Using reward function as the interface enables LLMs to work in a semantic-rich space that play to the strengths of LLMs, while ensures the expressiveness of the resulting controller. To further improve the performance of the system, we propose to use a structured motion description template to better extract internal knowledge about robot motions from LLMs. We evaluate our proposed system on two simulated robotic platforms: a quadruped robot and a dexterous manipulator robot. We apply our approach to both robots to acquire a wide variety of skills. Compared to alternative methods that do not use reward as the interface, or do not use the structured motion description, our method achieves significantly better performance in terms of stability and the number of tasks it can solve.

**Limitations and Future Work.** Though we show that our system can obtain a diverse set of skills through natural language interactions, there are a few limitations. First, we currently design templates of structured motion descriptions for each type of robot morphology, which requires manual work. An interesting future direction is to unify or automate the template design to make the system easily extendable to novel robot morphologies. Second, our method currently relies on language as the interaction interface with human users. As such, it can be challenging to design tasks that are not easily described in language (e.g., “walk gracefully"). One potential way to mitigate this issue is to extend the system to multi-modal inputs to allow richer forms of user interactions (e.g., by showing a video of the desirable behavior). Thirdly, we currently use pre-defined reward terms whose weights and parameters are modulated by the LLMs. Constraining the reward design space helps improve stability of the system while sacrifices some flexibility. For example, our current design does not support time-varying rewards and would require re-designing the prompt to support that. Enabling LLMs to reliably design reward functions from scratch is thus an important and fruitful research direction.

## References


[1] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

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

[5] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

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

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

[8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

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

[11] C. Snell, S. Yang, J. Fu, Y. Su, and S. Levine. Context-aware language modeling for goal-oriented dialogue systems. *arXiv preprint arXiv:2204.10198*, 2022.

[12] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[13] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[14] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=8kbp23tSGYv.

[15] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[16] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence. Interactive language: Talking to robots in real time. *arXiv preprint arXiv:2210.06407*, 2022.

[17] J. Lee, J. Hwangbo, and M. Hutter. Robust recovery controller for a quadrupedal robot using deep reinforcement learning. *arXiv preprint arXiv:1901.07517*, 2019.

[18] J. Siekmann, Y. Godse, A. Fern, and J. Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315. IEEE, 2021.

[19] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. *arXiv preprint arXiv:2008.07792*, 2020.

[20] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[21] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa. Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo. dec 2022. doi:10.48550/arXiv.2212.00541. URL https://arxiv.org/abs/2212.00541.

[22] P. Goyal, S. Niekum, and R. J. Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

[23] J. Lin, D. Fried, D. Klein, and A. Dragan. Inferring rewards from language in context. *arXiv preprint arXiv:2204.02515*, 2022.

[24] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=rc8o_j8I8PX.

[25] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox. Correcting robot plans with natural language feedback. In *Robotics: Science and Systems (RSS)*, 2022.

[26] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh. Reward design with language models. In *International Conference on Learning Representations (ICLR)*, 2023.

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

[28] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[29] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *Experimental robotics: the 13th international symposium on experimental robotics*, pages 403–415. Springer, 2013.

[30] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. *arXiv preprint arXiv:2010.07954*, 2020.

[31] A. Kamath, P. Anderson, S. Wang, J. Y. Koh, A. Ku, A. Waters, Y. Yang, J. Baldridge, and Z. Parekh. A new path: Scaling vision-and-language navigation with synthetic instructions and imitation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10813–10823, 2023.

[32] O. Mees, J. Borja-Diaz, and W. Burgard. Grounding language with visual affordances over unstructured data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023.

[33] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021.

[34] J. Fu, A. Korattikara, S. Levine, and S. Guadarrama. From language to goals: Inverse reinforcement learning for vision-based instruction following. *arXiv preprint arXiv:1902.07742*, 2019.

[35] S. Karamcheti, M. Srivastava, P. Liang, and D. Sadigh. Lila: Language-informed latent actions. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[36] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv:2108.07732*, 2021.

[37] K. Ellis, C. Wong, M. Nye, M. Sable-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv:2006.08381*, 2020.

[38] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624): 1092–1097, 2022.

[39] F. Alet, J. Lopez-Contreras, J. Koppel, M. Nye, A. Solar-Lezama, T. Lozano-Perez, L. Kaelbling, and J. Tenenbaum. A large-scale benchmark for few-shot program induction and synthesis. In *ICML*, 2021.

[40] L. Tian, K. Ellis, M. Kryven, and J. Tenenbaum. Learning abstract structure for drawing by efficient motor program induction. *NeurIPS*, 2020.

[41] D. Trivedi, J. Zhang, S.-H. Sun, and J. J. Lim. Learning to synthesize programs as interpretable and generalizable policies. *NeurIPS*, 2021.

[42] S. Nair, E. Mitchell, K. Chen, b. ichter, S. Savarese, and C. Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1303–1315. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/nair22a.html.

[43] D. Bahdanau, F. Hill, J. Leike, E. Hughes, A. Hosseini, P. Kohli, and E. Grefenstette. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*, 2018.

[44] A. Broad, J. Arkin, N. D. Ratliff, T. M. Howard, and B. Argall. Real-time natural language corrections for assistive robotic manipulators. *International Journal of Robotics Research (IJRR)*, 36:684–698, 2017.

[45] Y. Cui, S. Karamcheti, R. Palleti, N. Shivakumar, P. Liang, and D. Sadigh. "no, to the right"–online language corrections for robotic manipulation via shared autonomy. *arXiv preprint arXiv:2301.02555*, 2023.

[46] A. F. C. Bucker, L. F. C. Figueredo, S. Haddadin, A. Kapoor, S. Ma, and R. Bonatti. Reshaping robot trajectories using natural language commands: A study of multi-modal data alignment using transformers. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 978–984, 2022.

[47] A. F. C. Bucker, L. F. C. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti. Latte: Language trajectory transformer. *arXiv preprint arXiv:2208.02918*, 2022.

[48] N. Ziems, W. Yu, Z. Zhang, and M. Jiang. Large language models are built-in autoregressive search engines. *arXiv preprint arXiv:2305.09612*, 2023.

[49] E. Todorov, T. Erez, and Y. 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.

[50] OpenAI. Gpt-4 technical report. *arXiv*, 2023.

[51] L. Smith, J. C. Kew, T. Li, L. Luu, X. B. Peng, S. Ha, J. Tan, and S. Levine. Learning and adapting agile locomotion skills by transferring experience. *arXiv preprint arXiv:2304.09834*, 2023.

[52] Y. Fuchioka, Z. Xie, and M. van de Panne. Opt-mimic: Imitation of optimized trajectories for dynamic quadruped behaviors. *arXiv preprint arXiv:2210.01247*, 2022.

[53] W. Kuo, Y. Cui, X. Gu, A. Piergiovanni, and A. Angelova. F-vlm: Open-vocabulary object detection upon frozen vision and language models. *arXiv preprint arXiv:2209.15639*, 2023.

[54] E. Olson. Apriltag: A robust and flexible visual fiducial system. 2023.