Long-horizon Locomotion and Manipulation on a Quadrupedal Robot with Large Language Models

Yutao Ouyang[1*], Jinhan Li[2*], Yunfei Li[2], Zhongyu Li[3], Chao Yu[2], Koushil Sreenath[3], Yi Wu[1,2]

Abstract— We present a large language model (LLM) based system to empower quadrupedal robots with problem-solving abilities for long-horizon tasks beyond short-term motions. Long-horizon tasks for quadrupeds are challenging since they require both a high-level understanding of the semantics of the problem for task planning and a broad range of locomotion and manipulation skills to interact with the environment. Our system builds a high-level reasoning layer with large language models, which generates hybrid discrete-continuous plans as robot code from task descriptions. It comprises multiple LLM agents: a semantic planner for sketching a plan, a parameter calculator for predicting arguments in the plan, and a code generator to convert the plan into executable robot code. At the low level, we adopt reinforcement learning to train a set of motion planning and control skills to unleash the flexibility of quadrupeds for rich environment interactions. Our system is tested on long-horizon tasks that are infeasible to complete with one single skill. Simulation and real-world experiments show that it successfully figures out multi-step strategies and demonstrates non-trivial behaviors, including building tools or notifying a human for help.

I. INTRODUCTION

Quadrupedal robots have demonstrated advantages in mobility and versatility using a variety of skills. The field has witnessed exciting advancement in developing locomotion controllers to enhance traversability [1], [2], [3] and manipulation controllers to facilitate physical interactions with the world [4], [5], [6]. Although much progress has been made in the acquisition of specific motion skills, we would like to push the autonomy of the robot to a higher level beyond the individual skills and ask: How can a quadruped robot combine its locomotion and manipulation abilities to tackle challenging problems necessitating long-horizon planning and strategic interactions with the environment?

An example of a long-horizon task may be turning the lights off before exiting an office, a scenario demanding a strategic combination of actions such as climbing to reach a button, pressing an object, and walking. Such a long-term maneuver poses significant challenges both in high-level reasoning and in low-level motion planning and control.

For high-level reasoning, the agent must devise a multi-step strategy that takes into account both physical affordances and skill limitations, all without the aid of immediate learning signals, thus encountering substantial exploration hurdles in long-term problem-solving. Furthermore, the vast decision space of the high-level policy exacerbates the challenges.

*Equal contribution
[1]Shanghai Qi Zhi Institute, Shanghai, China
[2]Tsinghua University, Beijing, China
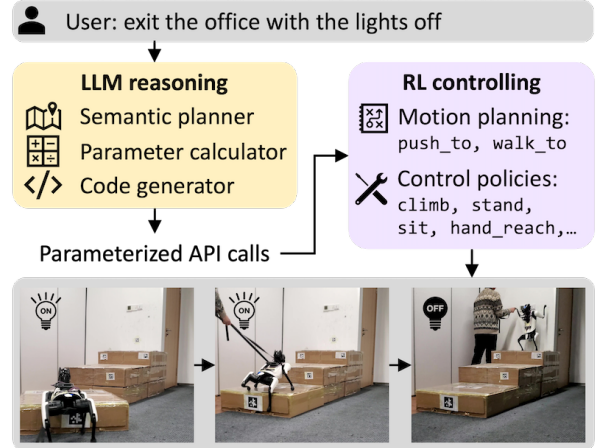[3]University of California, Berkeley, CA, USA

Fig. 1: Overview of the hierarchical system for long-horizon loco-manipulation task. The system is built up from a reasoning layer for task decomposition (yellow) and a controlling layer for skill execution (purple). Given the language description of the long-horizon task (top), a cascade of LLM agents perform high-level task planning and generate function calls of parameterized robot skills. The controlling layer instantiates the mid-level motion planning and low-level controlling skills with RL.

The policy needs to simultaneously determine which skill to execute (e.g., walking or pitching up) and how to execute it (e.g., specifying continuous parameters for the target position). Prior research often relies on demonstrations [7] or heuristics in search [8] to reduce the computation burden. As for low-level control, the quadrupeds' nonlinearity and high dimensionality pose challenges in acquiring versatile behaviors necessary to enable complex environmental interactions.

We propose a hierarchical system to address the challenges in long-horizon loco-manipulation problems for quadrupeds as illustrated in Fig. 1. At the high level, we harness the planning ability of pretrained large language models (LLMs) for abstract reasoning over the long horizon. Leveraging the wealth of encoded common-sense knowledge, the LLM-based reasoning layer can directly predict high-level strategies without demonstrations or searching heuristics. To better ground to robotic problems, we design the reasoning module as a cascade of multiple LLM agents. A semantic planner sketches a discrete plan, while a parameter calculator predicts continuous parameters in the plan. Subsequently, a code generator encapsulates the plan into function calls for robot skills. At the low level, we train a spectrum of motion

planning and control policies with model-free reinforcement learning, spanning from quadrupedal locomotion to bipedal manipulation.

Our system is tested with long-horizon loco-manipulation tasks such as turning off the lights before exiting the office, and delivering a package into a room with a closed door. The LLM-based reasoning module figures out strategic solutions including building stairs with appropriate heights to touch an originally unreachable button. Combined with versatile low-level motion planning and control policies, the system can address long-horizon tasks that require minutes of execution of multiple motion skills, demonstrating a success rate of over 70% in simulation and successful deployment in the real world.

II. RELATED WORK

Quadrupedal locomotion and manipulation: Developing versatile motions over quadrupedal robots to make them work like real animals has attracted much research interest. A large body of works focuses on robust and agile locomotion skills that can walk on challenging terrains [9], [3], [10] and perform extreme parkour [11], [12]. Some recent works develop manipulation skills for quadrupeds to better interact with the environment, such as learning manipulation via locomotion to push heavy objects [13], manipulating with one or two legs while balancing on other legs [14], [7], [6], or manipulating with a gripper mounted on the robot [15], [16]. There are also works that learn mid-level motion planning policies over these low-level motion controllers for more complex skills, such as soccer manipulation [17], [18]. We focus on empowering quadrupeds with long-horizon problem-solving abilities beyond motion planning and control skills for specific tasks. The problems considered in this work require an abstract task-planning ability that reasons over the semantics. The desired strategy, such as building stairs under the wall so that the robot can climb up and reach the light switch, requires common-sense knowledge about the physical constraints of the environment and the robot's capabilities and is challenging to discover from scratch.

Tackling long-horizon challenge in robotic tasks: Traditionally, Task and Motion Planning (TAMP) [19] methods are adopted to get plans for long-horizon tasks that involve a hybrid of discrete tasks and continuous parameters, but typically require entire environment states and suffer from heavy computational burden [20]. Training hierarchical policies incorporating temporal abstractions is another approach to tackle long-horizon problems [21], [22], [23]. Some other works adopt behavior trees [7] or skill chaining [24] to address the long-horizon challenge.

Recently, pretrained large language models [25], [26], [27] have demonstrated impressive reasoning abilities to plan over robot skills for solving long-horizon tasks. A line of works uses LLMs to decompose long-range tasks into step-by-step plans in natural language [28], [29], [30], and grounds the language plan to robot execution with language-conditioned control policies. With the development of code-completion LLMs, researchers have also considered using Pythonic code

as the interface between LLM reasoning and robot skills [31], [32] since it can be more directly grounded to available robot APIs and allows more fine-grained behaviors using numerical arguments in function calls. We similarly leverage LLMs to generate robot code for completing long-horizon tasks that involve physical constraints in this work and separate semantic planning, numerical argument computing, and code writing to different LLM agents.

The most relevant work to us is RoboTool [33] which also reasons over long-range physical puzzles with multiple LLMs. However, RoboTool only leverages the basic quadrupedal locomotion skills, which essentially treats the quadruped robot as a mobile platform and does not fully utilize its high dimensionality. Our system additionally supports a more diverse set of low-level skills such as pitching up and bipedal manipulation, therefore can be applied to more complex scenarios.

III. METHOD

In this work, we develop a hierarchical system on a quadruped robot to solve challenging long-horizon tasks that require a smart combination of dynamic skills. For high-level reasoning over the long horizon, we design a cascade of LLMs to figure out hybrid discrete-continuous plans represented as parameterized robot skill API calls. At the low level, we instantiate motion planning and control for locomotion and manipulation skills with model-free RL.

A. High-level reasoning for long-horizon tasks with LLMs

To address the challenging reasoning and grounding problem, we design a cascade of three LLM agents with different focuses as the high-level reasoning module as shown in Fig. 2. A semantic planner focuses on task decomposition with constraints from physical affordances and robot capabilities and sketches a plan of primitive skills for the long-horizon problem. A parameter calculator figures out the precise arguments for the primitives in the plan. A code generator converts the plan and the arguments into one single piece of executable code that can strategically combine the available low-level skills to handle the long-horizon problem. We use GPT-4-turbo-preview [34] for all the LLM agents [1].

1) Semantic planner: The agent reasons at an abstract level to predict a feasible plan from natural language descriptions of the task setup and the available robot skills. The required strategy may differ according to variations in the environment configurations, e.g., the robot can directly climb up stairs if the slope is within its ability but it needs to seek a box to step on before climbing otherwise. Different from previous approaches that run planning for specific environment configurations, we ask the planner to consider different conditions and use "if-else" clauses to unify all strategies into a single plan. Therefore, our generated plan could merit generalization over different setups at test time.

---

[1]We also tried other LLMs such as GPT-3.5 [26] and Kimichat [35]. GPT-3.5 struggles to generate feasible abstract plans due to its deficiency in reasoning. Kimichat can generate reasonable plans given the environment descriptions and constraints but fails to calculate the accurate parameters even when prompted with detailed instructions.
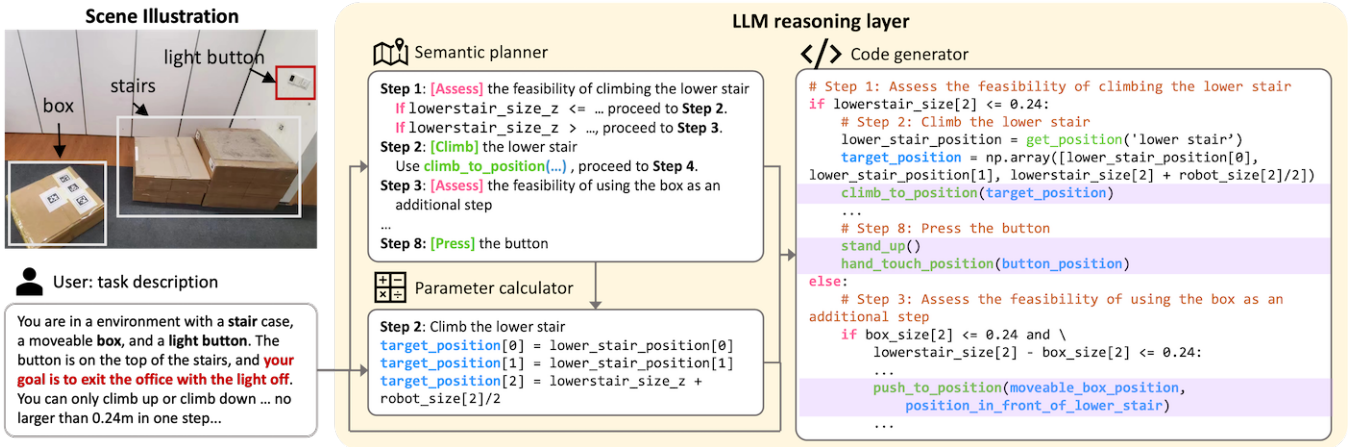
Fig. 2: Illustration of the LLM-based high-level reasoning layer that generates hybrid discrete-continuous plans from the task description in language. It is composed of a semantic planner that proposes a solution consisting of branches conditioned on environment specifications and primitive actions, a parameter calculator that fills in arguments for the actions, a code generator to summarize the plan as executable robot code. The texts are abbreviated generated contents from the LLMs.

Fig. 3: The decomposed plan generated by the semantic planner LLM. The plan unifies multiple control branches depending on the environment specification. The steps in the plan are either feasibility checks that could branch out different conditions, or sequences of primitive skills.

We first provide the planner with descriptions of each motion skill, the physical rules to follow, and an example showing the output format. We then give environment descriptions, specifically the information about the objects contained in the environment, the task to complete, and the limit of the robot's low-level skills. Note that the environment information does not contain numerical values about the objects. Instead, we encourage the planner to use "if-else" clauses in the plan to consider different cases.

The output of the planner is shown in Fig. 3. It formulates the plan as a workflow with branches and jumps. Each step is either a feasibility check or an action step. A check step redirects the control flow to different steps depending

on the environment configuration or raises an error if the task is deemed unsolvable given all the constraints. An action step invokes one or more parameterized skills to be executed sequentially, but the corresponding parameters may be incorrect or undefined, e.g., the predicted target position for the climbing skill in step 2 does not consider the size of the robot. The parameters are handled by a calculator described next.

2) Parameter calculator: We leverage a second LLM to obtain the correct parameters for the invoked robot skills in the plan. Previous research [32], [36], [33] find that a single LLM is not proficient at reasoning across multiple levels of abstraction. We similarly observed that the planner for task decomposition is erroneous regarding detailed parameters for the skills, therefore assigning the calculation to another LLM.

The parameter calculator is designed to generate formulas that can be executed to calculate the 3D targets from variables given in the environment description. It takes the predicted plan along with the original environment description as input, and is prompted with rules for computation and one example output. For each action step, the calculator first parses out the primitive skill that needs to compute arguments, then thinks about how to do the calculation verbally, and finally gives the formulas for every dimension of the arguments. The snapshot of its output can be seen from Fig. 2.

3) Code generator: Given the abstract plan and the detailed parameters for the skills in the plan, we use a third LLM agent to convert them into executable Python code that can directly orchestrate the low-level locomotion and manipulation skills. We prompt the code generator with documented function definitions and global variables that can be used in the code, rules for code writing, and one dummy example for formatting. The available functions are the skill set of the robot and a perception function to query the position of an object. The global variables are geometrical

properties of objects, whose values will be assigned at test time. We then provide the abstract plan, the parameters, and the environment description to the code generator. The code output is shown in Fig. 2. The code generator interprets the control flow in the plan as conditioning blocks, triggers relevant skills in each branch, and performs appropriate arithmetic operations to obtain parameters.

B. Low-level motion planning and control with RL

We implement robot skill APIs used in the high-level reasoning layer with RL-based motion planning and control policies. We first train control policies for short-term motions including quadrupedal locomotion on terrains, i.e., `climb_to_position`, and bipedal locomotion and manipulation skills `stand_up`, `hand_touch_position`, `sit_down`. Building upon the control policies, we develop motion planning policies for mid-term strategies such as `push_to_position` for moving objects and `walk_to_position` for navigation with collision avoidance. To switch between the skills more robustly, we further fine-tune the policies by initializing each of them from the termination states of the preceding skill following the predicted order in the LLM-generated code. All these skills are trained with PPO [37] in domain randomized simulation.

1) Learning control policies for short-term motions:
Quadrupedal locomotion is formulated as a velocity-tracking motion, and is trained with a multi-stage curriculum of increasingly challenging terrains to efficiently learn a robust policy. In the first stage, a walking policy is trained on a flat surface following the same reward specification of [38]. It is then transferred to the second stage to learn locomotion over static stairs where the stair height ranges from 0 to 0.35m. After the policy is capable of climbing up the highest stairs, it is transferred and continues to be trained in the final stage of environment where the stairs are built from multiple movable boxes. The policy is trained to track a linear velocity along a fixed direction across the boxes while avoiding moving the boxes at the same time. This final stage encourages the robot to learn a gentle and robust climbing strategy without stepping on the edges of the terrain.

Bipedal locomotion and manipulation To facilitate interaction with objects, we develop bipedal locomotion and manipulation policies that control the quadrupedal robot to reach target positions with its front legs while standing on its hind legs. We first train a bipedal locomotion policy that can pitch up from a four-leg standing pose, then track a linear bipedal walking velocity using the method in [39]. Afterward, we train a bipedal manipulation policy to reach target positions using one front leg with a summation of three categories of reward terms. The first category is position-tracking rewards that encourage the robot to move its left front toe to the desired position as close as possible. To reduce jitter around the desired position, the robot is also given a bonus reward linear to the number of consecutive steps during which the distance to the desired position is within a threshold. The second category shapes the robot base and the other three legs into an upright standing pose. The third

category is regularization terms that penalize abrupt actions, large joint velocities, and dangerous collisions with the robot. The policy is trained from states in bipedal standing poses with the robot base pose and the joint positions sampled from a hand-crafted range. To transit from bipedal to quadrupedal standing poses, we also train a sit-down policy following the reward in [39].

2) Learning motion planning policies with hierarchical RL: We build object-pushing and obstacle-avoiding skills with hierarchical RL, both trained as mid-level motion planning policies over the previously obtained quadrupedal locomotion controllers. The **object-pushing policy** aims to manipulate an object to a desired pose. It is trained to predict the linear and angular velocity for the locomotion controller every 0.5 seconds. The policy takes low-dimensional states including the current and the desired object pose, the robot body pose and the object size as input. The primary reward encourages a shorter distance between the current and the desired object pose. We additionally credit the robot to face towards the object so that it can keep the vision of the object.

As for the **obstacle-avoiding policy**, it aims to control the robot to move to a desired position and orientation while avoiding collision with an obstacle in the scene. The policy shares the same action space as the object-pushing policy and predicts actions every 0.2 seconds. The observation similarly contains the current pose of the obstacle, the current and the desired pose of the robot base and the size of the obstacle. The rewards consist of a pose-tracking term that encourages the robot to match the target pose and an obstacle-avoidance term that penalizes the robot for being too close to the object.

3) Chaining RL skills: Directly executing different skills sequentially often leads to suboptimal behavior since the terminate state distribution of one policy may not perfectly match the initial state distribution of the next policy. For instance, the quadrupedal climbing policy is trained from initial states where the edges of all the stairs are parallel. However, the preceding box-pushing policy often terminates with the box deviating from the ideal pose right beneath the existing stairs. Executing the climbing skill from the out-of-distribution states will lead to missteps and falls, thus resulting in lower success rates as more skills are executed. To mitigate the impact of the preceding strategy's termination state on the subsequent strategy, we collect a set of termination states of the preceding policy of each policy, then fine-tune the policy from a mixture of the original initial state distribution and the collected states. Note that we only run fine-tuning for the pair of policies that will be executed sequentially according to the plan generated by the reasoning layer to reduce the computation burden.

IV. EXPERIMENTS

A. Experiment setup

We experiment with two benchmark tasks that require strategic high-level reasoning. One task is turning off the light with the button high beyond the reach of the robot. The desired solution is to first build up stairs from boxes, then climb on top of the stairs and stand up to press the

button. The other task is delivering a package into a room with the door closed. The robot should ring the doorbell to ask a human inside the room to open the door, and then push the package. We generate robot code as illustrated in Fig. 2 for each task then execute the code in simulation or on the real robot.

To facilitate the successful deployment of RL policies to a real Xiaomi Cyberdog2 [40] robot, we add domain randomization to the simulation including joint friction, joint damping, mass displacement, motor strength, and observation noise to the object positions during training. We attach multiple AprilTags [41] to the surface of the objects and use a forward-view RealSense D430 camera on the robot for pose estimation. Since the tags frequently go out of the view of the cameras during deployment, we also mimic the phenomenon in simulation by freezing the relevant observations when the object is out of the camera's field osf view. We also randomly freeze the observations to mimic the situation when the detection fails due to the motion blur of captured images.

B. Ablation studies

Ablation on the high-level reasoning layer: We first verify the proposed design of the high-level reasoning layer composed of a cascade of LLM agents in the simulation. We compare our method **LLM (S+P+C)** that leverages a semantic planner, a parameter calculator and a code generator with the following variants:

- **LLM (S+C)** that removes the parameter calculator agent, i.e., passing the output of the planner and the task description to the coder. The coder needs to calculate the parameters to write functional programs;
- **LLM (C)** with code generator only, i.e., directly generating code from the task description.

The performances are measured using (i) the overall success rate for completing the whole task, and (ii) a normalized distance metric defined specific to the semantic of each task. The light-switching task is considered successful when the distance between the front-left end-effector and the button is smaller than 0.03m in the direction perpendicular to the wall and 0.06m in the other two axises parallel to the wall. The normalized distance is calculated as the shortest distance between the robot and the button within one episode divided by the initial distance. The success condition for the package-delivery task is when the bounding box of the package is in the room. We also report the shortest distance between the package and a fixed spot behind the door within each episode normalized by the initial distance for this task.

We use a temperature of 0.2 to generate 3 runs of code for each method. The generated codes are evaluated for 100 trajectories in simulation. In each trajectory, we randomize the height and density of the boxes, the initial position and density of the package, the height of the button and doorbell, and the dynamics of the robot joints. For the light-switching task, the height of the lower stair will affect the necessity of pushing the box to construct the intermediate platform.

As reported in Table I, removing the parameter calculator and the semantic planner leads to a significant performance drop. Note that LLM (S+C) and LLM (C) fail to achieve any success in the light-switching task since they mostly get stuck at the first executed skill with incorrect arguments. The arguments generated without the parameter calculator fail to consider the bounding box of objects and their geometric relationship. As the code snippets in Fig. 4 show, LLM (S+C) sets the target position of climbing to the center of a stair rather than the top surface, and it decides to push the movable box exactly beneath the lower stair, which is physically infeasible. LLM (C) fails to consider basic physical concepts like volumes. For example, it calls a walking skill with the target position at the center of a box, which will collide with the box and make the later steps more difficult to succeed. Similarly, in the package-delivery task, the variants without the parameter calculator struggle to obtain the correct arguments. Some sampled code plans would even attempt to push the box into the locked room without ringing the bell.

Comparison with hierarchical RL: We also validate the LLM-based reasoning module by comparing it with a hierarchical RL (HRL) baseline. It trains a high-level RL policy that operates over the same set of low-level skills used in our system to function as the reasoning module. The high-level policy jointly predicts the discrete low-level skill ID and its continuous parameters every 0.1 seconds. We implement a two-head MLP with shared base layers and separate output layers for discrete and continuous predictions respectively. For the light-switching task, the policy is trained with a reward that encourages the robot to get its front-left end-effector closer to the light button, and for the package delivery task, it is trained with a reward regarding the distance between the package and the door. We early-terminate an episode when the robot falls or moves too far away from its initial position. The high-level policy is optimized with PPO [37]. As shown in Table I, HRL cannot outperform LLM-based methods. In the light-switching task, HRL learns a sub-optimal policy that consistently tries to climb straight up the stairs instead of using the box as a step when necessary. This results in many falls when trying to climb up the stairs. As for the package-delivery task, HRL only learns to perturb the package closer to the door a little bit with quadrupedal locomotion skills, and cannot discover how to unlock the door. They both fail to learn the correct solutions since the vast search space makes it almost impossible to explore the exact combination of skills without prior knowledge.
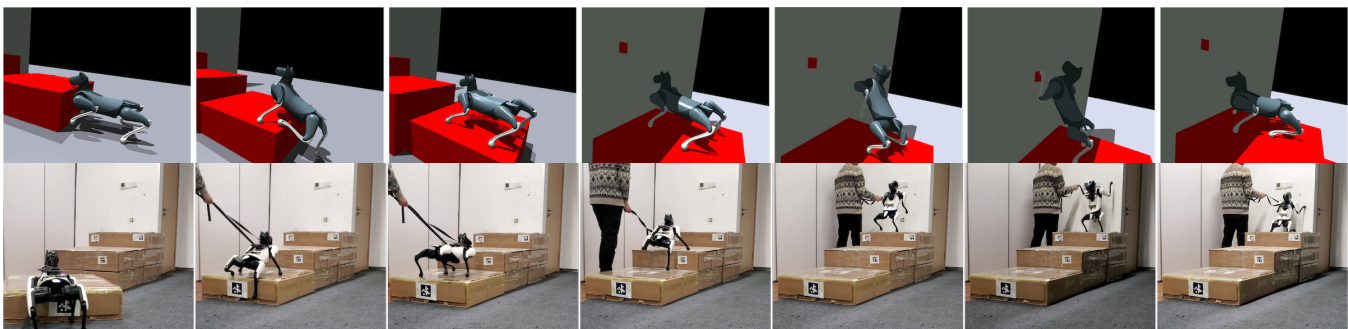
Ablation on fine-tuning RL skills for smooth chaining: We study whether chaining RL skills via fine-tuning helps alleviate compounding errors by comparing our method with a variant without fine-tuning the low-level skills from augmented initial states. The two sets of skills are orchestrated with the same high-level code plans and are tested in the light-switching task. As reported in Table II, our method with additional fine-tuning performs significantly better in terms of the success rate and the relative distance metric. Since the two methods use the same high-level plans, it indicates that the proposed fine-tuning stage is critical to the robustness of the system when transiting between different skills.

```
# Step 1: Feasibility Check for Climbing Lower Stair
if lowerstair_size_z <= 0.24:
    # Step 2: Climb Lower Stair
    lower_stair_position = get_position('lower stair')
    target_position_lower_stair = [lower_stair_position[0],
lower_stair_position[1], lowerstair_size_z + robot_size[2]/2]
    climb_to_position(target_position_lower_stair)
else:
    # Step 1.1: Feasibility Check for Using Box as an Additional Step
    if box_size_z <= 0.24 and (lowerstair_size_z - box_size_z) <=
0.24:
        # Step 3: Move Box and Use as Additional Step
        box_position = get_position('moveable box')
        position_in_front_of_lower_stair = [get_position('lower
stair')[0] - (lowerstair_size_x/2 + box_size_x/2), get_position('lower
stair')[1], box_position[2]]
        push_to_position(box_position,
position_in_front_of_lower_stair)
...
```
LLM (S+P+C)

```
# Step 1: Assess Climbing Feasibility for Lower Stair
if lowerstair_size[2] <= 0.24:
    # Step 2: Climb Lower Stair
    lower_stair_position = get_position('lower
stair')
    climb_to_position(lower_stair_position)
else:
    # Check if the box can be used as an additional
step
    if lowerstair_size[2] - box_size[2] <= 0.24:
        # Step 2b: Move the box in front of the lower
stair
        box_position = get_position('moveable box')
        target_position_in_front_of_lower_stair =
get_position('lower stair') - np.array([0, 0,
box_size[2]])
        push_to_position(box_position,
target_position_in_front_of_lower_stair)
...
```
LLM (S+C)

```
# Step 1: Assess the height of the stairs
and decide if the box is needed to climb the
lower stair
lower_stair_position = get_position('lower
stair')
if lowerstair_size[2] <= 0.24:
    # Climb the lower stair
    walk_to_position(lower_stair_position)
    climb_to_position(lower_stair_position +
np.array([0, 0, lowerstair_size[2]]))
elif box_size[2] <= 0.24 and
(lowerstair_size[2] - box_size[2]) <= 0.24:
    # Move the box to the lower stair and
use it as a step
    push_to_position(box_position,
lower_stair_position - np.array([0, 0,
box_size[2]/2]))
...
```
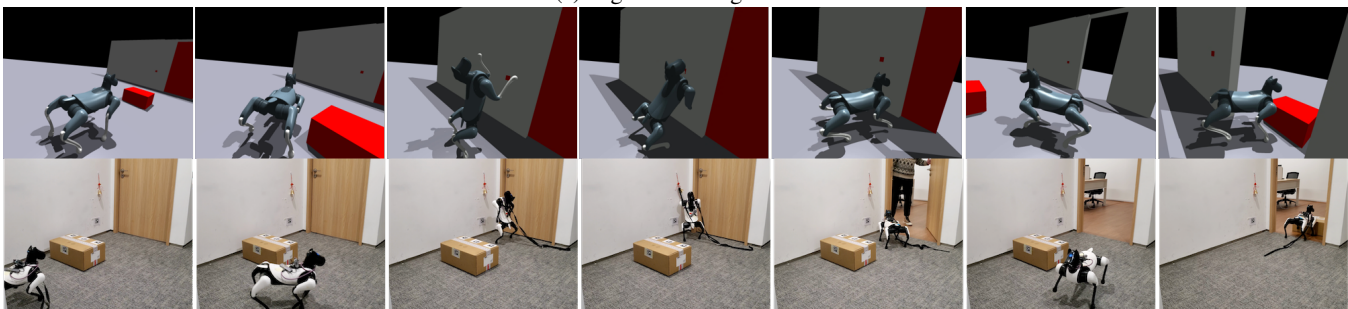**LLM (C)**

Fig. 4: The code snippets generated by different variants of the LLM-based reasoning layer in the light-switching task. Without the parameter calculator, the generated arguments by LLM (S+C) and LLM (C) fail to ground to the physical world. Without the semantic planner, the generated code from LLM (C) contains more unnecessary primitive actions.

TABLE I: Ablation studies on the reasoning layer. The mean and standard deviation of the success rate and a distance metric in two tasks across 3 seeds are reported. The composition of multiple LLM agents is critical to the good performance of the system. LLM-based reasoning performs better than an RL-based high-level policy.

	Light switching		Package delivery	
	success ↑	normalized distance ↓	success ↑	normalized distance↓
LLM (S+P+C)	**0.747±0.012**	**0.032±0.004**	**0.770±0.022**	**0.265±0.016**
LLM (S+C)	0.0±0.0	0.326±0.002	0.0±0.0	0.697±0.408
LLM (C)	0.0±0.0	0.397±0.083	0.0±0.0	0.483±0.249
Hierarchical RL	0.0±0.0	0.662±0.196	0.0±0.0	0.933±0.021

(a) Light-switching task.

(b) Package-delivery task.

Fig. 5: Execution of LLM-generated long-term strategies for two benchmark tasks in the simulation and real world.
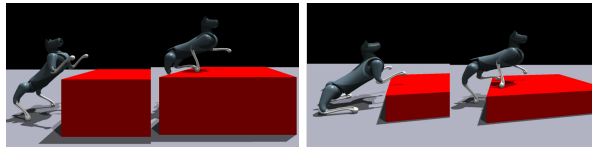
C. Main results

We test our system on two aforementioned benchmark tasks in the real world. As shown in Fig. 5a, we execute the generated code together with the RL motion skills to turn off the light. In the initial setup, the height of the light button is out of reach with any single skill, and the first level of stairs is

TABLE II: The effectiveness of fine-tuning low-level RL policies from the terminal states of preceding policies.
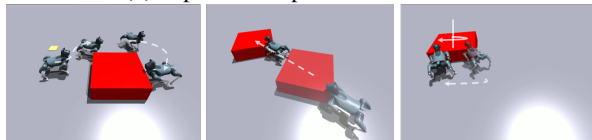
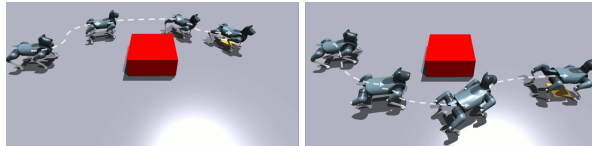|	success rate ↑	normalized distance ↓
w/ fine-tune	**0.747±0.012**	**0.032±0.004**
w/o fine-tune	0.540±0.020	0.177±0.012

(a) Quadrupedal locomotion controller.

(b) Bipedal manipulation controller.

(c) Object-pushing motion planner.

(d) Obstacle-avoidance motion planner.

Fig. 6: Visualization of versatile behaviors driven by the RL-based skill repertoire, covering short-term locomotion and manipulation controllers (a)(b) and mid-term motion planning skills (c)(d). Each policy is capable of a range of continuously parameterized problems.

too high for the robot to climb directly. Considering the scene configuration, the LLM planner decides to push a lower box in front of the existing stairs so that the robot can use it as an intermediate platform to climb up then to touch the button. Finally, it runs a sit-down policy to transit to the quadrupedal pose. In Fig. 5b, we show how the quadrupedal robot delivers a package into a room with the door closed initially. To get access to the room, our reasoning layer figures out a strategy to ring a bell to notify the human inside the room to open the door. To protect the package from unnecessary collisions, it leverages the obstacle-avoidance locomotion skill when moving to the bell and before pushing the package.

D. More analysis

Analysis of low-level RL skills: We showcase how the parameterized motion planning and controlling policies enable versatile motions in Fig. 6. The quadrupedal locomotion policy can walk or climb up boxes of various sizes. Fig. 6a shows snapshots when climbing the height of 0.35m and 0.15m. The bipedal locomotion and manipulation
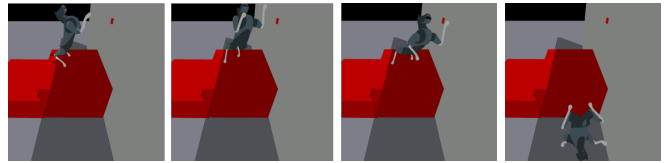

Fig. 7: One typical failure case in the light-switching task. The quadruped robot falls to the ground from the stairs after it loses balance during the execution of the bipedal manipulation skill.

control policy could touch target spots with different relative positions to the robot, as shown in Fig. 6b, where the robot is adapting itself when the target position is lower, higher, to its left or right. As for the motion planning policies, the object-pushing policy demonstrates some adjustment trajectories when aligning the pose of the box to the target, shown in Fig. 6c. We test the robot to walk to different goal positions while avoiding collision with an obstacle shown in Fig. 6d, and find the policy can choose different paths according to the target position.

Failure case: The quadrupedal robot may enter dead ends when the execution of some low-level skills fails. As shown in Fig. 7, the robot occasionally loses balance after standing up and then falls from the stairs. Since our system gives plans in an open-loop manner currently, the robot cannot get an updated plan in such cases. In addition, our system cannot invent policies beyond the fixed set of low-level skills, and thus cannot recover from fatal states without a reset policy.

V. CONCLUSION

We present an LLM-based system for tackling long-horizon tasks with a quadruped robot that extends the autonomy of quadrupeds from shorter-term motions for specific skills to long-range complex behaviors that require orchestrating multiple locomotion and manipulation policies. The high-level task planning is enabled with our design of multiple LLM agents for decomposing the whole task into robot actions and then grounded to parameterized robot API calls. The low-level control that exploits the agility of quadrupeds to unlock rich environment interactions is instantiated with model-free RL. Our LLM reasoning layer plans in an open loop. Allowing it to improve from feedback such as execution traces with in-context learning is an interesting future direction. Currently, the reasoning layer plans over a fixed set of low-level robot skills crafted by experts. Future research could investigate the possibility of actively discovering new skills to learn when encountering unprecedented circumstances.

References

[1] P. G. De Santos, E. Garcia, and J. Estremera, *Quadrupedal locomotion: an introduction to the control of four-legged robots*. Springer, 2006, vol. 1.

[2] P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5761–5768.

[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[4] F. Shi, T. Homberger, J. Lee, T. Miki, M. Zhao, F. Farshidian, K. Okada, M. Inaba, and M. Hutter, "Circus anymal: A quadruped learning dexterous manipulation with its limbs," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2316–2323.

[5] M. Sombolestan and Q. Nguyen, "Hierarchical adaptive loco-manipulation control for quadruped robots," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 156–12 162.

[6] P. Arm, M. Mittal, H. Kolvenbach, and M. Hutter, "Pedipulate: Enabling manipulation skills using a quadruped robot's leg," 2024.

[7] X. Cheng, A. Kumar, and D. Pathak, "Legs as manipulator: Pushing quadrupedal agility beyond locomotion," in *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*. IEEE, 2023, pp. 5106–5112. [Online]. Available: https://doi.org/10.1109/ICRA48891.2023.10161470

[8] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2015, pp. 179–195.

[9] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[10] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[11] Z. Zhuang, Z. Fu, J. Wang, C. G. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao, "Robot parkour learning," in *Conference on Robot Learning*. PMLR, 2023, pp. 73–92.

[12] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, "Extreme parkour with legged robots," *arXiv preprint arXiv:2309.14341*, 2023.

[13] S. Jeon, M. Jung, S. Choi, B. Kim, and J. Hwangbo, "Learning whole-body manipulation for quadrupedal robot," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 699–706, 2023.

[14] C. Schwarke, V. Klemm, M. Van der Boon, M. Bjelonic, and M. Hutter, "Curiosity-driven learning of joint locomotion and manipulation tasks," in *Proceedings of The 7th Conference on Robot Learning*, vol. 229. PMLR, 2023, pp. 2594–2610.

[15] C. D. Bellicoso, K. Krämer, M. Stäuble, D. Sako, F. Jenelten, M. Bjelonic, and M. Hutter, "Alma-articulated locomotion and manipulation for a torque-controllable robot," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 8477–8483.

[16] Z. Fu, X. Cheng, and D. Pathak, "Deep whole-body control: learning a unified policy for manipulation and locomotion," in *Conference on Robot Learning*. PMLR, 2023, pp. 138–149.

[17] Y. Ji, Z. Li, Y. Sun, X. B. Peng, S. Levine, G. Berseth, and K. Sreenath, "Hierarchical reinforcement learning for precise soccer shooting skills using a quadrupedal robot," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 1479–1486.

[18] X. Huang, Z. Li, Y. Xiang, Y. Ni, Y. Chi, Y. Li, L. Yang, X. B. Peng, and K. Sreenath, "Creating a dynamic quadrupedal robotic goalkeeper with reinforcement learning," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 2715–2722.

[19] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[20] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon multi-robot rearrangement planning for construction assembly," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 239–252, 2022.

[21] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *Conference on Robot Learning*. PMLR, 2020, pp. 1025–1037.

[22] X. Yang, Z. Ji, J. Wu, Y.-K. Lai, C. Wei, G. Liu, and R. Setchi, "Hierarchical reinforcement learning with universal policies for multistep robotic manipulation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4727–4741, 2021.

[23] Y. Zhu, P. Stone, and Y. Zhu, "Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4126–4133, 2022.

[24] Y. Lee, J. J. Lim, A. Anandkumar, and Y. Zhu, "Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization," in *Conference on Robot Learning*. PMLR, 2022, pp. 406–416.

[25] 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*, vol. 33, pp. 1877–1901, 2020.

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

[27] 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," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[28] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*. PMLR, 2023, pp. 287–318.

[29] 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*. PMLR, 2022, pp. 9118–9147.

[30] 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," in *Conference on Robot Learning*. PMLR, 2023, pp. 1769–1782.

[31] A. Zeng, M. Attarian, K. M. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. S. Ryoo, V. Sindhwani, J. Lee *et al.*, "Socratic models: Composing zero-shot multimodal reasoning with language," in *The Eleventh International Conference on Learning Representations*, 2022.

[32] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[33] M. Xu, W. Yu, P. Huang, S. Liu, X. Zhang, Y. Niu, T. Zhang, F. Xia, J. Tan, and D. Zhao, "Creative robot tool use with large language models," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[34] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[35] M. AI, "Kimichat," https://kimi.moonshot.cn/, 2023.

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

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[38] G. B. Margolis and P. Agrawal, "Walk these ways: Tuning robot control for generalization with multiplicity of behavior," in *Conference on Robot Learning*. PMLR, 2023, pp. 22–31.

[39] Y. Li, J. Li, W. Fu, and Y. Wu, "Learning agile bipedal motions on a quadrupedal robot," *arXiv preprint arXiv:2311.05818*, 2023.

[40] Xiaomi, "Cyberdog2," https://www.mi.com/cyberdog2, 2023, accessed: Mar. 2024.

[41] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.