Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach

Bin Hu[*]
hubin@zhejianglab.com
Zhejiang Lab

Chenyang Zhao[*]
c.zhao@zhejianglab.com
Zhejiang Lab

Pu Zhang[*]
puz@zhejianglab.com
Zhejiang Lab

Zihao Zhou[*]
zhouzihao@zhejianglab.com
Zhejiang Lab

Yuanhang Yang[†]
ysngkil@gmail.com
Harbin Institute of Technology (Shenzhen)

Zenglin Xu
zenglin@gmail.com
AI Innovation and Incubation Institute, Fudan University

Bin Liu[‡]
bins@ieee.org
Zhejiang Lab

Abstract

Large language models (LLMs) encode a vast amount of world knowledge acquired from massive text datasets. Recent studies have demonstrated that LLMs can assist an embodied agent in solving complex sequential decision making tasks by providing high-level instructions. However, interactions with LLMs can be time-consuming. In many practical scenarios, it requires a significant amount of storage space that can only be deployed on remote cloud servers. Additionally, using commercial LLMs can be costly since they may charge based on usage frequency. In this paper, we explore how to enable intelligent cost-effective interactions between a down stream task oriented agent and an LLM. We find that this problem can be naturally formulated by a Markov decision process (MDP), and propose *When2Ask*, a reinforcement learning based approach that learns when it is necessary to query LLMs for high-level instructions to accomplish a target task. On one side, *When2Ask* discourages unnecessary redundant interactions, while on the other side, it enables the agent to identify and follow useful instructions from the LLM. This enables the agent to halt an ongoing plan and transition to a more suitable one based on new environmental observations. Experiments on MiniGrid and Habitat environments that entail planning sub-goals demonstrate that *When2Ask* learns to solve target tasks with only a few necessary interactions with the LLM, significantly reducing interaction costs in testing environments compared with baseline methods. Our code is available at: https://github.com/ZJLAB-AMMI/LLM4RL.

1 Introduction

To empower embodied agents with the capability to effectively handle demanding sequential decision-making tasks, it is essential for them to possess reasoning abilities that enable them to plan for the long-term consequences of their actions Deitke et al. (2022). Reinforcement learning (RL), particularly deep RL, has emerged as a popular paradigm for addressing these challenges. Deep RL involves agents interacting with the environment and learning from feedback to improve their

[*]Equal contributions
[†]Y. Yang did this work during internship at Zhejiang Lab.
[‡]Corresponding author

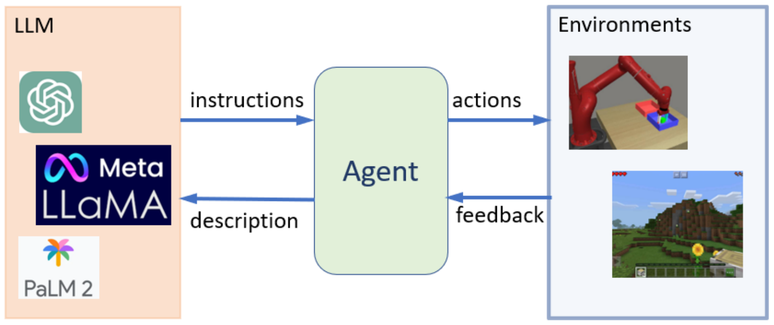


Figure 1: A general framework of using LLMs for solving complex embodied tasks. The LLMs provide high-level instructions based on state descriptions, and the agent generates low-level actions following these instructions and interacts with the target environment to collect further feedback.

decision-making over time. Despite recent advancements, several challenges still remains and limits its vast applications in real world scenarios. For instance, solving complex problems using deep RL often requires significant computational resources. Additionally, safety concerns can arise during the learning phase, especially in scenarios where the agent's exploration might interact with the real world or other sensitive environments Das et al. (2018); Chevalier-Boisvert et al. (2018). As an alternative, the emergence of large language models (LLMs) has shown promise in tackling these issues. Previous studies have demonstrated that LLMs possess reasoning capabilities Radford et al. (2019); Brown et al. (2020); Wei et al. (2022). Researchers have explored leveraging LLMs' reasoning abilities to solve various embodied tasks, including robot manipulation tasks Ahn et al. (2022); Huang et al. (2022); Jiang et al. (2022) and playing video games Dasgupta et al. (2023); Wang et al. (2023a;c). As depicted in Fig. 1, the embodied agent interacts with the environment, gathering information ralated to the target task, and utilizes LLMs as explicit reasoners to make high-level plans using natural language instructions, such as instructing a robot to "pick up a can of coke" or "place an apple on the table".

While the integration of pre-trained LLMs as explicit planners in embodied agents has demonstrated promising results, enabling efficient interaction between these agents and LLMs to solve real-world problems remains challenging. Frequent queries to LLMs can result in unnecessary resource wastage, including fees (if a commercial LLM is used), communication overhead and reasoning time. Whereas insufficient queries to LLMs prevent the agent from obtaining useful instructions in time to adjust its plan to respond to the complex and changing environment.

Determining an appropriate guideline for querying LLMs requires expert knowledge of the target task. Consider a scenario where a robot is instructed to collect a can of coke but encounters a locked door on its way to the kitchen. Ideally, the agent should recognize this incident and adjust its plan accordingly by consulting the LLM on how to deal with the locked door. In such cases, timely decision-making regarding when to consult the LLM planner becomes crucial. Failure to interrupt the ongoing action plan and request a new one in time can hinder task completion progress or even lead to safety issues, such as damaging the door or the robot itself. Conversely, frequent requests for plans from the LLM can be time-consuming and costly, particularly when using commercial LLMs deployed on remote cloud servers that charge based on usage frequency.

In this paper, we propose *When2Ask*, a general approach that trains the agent to make intelligent cost-effective interactions between itself and an LLM remotely deployed. Our objective is to facilitate effective completion of a target task while reducing unnecessary non-informative interactions with the LLM. Specifically, we adopt a Planner-Actor-Mediator framework, similar to Dasgupta et al. (2023), where the planner is a pre-trained LLM used for making plans, the actor contains policies for executing the plans, and the mediator serves as an interface in between by deciding when to request a new plan and generating observation representations for the planner (which are text descriptions).

With a focus on optimizing interacting timings, we use RL to learn an asking policy that instructs the agent to either adhere to the current plan or request a new plan from the LLM.

To summarize, our ***main contributions*** include:

- We propose an RL approach termed *When2Ask* to coordinate the interaction between a downstream task oriented agent and a pre-trained LLM based on the Planner-Actor-Mediator framework Dasgupta et al. (2023). Specifically, we propose to introduce an explicit asking policy in the mediator and train it using an RL approach to determine when to query the LLM planner.
- We conducted a comprehensive evaluation of *When2Ask* against baseline methods based on simulation platforms MiniGrid Chevalier-Boisvert et al. (2023) and Habitat Szot et al. (2021). The results demonstrate that the learned asking policy is able to make intelligent decisions on when to query LLMs, resulting in high success rates with only a few necessary LLM interactions in the testing phase. In contrast to conventional interaction strategies that rely on predetermined termination criteria, *When2Ask* offers a significant advantage by enabling the interruption of an ongoing plan in favor of a new plan that addresses emerging observations.

2 Preliminary

2.1 The Options Framework

We consider sequential decision-making in embodied environments, which is commonly formalized as a Markov decision process (MDP), denoted as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$. Here $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action spaces, $p(s'|s, a)$ denotes the state transition probability function, $r(s, a)$ represents the reward function, and γ is the discount factor. The objective is to learn an optimal policy that maximizes the cumulative reward over time $\sum_t \gamma^t r(s_t, a_t)$, where t denotes the time index.

The options framework extends the traditional notion of action in an MDP to include options, which are essentially closed-loop policies that encompass a sequence of actions over a period of time Sutton et al. (1999); Precup (2000). Options can range from higher-level tasks such as picking up an object or going to lunch, to more primitive actions like muscle twitches and joint torques. The introduction of options allows for the incorporation of temporally abstract knowledge and action within the RL framework in a natural and general manner, thus provides a flexible and intuitive approach to handle complex tasks with varying levels of granularity. Formally, an option ω is defined as a 3-tuple $\langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle$, where $\mathcal{I}_\omega$ represents the initial state set, π_ω denotes the acting policy, and β_ω represents the termination condition for this option. Given a state s, a policy-over-options would select an option ω from the set of available options Ω. The agent would then plan low-level actions by following its current option policy $a \sim \pi(\cdot|s, \omega)$ until the termination condition β_ω is satisfied. In our work, we use pre-defined skills as options and a pre-trained LLM as the policy-over-options to generate high-level options.

2.2 LLM as a Planner

Recent research has shown that LLMs have achieved significant success in various tasks within embodied environments Wang et al. (2023b;c); Ahn et al. (2022). Taking inspiration from these works, we employ a pre-trained LLM to act as a planner, generating a sequence of options using descriptions of observations and tasks. The generated plan, represented as a list of options $[\omega_k]_{k=1,\ldots,K}$, is then executed by following the corresponding option policies. Formally, with text descriptions as input prompts, the LLM outputs a plan in the form of a sequence of options. An actor module subsequently generates low-level actions at each time step, following the option policy $\pi(a|s;\omega_k)$. The policies for the actor module, π_ω, can either be hard-coded or learned from data.

3 Related work

LLMs have emerged as powerful tools for plan generation. There are some research works focusing on designing effective interfaces between planners and actors. In Ahn et al. (2022), LLMs are employed to plan the entire sequence of options at the beginning of each task, enabling the agent to complete the task without further interaction with the planner. In Wang et al. (2023c), the authors introduce a feedback system where the agent requests the LLM to generate an updated plan based on environmental feedback when the execution of the previous plan fails. This approach enhances the robustness of the acting agent in the face of environmental uncertainties. However, these methods often rely on hard-coded failure detectors or apply a threshold to limit the number of permissible MDP state-transition timesteps for an option. In Ren et al. (2023), a framework is proposed for measuring and aligning the uncertainty of LLM-based planners, allowing them to seek assistance from humans when necessary. In addition, Dasgupta et al. (2023) introduce the Planner-Actor-Reporter framework, which includes a reporter module to enhance information exchange between the actor and the LLM-based planner. In this framework, the agent interacts with the LLM at each timestep, regardless of whether new information is acquired or not. While this approach eliminates the need for hard-coded termination conditions and reduces uncertainties during option execution, it leads to excessive resource consumption, especially when utilizing a large-scale and expensive LLM as the planner.

In this paper, we propose learning an interaction policy that enables the agent to interact with a remote LLM in an autonomous and "smarter" way. We empirically demonstrate that our approach overcomes the limitations of previously mentioned hard-coded rule-based interaction protocols or protocols that entail querying the LLM at each timestep.

4 Our Approach When2Ask

We design *When2Ask* based on the Planner-Actor-Mediator framework Dasgupta et al. (2023). In particular, we enhance this framework by incorporating an mediator model that learns to facilitate intelligent and cost-effective interactions between the agent and the LLM using RL.

4.1 The Planner-Actor-Mediator Framework

This framework consists of three components, as illustrated in Fig. 2: the planner, the actor and the mediator. The planner component is responsible for providing high-level instructions to guide the agent's actions. The actor component generates low-level actions based on these instructions. Lastly, the mediator acts as an interface between the planner and the actor, facilitating communication and coordination between them.

Planner The planner component reads text-based descriptions of the current state and generates a plan for the next high-level option or a sequence of options to perform. In our framework, we utilize a pre-trained LLM as the planner. The LLM receives the descriptions of the current observation and is asked to generate high-level skill instructions for the actor. Whenever the planner is activated, the LLM generates an option plan given the descriptions provided with appropriately designed prompts.

Actor The actor component is responsible for planning the low-level actions that align with the instructed option, such as "*go to the red door*" or "*pick up the yellow key*". In our approach, we consider these option policies to be hard-coded using human expert knowledge. It is also possible to pre-train these policies using option-conditioned reward functions to achieve more complex skills.

Mediator In this work, our primary focus is on designing an intelligent mediator component within the Planner-Actor-Mediator framework. Our approach involves training an explicit asking policy using RL to determine when to interact with the planner. The mediator component consists of two sub-components: an asking policy that decides whether to request a new plan from the planner based on observations and the current option, and a translator module that converts observations into text descriptions readable by the LLM. Following Ahn et al. (2022); Carta et al. (2023), we

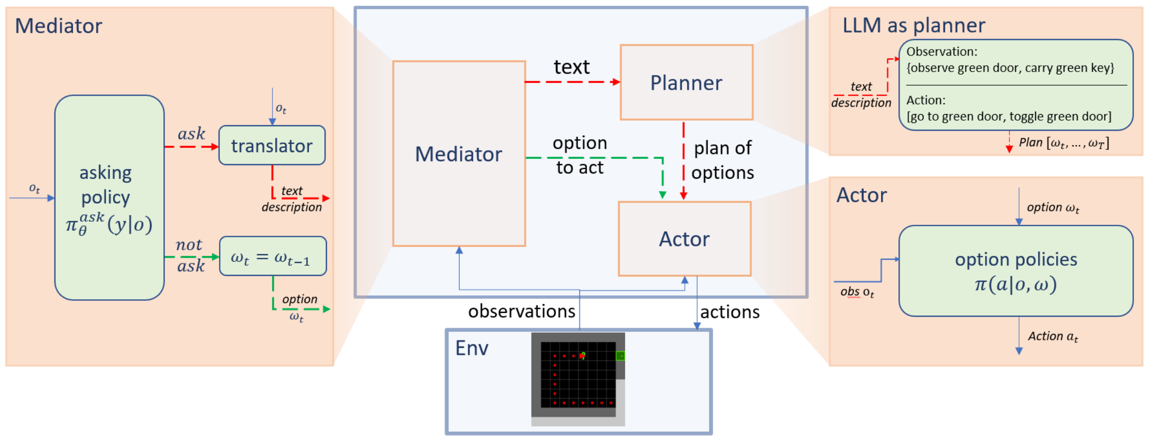


Figure 2: An overview of the Planner-Actor-Mediator paradigm and an example of the interactions. At each time step, the mediator takes the observation o_t as input and decides whether to ask the LLM planner for new instructions or not. When the asking policy decides to *ask*, as demonstrated with a red dashed line, the translator converts o_t into text descriptions, and the planner outputs a new plan accordingly for the actor to follow. On the other hand, when the mediator decides to *not ask*, as demonstrated with a green dashed line, the mediator returns to the actor directly, telling it to continue with the current plan.

assume the availability of an expert translator here. In our experiments, the translator is designed with two stages. Firstly, we extract the IDs of objects, such as keys, doors, and boxes, observed within the field of view of the agent using the built-in interface of the simulation platform. Next, we input this information into our predefined prompt template and output it to LLM in a fixed format. An example of the format can be seen in the green box of Fig.4. The translator can also be learned from data Wang et al. (2023c); Dasgupta et al. (2023).

4.2 Learning asking policy with RL

Here we introduce our proposed approach to learn an asking policy for use in the mediator component.

As mentioned earlier, interacting with the LLM can be costly. Ideally, the asking policy should be trained to enable the agent to request a new plan from the LLM only when it discovers new and informative observations. The expectation is that the LLM will provide a different plan in response to these new observations. To address this, we formulate the problem as an MDP, where the state includes information about the agent's observation and current option in action. The action space consists of two actions: "*Ask*" and "*Not Ask*". In this formulation, the LLM planner is considered as part of the environment that can influence state transitions. The reward function consists of both the task-related return, denoted as r, and an additional penalty term that penalizes unnecessary interactions. Specifically, when the asking policy decides to ask the LLM for a new plan, but the plan provided by the LLM remains the same as the current one, the agent incurs a penalty. This penalty encourages the asking policy to avoid unnecessary interactions and ensures that requesting a new plan is primarily motivated by the discovery of new informative observations.

Denote the asking policy as π^{ask} with its parameters represented by θ. We train this policy using standard on-policy RL methods, specifically Proximal Policy Optimization (PPO) Schulman et al. (2017). The objective function for training the asking policy is defined as follows:

$$\max_{\theta} \sum_{t=1} \left[\gamma^t r_t - \lambda \mathbb{1}(y_t == Ask \wedge \omega_t == \omega_{t-1})\right], \quad (1)$$

where $y_t \in \{Ask, Not\ Ask\}$ represents the decision made by the asking policy at timestep t, r_t denotes the task reward obtained at t, and ω_t is the planned option provided by the LLM at t. The penalty factor λ is used to balance the importance of avoiding unnecessary interactions. Note that if the decision made by the asking policy is "Not Ask" ($y_t == Not\ Ask$), we set ω_t to be the plan executed at the previous timestep, namely let $\omega_t = \omega_{t-1}$. This ensures that if the agent decides not to ask for a new plan, it continues executing the same plan as before. During each iteration, data is collected on-policy using the model π_θ^{ask}.

5 Experiments

We seek to address the following questions by experiments, : can our agent effectively reduce interaction costs while maintaining a high target task completion rate, compared with baseline methods? Can our agent proactively seek assistance from an LLM in exploratory environments? The experimental results indicate that the answer to both questions is yes. As a byproduct, we find that our approach is able to tolerate the imperfection of a crucial component, the translator in the mediator module, which is used to convert observed images into textual descriptions (refer details to the Appendix). We employ two versions of the Vicuna model (Vicuna-7b and Vicuna-13b) Touvron et al. (2023) as LLM planners.

5.1 Baselines

In our experiments, we considered four baseline interaction methods as follows:

Hard-coded The timing and conditions for requesting new instructions from LLMs are manually determined by human experts for each option Wang et al. (2023c). The agent will only request a new plan from the LLM planner when specific termination conditions for the option are met. These conditions involve a goal-finishing detector and a constraint on the maximum number of allowed timesteps. For example, let's consider the option "go to the red door." The termination condition for this option specifies that the agent should reach the target door location or exceed 100 timesteps spent on this option.

Always The agent queries the LLM planner at every timestep, ensuring that any newly acquired information is immediately relayed to the planner Dasgupta et al. (2023). This strategy theoretically leads to better task performance as there is no delay between gathering new information and requesting a re-plan. However, it comes with the drawback of consuming significantly more interaction resources.

Random At each timestep, the agent has a fixed probability of 50% to query the LLM for instructions.

Never The agent never interacts with the LLM. Instead, the policy-over-options (i.e., the planner) is learned using RL techniques based on data collected during interactions with the environment Sutton et al. (1999); Precup (2000). This means that the agent learns to make decisions and generate plans without actively querying the LLM in real-time decision-making. By comparing this method with other approaches, we can assess the contribution of using an LLM as the planner. This comparison helps evaluate the effectiveness and advantages of incorporating a pre-trained LLM into the planning process.

5.2 MiniGrid Experiments

The MiniGrid environment consists of a collection of 2D grid-world environments with goal-oriented tasks Chevalier-Boisvert et al. (2023). In these environments, the agent must navigate within a 2D grid room and interact with specific objects to complete various tasks, such as "open the red door" or "put the green ball next to the yellow box".

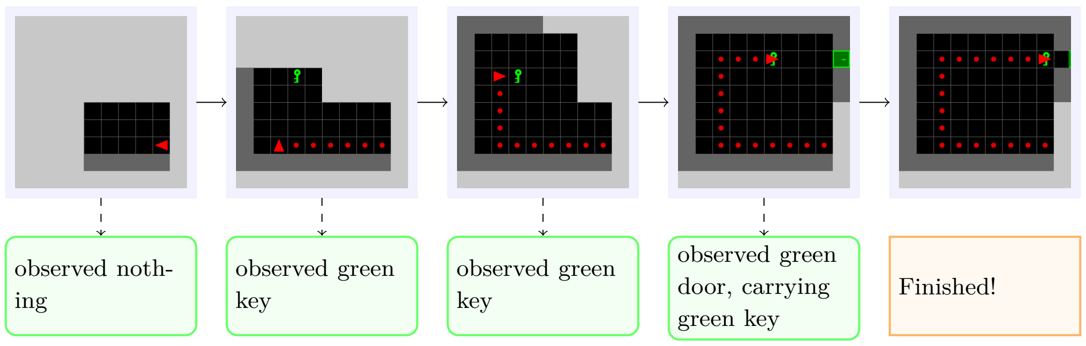
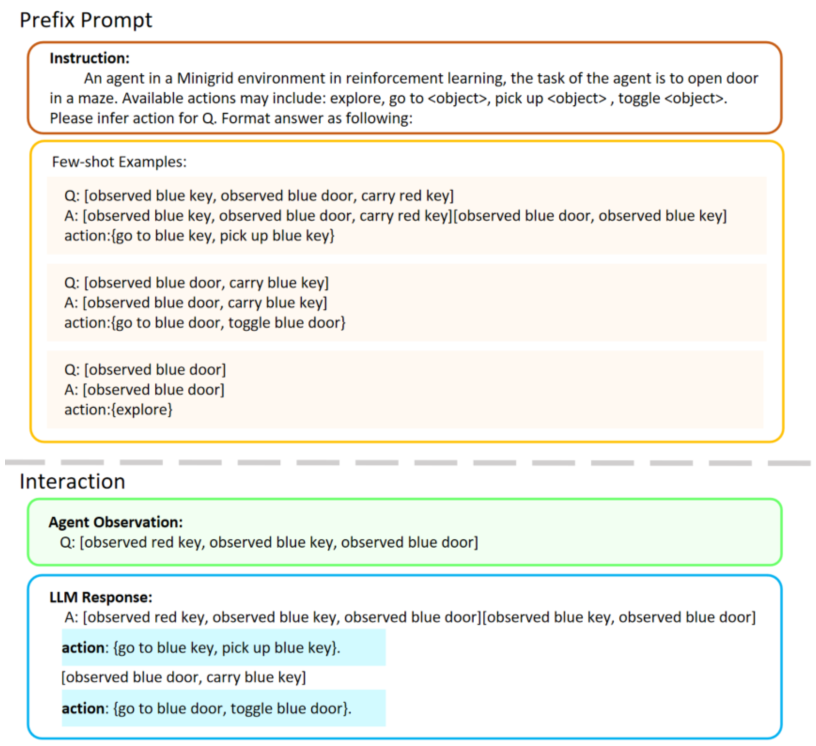


Figure 3: An illustrative example of the partial observations and their corresponding text descriptions in environment *SimpleDoorKey*. The agent is illustrated with a red triangle, and the path it takes is illustrated with red dots. At the start of each episode, the agent is provided with only limited information, with the unexplored area masked (light grey). As the agent progresses in this room, it reveals more information about the room layout for the planner, until it successfully opens the locked door.

One important characteristic of this environment is that the agent's view range is limited. This means that the agent needs to explore the environment and gather useful information to plan its actions effectively. The environment returns observations in the form of a full grid, but with unexplored areas occluded, similar to the concept of "fog of war" in games like StarCraft. Technically, the observation returned by the environment has a shape of $o \in \mathbb{R}^{W \times H \times 4}$, where W and H represent the width and height of the grid, respectively. For an unexplored grid at location $[w, h]$, the observation returns the vector $[-1, -1, -1, -1]$. For an explored grid, the corresponding 4D vector contains information about the object ID, color ID, state ID (e.g., closed or locked for a door), and the agent's direction ID (indicating the agent's orientation if it is present at this location, or 4 otherwise). This design allows us to focus on the agent's reasoning ability and exclude potential influences from factors like memorization. Fig. 3 provides an example of the environment setup in the *SimpleDoorKey* scenario.

In our experiments, we focus on the task of opening a locked door in five distinct environments: *SimpleDoorKey, KeyInBox, RandomBoxKey, ColoredDoorKey*, and *MovingObstacle*. All of these environments are procedurally generated, i.e., the grid layout (including room size, key and door locations) is randomly determined each time the environment is reset. To evaluate generalization, a held-out test set consisting of 100 randomly selected seeds is predefined for each environment. Refer to the Appendix for more details.

We use the Vicuna-7b model for the *SimpleDoorKey*, *KeyInBox*, *RandomBoxKey*, and *MovingObstacle* environments, while for the more complex *ColoredDoorKey* environment we use the Vicuna-13b model. As demonstrated in previous work Min et al. (2022), language models like LLMs require carefully designed prompts and few-shot demonstrations to generalize to different tasks. In our experiments, we provide task instructions and few-shot examples as in-context prompts for each environment. These prompts serve to guide the LLM to understand the task. For the challenging reasoning task in the *ColoredDoorKey* environment, we utilize Chain-of-Thought prompts proposed by Wei et al. (2022). These prompts help the LLM to deal with complex reasoning tasks specific to the *ColoredDoorKey* environment. The few-shot examples in the prompts are used to constraint the output formats. The LLM planner must utilize its ability to generalize and reason in order to comprehend the target task and adjust to situations that deviate from the few-shot examples, such as variations in objects' colors. Fig. 4 provides an example of the prefix prompts and an interaction example in the *ColoredDoorKey* environment. It shows the effective performance of the LLM planner in producing an accurate plan in response to new observations.


Prefix Prompt

Instruction:
An agent in a Minigrid environment in reinforcement learning, the task of the agent is to open door in a maze. Available actions may include: explore, go to <object>, pick up <object> , toggle <object>. Please infer action for Q. Format answer as following:

Few-shot Examples:

Q: [observed blue key, observed blue door, carry red key]
A: [observed blue key, observed blue door, carry red key][observed blue door, observed blue key]
action:{go to blue key, pick up blue key}

Q: [observed blue door, carry blue key]
A: [observed blue door, carry blue key]
action:{go to blue door, toggle blue door}

Q: [observed blue door]
A: [observed blue door]
action:{explore}

Interaction

Agent Observation:
Q: [observed red key, observed blue key, observed blue door]

LLM Response:
A: [observed red key, observed blue key, observed blue door][observed blue key, observed blue door]
action: {go to blue key, pick up blue key}.
[observed blue door, carry blue key]
action: {go to blue door, toggle blue door}.

Figure 4: An example of the prefix prompt and one interaction for the *ColoredDoorKey* environment. Prefix prompt consists of task instruction and few-shot examples. In Chain-of-Thought-style prompts, we add inference processes within the examples. Note that these few-shot examples are only provided for grounding a few but not all task-related knowledge to, and constraining the output formats, of the LLM. We do not need to exhaustively enumerate all knowledge and rules to construct prompts, as a qualified LLM can do logical reasoning based on a limited number of prompts, then provide proper plans (instructions) that are adaptable to new scenarios encountered in the environment.

5.2.1 Can our agent complete target tasks with less interaction costs?

We compare our approach *When2Ask* with baseline methods to evaluate its effectiveness. We analyze the learning curves for both interaction costs (Fig. 5) and task performances (Fig. 6) across all five environments. Additionally, we provide asymptotic performances in Table 1. As is shown, our approach successfully reduces the number of interactions with the LLM while maintaining task performance across all environments. This reduction in interaction cost indicates that our method effectively learns to reduce non-informative interactions with the LLM. Furthermore, our approach maintains consistently high success rates throughout the learning process. This observation indicates that the asking policy learns to filter out unnecessary interactions while still engaging in essential ones to achieve successful task completion.

5.2.2 Can our agent proactively seek assistance from an LLM in exploratory environments?

Upon analyzing the agent's performance in situations where it is expected to ask the LLM planner for help, we observe that the baseline method with a hard-coded asking policy exhibited significantly lower success rates compared to other approaches. This discrepancy occurs because the agent continues executing every option until its termination condition is met, even when it has already gathered sufficient information to complete the task. Consequently, this inefficient approach results in wasted time on each option and ultimately leads to failure in completing the task within the

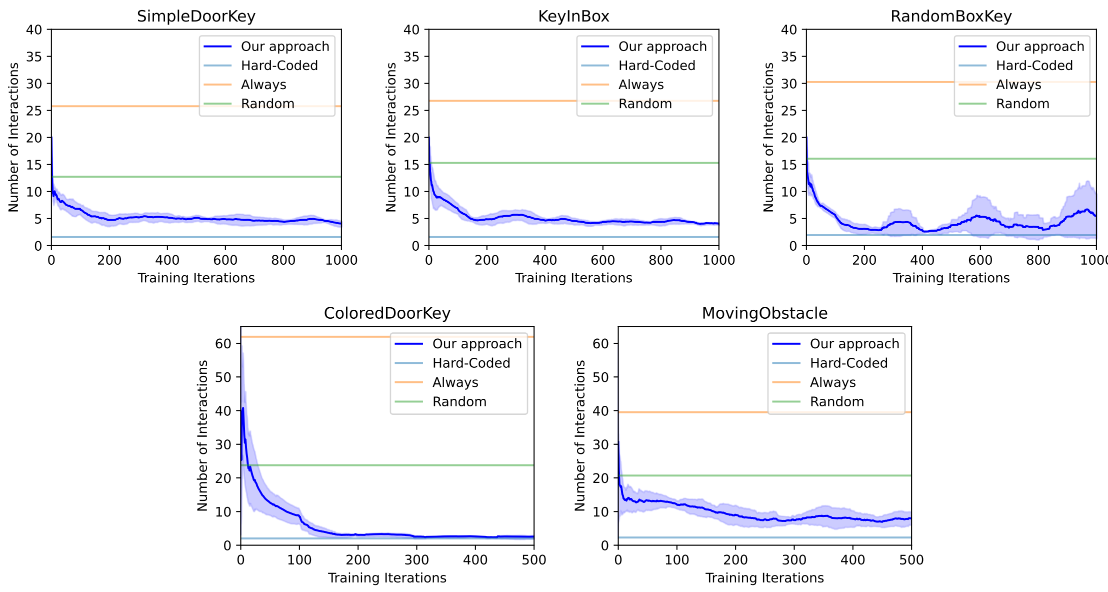


Figure 5: The number of interactions with the LLM vs. the number of RL iterations used for learning the asking policy. It shows that, for every environment, the more thoroughly the asking policy is trained, the fewer interactions with the LLM planner (i.e., the less interaction costs) are required to complete the task. The shaded areas within the curves represent confidence intervals based on three standard errors.

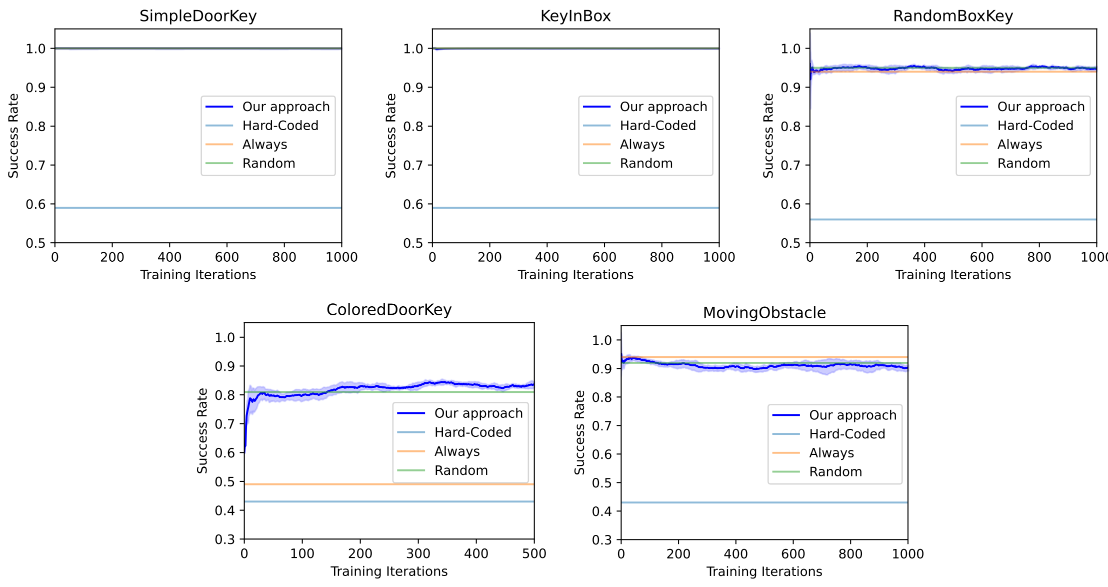


Figure 6: Success rate of completing target tasks vs. the number of RL iterations used for learning the asking policy. It demonstrates that our approach consistently maintains a high success rate across all environments, and outperforms baseline methods in *ColoredDoorKey*.

given time limit. In contrast, *When2Ask*, along with other baseline methods, demonstrates the ability to early-stop options when necessary. As a result, they achieve 100 percent success rates in *SimpleDoorKey* and *KeyInBox*.

Table 1: Asymptotic performance comparison on five *MiniGrid* environments. The performance metrics include the total number of interactions with the LLM, the number of MDP state-transition timesteps, and the success rate for completing a task. *These results show that our approach achieves competitive task performance in terms of success rate while significantly reducing interaction costs (indicated by the number of interactions) compared to Always and Random. Hard-coded* requires the fewest LLM interactions but often fails to complete tasks. All results are averaged over 500 test trials (We use 5 training seeds to initialize the policy network, and conduct 100 independent tests per seed).

Environment	Performance metric	Hard-Coded	Always	Random	Our approach
SimpleDoorKey	*Number of interactions* ↓	**1.58**	25.78	12.75	4.24
	Number of timesteps ↓	64.9	**25.78**	26.55	29.20
	Task success rate ↑	59%	**100%**	**100%**	**100%**
KeyInBox	*Number of interactions* ↓	**1.58**	26.78	15.3	4.33
	Number of task timesteps ↓	65.49	**26.78**	27.46	29.01
	Task success rate ↑	59%	**100%**	**100%**	**100%**
RandomBoxKey	*Number of interactions* ↓	**1.93**	30.26	16.09	3.61
	Number of task timesteps ↓	61.71	30.26	**30.2**	34.41
	Task success rate ↑	56%	94%	**95%**	**95%**
ColoredDoorKey	*Number of interactions* ↓	**2.01**	61.96	23.75	3.29
	Number of timesteps ↓	75.54	61.96	**44.64**	47.87
	Task success rate ↑	43%	49%	81%	**83%**
MovingObstacle	*Number of interactions* ↓	**2.29**	39.49	20.70	6.94
	Number of timesteps ↓	82.36	**39.49**	44.90	48.63
	Task success rate ↑	43%	**94%**	93%	92%

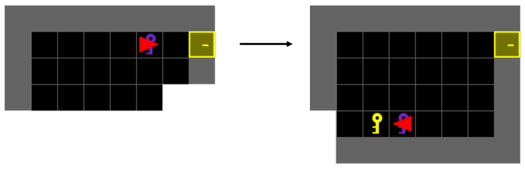
Table 2: Success rate of each stage completions and total number of interactions with the LLM planner in the *Habitat* during testing.

Performance metric	Hard-Coded	Random	Our approach
Stage1 success rate↑	10.8%	7.6%	**53.6%**
Stage2 success rate↑	2.4%	1.6%	**46.4%**
Stage3 success rate↑	2.0%	1.2%	**35.6%**
Total # of interactions↓	**1.00**	295.60	7.99

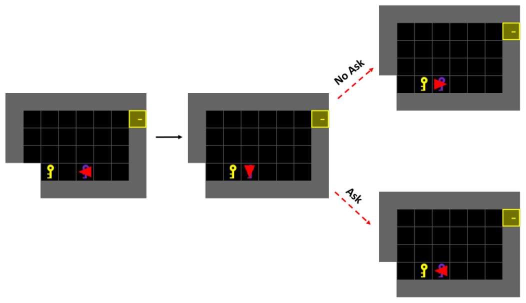
In a specific scenario within the *ColoredDoorKey* environment, as illustrated in Fig. 7a, we see an interesting phenomenon. The agent has chosen to take the *Explore* option and acquired information about the location of the yellow key (frame 2). With use of the *Hard-coded* baseline approach, the agent shall continue with the *Explore* option until it has fully explored the entire room. In contrast, using our proposed approach, the agent can recognize the value of asking the LLM planner for guidance given the current information, and immediately propose querying the LLM planner for an updated plan while ceasing further exploration. The LLM would instruct the agent to efficiently pick up the yellow key without wasting additional time. This example highlights the effectiveness of our approach in recognizing when to seek assistance from the LLM planner and making more efficient decisions based on the available information. By leveraging the power of the LLM planner, our approach enables the agent to make informed choices that expedite task completion and improve overall performance.

5.3 Habitat Experiments

We further evaluate our approach with the Habitat environment Szot et al. (2021). The results indicate the potential of our approach to function effectively in visually realistic domains. The details on the experiment setting are referred to the Appendix.



(a) An example scenario where the agent discovers new information during option *explore*.



(b) An example scenario where the hard-coded translator fails to encode all information.

Figure 7: Two example scenarios where the agent is expected: (a) to ask the LLM planner for help as it has collected useful information for the planner to adjust its plan; and (b) to not ask the LLM, as the LLM may propose wrong options due to an imperfect translator.

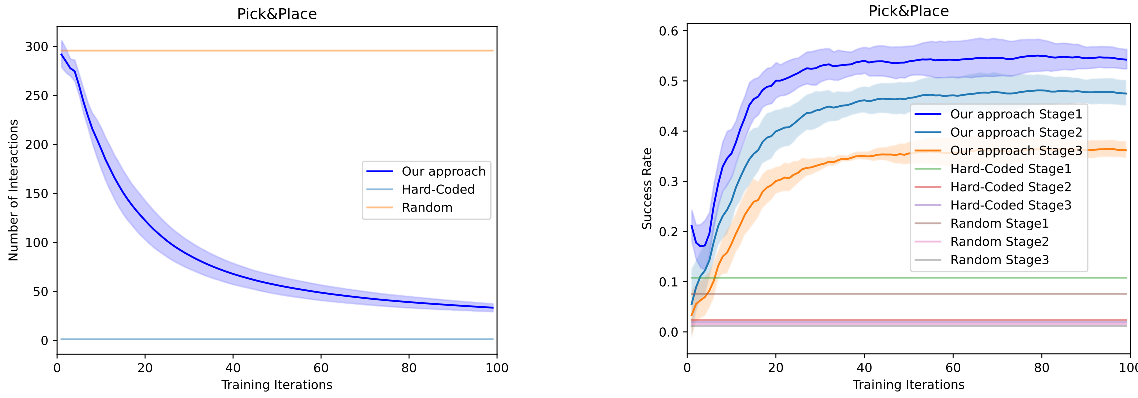


Figure 8: The number of interactions with the LLM (left) and the stage success rates (right) vs. the number of training iterations used for learning the asking policy on the *Pick&Place* task.

We compare our approach against baselines on the *Pick&Place* task. To ensure reliability of experimental results, we utilize 10 training seeds to initialize the policy network. This allows us to explore different initializations and avoid biased results. Subsequently, we select the best policy obtained from these training runs to run 250 independent testing trials. As presented in Table 2 and Fig. 8, our approach significantly outperforms baselines across all three stages. Particularly, compared to the hard-coded baseline where the preset plan is executed step-by-step, our approach is significantly better at addressing the "hand-off problem" Szot et al. (2021) that can arise when the preceding option terminates at a state that makes it challenging for the succeeding option to initiate. This issue is depicted in Fig. 9, where the robot stops at an unfavorable location at the end of the *Navigate* option, resulting in a failure to execute the subsequent *Pick* option. Our approach effectively bypasses this problem by seeking guidance from the LLM planner.

The obtained results demonstrate that the RL learned asking policy effectively establishes a connection between the world knowledge embedded within the LLMs and the downstream fine-grained knowledge embedded within the agent. This connection leads to a superior overall performance of our approach compared to the baselines that do not involve any learning. These findings align with the main observations from our experiments associated with the MiniGrid environments, particularly in the *ColoredDoorKey* scenario, where the RL learned asking policy enables the agent to outperform all baselines in terms of task completion success rate.

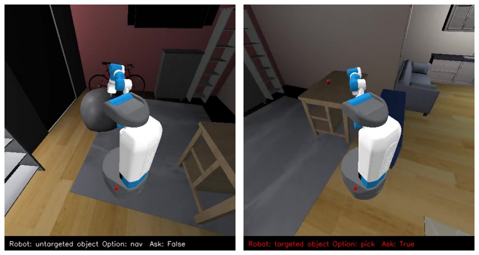

Robot: untargeted object Option: nav Ask: False
Robot: targeted object Option: pick Ask: True

Figure 9: An illustrative example demonstrating the "hand-off" problem in Habitat. The robot's objective is to navigate to the living room and pick up the apple from the table. With the *Hard-Coded* baseline in use (left), according to preset hard-coded rules, the agent must first complete the *Navigate* option before executing the *Pick* option. Consequently, the agent stops at a location where the apple is not visible at the end of *Navigate*, resulting in its future failure in the *Pick* option. With our approach (right), in the middle of *Navigate*, the agent finds itself at a suitable location where the apple can be spotted. The learned mediator interrupts the ongoing *Navigate* and query the LLM planner, which returns the *Pick* option. This helps the agent subsequently pick up the apple successfully. This example demonstrates the effectiveness of our approach in bypassing the "hand-off" issue.

6 Concluding Remarks

We examine the application of RL in acquiring "mediator" policies for instruction-following agents powered by LLMs. Prior research has indicated that LLMs, when coupled with well-constructed prompts, can effectively generate high-level instructions conditional on state descriptions to devise detailed plans for task completion. Recent frameworks for LLM-driven planning have explored two primary strategies: 1) generating an updated plan at each timestep, and 2) requesting an update only after each plan (option) concludes based on predefined termination criteria. The former method is computationally intensive due to the expense of generating a new response from the LLM, while the latter may encounter challenges as an ongoing plan executed cannot be interrupted in time to respond to new observations.

To this end, we propose *When2Ask*, which involves training a mediator policy to determine when to prompt the LLM to generate an appropriate plan for the present moment. *When2Ask* trains the mediator policy to optimize task-oriented rewards while penalizing cases where the planner was invoked but returned the same plan the agent was already following. Experiment results across different embodied environments illustrate that the learned mediator policies achieve comparable task success rates to fixed policies that query the LLM at each timestep, while significantly reducing the number of queries to the LLM and consequently lowering the computational burden on the agent.

The utilization of LLMs to furnish robots and other autonomous agents with general-purpose reasoning and planning capabilities has shown great potential. However, this potential is somewhat limited by the quality of the mapping between low-level observations and actions, and the high-level LLM-based planner. A long-term goal would be to learn this mapping in an end-to-end way. In the interim, however, it is worth to investigate how different elements of this mapping can be learned, and how much benefit can be gained from such an endeavor, as demonstrated in this study.

Acknowledgments

We appreciate anonymous reviewers for valuable comments that helped us to improve the quality of this paper. This work was primarily supported by Exploratory Research Project (No.2022RC0AN02) of Zhejiang Lab, and partly supported by China Postdoctoral Science Foundation (No.2023M743266), and Zhejiang Provincial Postdoctoral Research Project (No.ZJ2023067). Z. Xu was partly supported by Fudan University (No.JIH2325015Y).

References

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

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.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. *arXiv preprint arXiv:2302.02662*, 2023.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–10, 2018.

Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. Collaborating with language models for embodied reasoning. *arXiv preprint arXiv:2302.00763*, 2023.

Matt Deitke, Dhruv Batra, Yonatan Bisk, Tommaso Campari, Angel X Chang, Devendra Singh Chaplot, Changan Chen, Claudia Pérez D'Arpino, Kiana Ehsani, Ali Farhadi, et al. Retrospectives on the embodied ai workshop. *arXiv preprint arXiv:2210.06849*, 2022.

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.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2022.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

Doina Precup. *Temporal abstraction in reinforcement learning.* University of Massachusetts Amherst, 2000.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, and Jake Varley. Robots that ask for help: Uncertainty alignment for large language model planners. In *7th Annual Conference on Robot Learning*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34:251–266, 2021.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023b.

Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023c.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Zihao Zhou, Bin Hu, Chenyang Zhao, Pu Zhang, and Bin Liu. Large language model as a policy teacher for training reinforcement learning agents. In *33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

A Appendix

A.1 Experimental settings on MiniGrid

In the basic setups of *SimpleDoorKey* and *KeyInBox*, each room contains only one key and one locked door. In *SimpleDoorKey*, the key is placed on the floor, while in *KeyInBox*, the key is inside a box. The agent needs to explore the room to locate the target door and the key/box, pick up the key, and finally use the key to unlock the target door.

In the *RandomBoxKey* environment, the placement of the key is randomized, either on the floor or inside a box. The agent needs to actively plan its actions based on the feedback from the environment, adjusting its plan depending on whether it observes a key or a box.

ColoredDoorKey introduces multiple keys and only one exit door. Each key and its corresponding door are color-coded, requiring a matching-colored key to unlock the door. This environment tests the agent's ability to identify and utilize color information for successful task completion.

MovingObstacle adds another layer of complexity by introducing obstacles that move randomly within the room, potentially blocking the agent's path. The agent needs to navigate in this dynamically changing environment and adapt its plans accordingly based on new observations.

A.1.1 More details on the design of our agent

Actor In our experiments, the actor comprises a set of pre-defined option policies. The available options are as follows:

- Explore: This option is implemented using preset rules. The specific procedure is as follows: first, the agent is instructed to move to the top-left corner of the task environment, and then proceed to traverse each row alternately while scanning, until all information within the environment becomes visible. This option enables the agent to uncover unexplored areas for discovering new information.

- Go to [an object]: With this option, the agent can navigate to an object within the environment. The object can be any interactable element, such as a key, box, or door.

- Pickup [an object]: This option enables the agent to pick up a specified object. It is useful when the agent needs to acquire an item to progress in the task, like grabbing a key to unlock a door.

- Toggle [an object]: Using this option, the agent can the state of a particular object. Examples include opening or closing a door, use a key to unlock a door or open a box.

These pre-defined options provide the agent with a repertoire of high-level actions to choose from during its decision-making process. By selecting the appropriate option based on its current objective and observations, the agent can efficiently navigate and interact with the environment to accomplish the given task. For more details, refer to the supplement materials.

How to train the asking policy? In our experiments, we train a neural network to serve as the asking policy. Specifically, this neural network receives observations from the current and previous frames as input. Before passing these observations to the network, we compute the difference between the two frames. This encourages the asking policy to generate an “ask" action only when there are noticeable changes in the environment compared to the previous frame. The network architecture for the asking policy comprises three convolutional neural network (CNN) layers followed by two multilayer perceptron (MLP) layers. The output of the network consists of logits for each option, indicating the probability of selecting the “ask" or “not ask" action for each option. Therefore, the dimensionality of the network's output is $2 \times K$, where the ($2k$-1)-th and $2k$-th entries collectively determine the action distribution for option k. Here, K represents the size of the option set used in our approach. By training the asking policy network with this architecture, we enhance the agent's ability to make informed decisions regarding whether it should pose a query to the LLM planner or not, based on changes observed in the environment between consecutive frames.

A.2 Our approach can tolerate the imperfection of the translator in the mediator module

In the complex environment of *ColoredDoorKey* in MiniGrid, the baseline interaction method *Always* has been observed to fail in certain corner cases due to flaws of other components within the framework. Fig. 7b presents an example scenario in *ColoredDoorKey* that showcases such a case. In the first frame, the agent is instructed to *go to then pick up the key.* After taking a left turn to drop the carried purple key (frame 2), the LLM instructs the agent once again with *go to then pick up the key*, where the agent should proceed to pick up the yellow key. However, the *Always* baseline fails in this case because the translator does not encode information about the relative position between the agent and the target object accurately. Consequently, the translator returns the same observation [*observed yellow key, observed yellow door, carrying purple key*] for both frames 1 and 2. In contrast, our approach learns “not to ask" for assistance in this particular case, allowing the agent to complete the action of picking up the yellow key before requesting further instructions. This highlights a significant advantage of our approach over baseline methods: it can adapt to situations where the translator's translation process loses a lot of information. The learned asking policy enables the agent to make more informed decisions based on its observations and context, leading to robust performance in scenarios where baseline methods may fail due to flaws of the translator.

A.3 Experimental setting on Habitat

Habitat is a simulation platform specifically designed for end-to-end development of embodied AI Szot et al. (2021). It provides a framework for defining various embodied AI tasks such as navigation, object rearrangement, and question answering. Additionally, it allows for the configuration of embodied agents with specific physical forms and sensors. Agents can be trained using either imitation or reinforcement learning techniques. In our experiments, we demonstrate that our approach can generalize effectively to visually realistic domains by conducting experiments within the Habitat environment.

In our experiments, we focus on the manipulation task known as *Pick&Place*. In this task, the robot agent's objective is to pick up an object from a desk and precisely place it into a designated target receptacle, which in this case is the kitchen sink. The task setting is the same as in the Habitat experiment in Zhou et al. (2024).

The robot agent is equipped with a wheeled base, a 7-degree-of-freedom (DoF) arm manipulator, and a parallel-jaw gripper. Additionally, it features a camera mounted on its "head" that provides a field of view of 90° and captures visual data at a resolution of 256×256 pixels. As a result, the observation space of the environment comprises a visual observation denoted as $o_v \in \mathbb{R}^{256\times256\times1}$ from the depth camera. It also includes a sensor observation $o_s \in \mathbb{R}^{24}$ sourced from various sensors such as joint sensors, gripping sensors, the end effector of the arm, object and target GPS sensors, among others. The action space in our setup is 11-dimensional, consisting of 3 actions controlling the robot positions, 7 actions controlling the robot arm and one action indicating termination. This action space enables the agent to execute precise movements and manipulations necessary for accomplishing the target task.

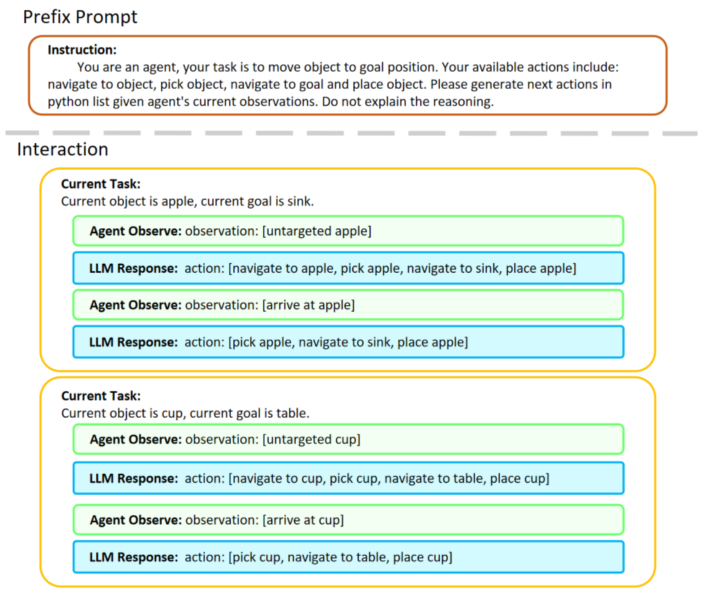
To effectively train each option, we design the rewards based on rearrangement measures. These measures take into account various factors such as the force exerted by the articulated agent, the distance between the object and the goal, and the angle between the agent and the goal. The specific details of these measures can be found in the Habitat documentations Szot et al. (2021).

In the *Pick&Place* environment, as solving the task requires progressively achieving several subgoals, we use a composite stage reward system. More specifically, picking up the object successfully is referred to as *Stage1 Completion* and rewards a value of 1. Achieving navigation to the sink with the object is referred to as *Stage2 Completion* and also rewards a value of 1. Finally, successfully placing the apple into the target sink is referred to as *Stage3 Completion* and grants a higher reward value of 5. It is important to note that if any of the high-level options exceed their designated time limit, the task may terminate prematurely.

A.3.1 Implementation details of our approach on Habitat

Planner We employ the pre-trained Vicuna-7b model as the LLM planner in our approach. In terms of prompt design, we begin by furnishing a concise instruction that conveys information about the target task. Subsequently, we provide a description of the current observation in the form of a Python list. An example of the dialogue generated by the LLM planner can be found in Fig. 10.

Actor In our experiments, we use three high-level options: $\{$*Navigate*, *Pick*, *Place*$\}$, each pre-trained with RL independently. Whenever there is a transition between these options, an automatic execution of the default action *Reset Arm* occurs. To ensure effective training of these options, we use 32 distinct training environment specifications with different object locations and target locations. Additionally, the agent's initial positions are randomly generated each time the environment is reset, guaranteeing variability in training scenarios. For each option, we employ a ResNet18 backbone combined with a 2-layer LSTM architecture to train the corresponding models. During testing, the success rates of *Navigate*, *Pick*, and *Place* are 84%, 92%, and 91% respectively. These pre-trained models remain fixed throughout the task, ensuring consistency and stability during execution.



Instruction:
You are an agent, your task is to move object to goal position. Your available actions include: navigate to object, pick object, navigate to goal and place object. Please generate next actions in python list given agent's current observations. Do not explain the reasoning.

Current Task:
Current object is apple, current goal is sink.

Agent Observe: observation: [untargeted apple]

LLM Response: action: [navigate to apple, pick apple, navigate to sink, place apple]

Agent Observe: observation: [arrive at apple]

LLM Response: action: [pick apple, navigate to sink, place apple]

Current Task:
Current object is cup, current goal is table.

Agent Observe: observation: [untargeted cup]

LLM Response: action: [navigate to cup, pick cup, navigate to table, place cup]

Agent Observe: observation: [arrive at cup]

LLM Response: action: [pick cup, navigate to table, place cup]

Figure 10: An example of the prompts and interactions for the habitat environment. Prefix prompt only contains a short task instruction.

Training of the asking policy Similar to our Minigrid experiment, we stack five consecutive frames of observations as inputs to the asking policy. This enables the network to capture temporal information and make informed decisions based on past observations. The network architecture for the asking policy consists of three CNN layers for embedding visual observations, one MLP layer for embedding sensor observations, and two additional MLP layers to output the logits for the binary question of whether to *ask* or *not ask*.