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

Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, Bin Liu

***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, they require a significant amount of storage space that can only be deployed on remote cloud server nodes. 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 the agent and an LLM. We 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. 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 an LLM, and significantly reduces interaction costs in testing environments compared with baseline methods. Experiment results also suggest that by learning a mediator model to interact with the LLM, the agent's performance becomes more robust against partial observability of the environment. Our code is available at https://github.com/ZJLAB-AMMI/LLM4RL.**

***Index Terms*—Large language models, embodied agents, interaction, reinforcement learning, sequential decision making.**

I. 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 [1]. 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 decision-making over time. Despite recent advancements in deep RL, 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 [2], [3]. 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 [4]–[6]. Researchers have explored leveraging LLMs' reasoning abilities to solve various embodied tasks, including robot manipulation tasks [7]–[9] and playing video games [10]–[12]. 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" for the next step.

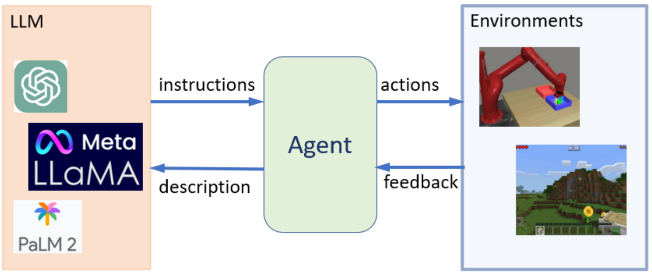


Fig. 1: A general framework of using LLMs for solving complex embodied tasks. The LLMs provide high-level instructions based on state descriptions, and the agent generates low-level actions following these instructions and interacts with the target environment to collect further feedback.

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 adjusting its plan according to online feedback from the environment, making it vulnerable to uncertainties in the 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

Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou and Bin Liu are with Research Center for Applied Mathematics and Machine Intelligence, Zhejiang Lab, Zhejiang 311121 , China. E-mails: {hubin, c.zhao, puz, zhouzihao, liubin} @zhejianglab.com. *(Corresponding author: Bin Liu.)*

Yuanhang Yang, Zenglin Xu are with Harbin Institute of Technology (Shenzhen). E-mails: {ysngkil, zenglin}@gmail.com. Y. Yang did this work during his internship at Research Center for Applied Mathematics and Machine Intelligence, Zhejiang Lab.

The first four authors contributed equally on this work.

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 server nodes 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 deployed on a remote cloud server. Our objective is to facilitate effective completion of a target task while minimizing communication costs incurred from interactions with the LLM. Specifically, we adopt a Planner-Actor-Mediator framework, similar to [10], 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.

We evaluate the performance of *When2Ask* in different embodied environments included in MiniGrid [13] and Habitat [14]. First, we consider five distinct partially-observable MiniGrid environments [13], which require the agent to explore the environment and react to newly acquired information. Experiment results demonstrate that *When2Ask* can effectively balance the desired task performance with the interaction costs associated with using an LLM. Specifically, it achieves competitive task performance with only a few necessary interactions with the LLM. Additionally, we find that *When2Ask* performs more robustly against partial observability of the environments in two scenarios, where the agent needs to handle newly acquired information and unexpected errors, respectively, when providing subsequent plans. We also evaluate *When2Ask* in a visually realistic environment named Habitat [14]. In this environment, the robot agent is required to navigate through different areas of an indoor scenario and manipulate various types of objects. Results showcase the generality of our approach, and indicate a significant advantage of *When2Ask* over baseline methods in terms of task completion success rate.

To summarize, our main contributions are as follows:

- We propose an RL approach termed *When2Ask* to coordinate the interaction between the agent and the LLM based on the Planner-Actor-Mediator framework [10]. Concretely, 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 using both MiniGrid and Habitat environments. 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 test tasks.
- To ensure reproducibility and facilitate future research on the applications of LLMs, we have made our code open source at https://github.com/ZJLAB-AMMI/LLM4RL. This allows other researchers to access and utilize our code for their own experiments and investigations.

In the remainder of this paper, we begin by formalizing the problem based on the options framework [15], [16], and discussing related works in Section II. We then present our approach *When2Ask* in detail in Section III. We thoroughly evaluate its performance by comparing it with baseline methods in various MiniGrid [13] and Habitat [14] in Section IV. Finally, we conclude the paper in Section V.

II. Preliminary

A. 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 in such a framework is to learn an optimal policy that maximizes the cumulative return 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 [15], [16]. 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 for this option, π_ω denotes the acting policy for the option, and β_ω represents the termination condition. 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.

B. LLM as a Planner

Recent research has shown that LLMs have achieved significant success in various tasks within embodied environments [7], [12], [17]. In these studies, LLMs play the role of planners by generating a sequence of options based on descriptions of observations and tasks. The generated plan, represented as a list of options $[\omega_k]_{k=1,...,K}$, 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 pre-trained using RL.

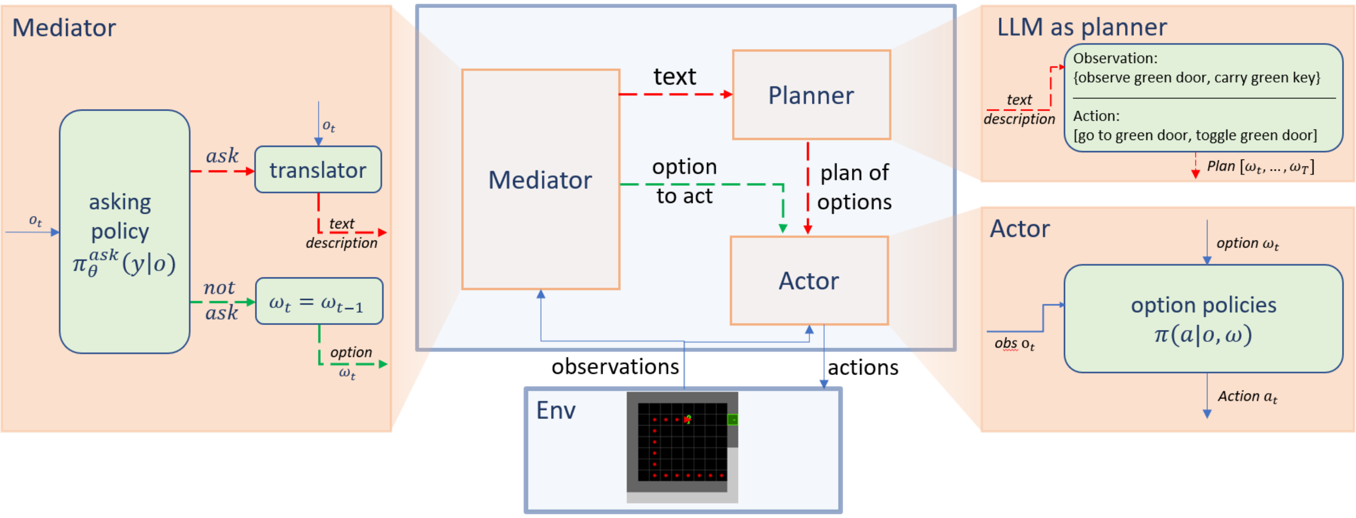


Fig. 2: An overview of the Planner-Actor-Mediator paradigm and an example of the interactions. At each time step, the mediator takes the observation o_t as input and decides whether to ask the LLM planner for new instructions or not. When the asking policy decides to *ask*, as demonstrated with a red 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.

C. Related work

LLMs have proven to be powerful tools for generating plans, and previous research has focused on designing the interface between the planner and the actor to facilitate their effectiveness. In [7], the LLMs are deployed for planning the entire sequence of options at the beginning of each task, allowing the agent to complete the task without further interaction with the planner. In [12], the authors introduce a feedback system where the agent would ask the LLM to make an updated plan according to the environment feedback when it fails to execute the previous plan. This approach makes the acting agent more robust to uncertainties in the environment. However, these methods often depend on hard-coded failure detectors, such as imposing a threshold to restrict the number of permissible MDP state-transition timesteps for an option.

In parallel with our work, Dasgupta et al. propose a Planner-Actor-Reporter framework that incorporates a reporter module to facilitate information exchange between the actor and the LLM-based planner [10]. In this framework, the agent interacts with the LLM at each timestep, regardless of whether it acquires new information or not. Although this approach eliminates the necessity for hard-coded termination conditions and mitigates uncertainties during option execution, it consumes much unnecessary resources, particularly when employing a large-scale costly LLM as the planner.

In this paper, we propose an approach termed *When2Ask* that enables the agent to interact with the LLM in a more intelligent and cost-effective manner. With *When2Ask*, the agent queries the LLM only when necessary, thereby optimizing resource usage incurred by using the LLM.

III. Our Approach When2Ask

In this section, we introduce our approach, *When2Ask*, which empowers an embodied agent to interact with an LLM for assistance while avoid unnecessary interactions to reduce the cost incurred by querying an LLM. We design *When2Ask* based on the Planner-Actor-Mediator framework proposed by Dasgupta et al. [10]. 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.

A. 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. We introduce these components as follows, while refer readers to the experiment section, namely Section IV, for detailed implementation information regarding each component in our experiments.

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 [7], [18], we assume the availability of an expert translator here, while it is worth noting that the translator module can also be replaced with a learned model [10], [12].

B. 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 encompasses 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) [19]. The objective function for training the asking policy is defined as follows:

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

where $y_t \in \{\textit{Ask}, \textit{Not Ask}\}$ represents the decision made by the asking policy at time step t, r_t denotes the task reward obtained at time step t, and ω_t is the planned option provided by the LLM at time step 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" ($y_t == \textit{Not Ask}$), we set ω_t to be the plan from the previous time step ($\omega_t = \omega_{t-1}$). 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 $\pi^{\text{ask}}_{\theta}$.

IV. Experiments

To evaluate our approach, we compared it against baseline methods using two environments: MiniGrid [13] and Habitat [14]. In this section, we first introduce the baseline interaction methods in subsection IV-A. Then, we provide a detailed description of our experiments conducted on the MiniGrid environment in subsection IV-B. This includes information about the experimental setup, implementation details of our approach, and the results obtained. Following that, we present our experiments performed on the Habitat environment in subsection IV-C. Similar to the MiniGrid experiments, we describe the experimental setup, implementation details of our approach, and experiment results obtained.

A. 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 [12]. 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. We argue that, adopting such hard-coded termination rules, the agent cannot fully utilize newly acquired information during option execution. Additionally, these hard-coded rules may be vulnerable to uncertainties embedded in other components of the framework.

Always The agent queries the LLM planner at every timestep, ensuring that any newly acquired information is immediately relayed to the planner [10]. 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 [15], [16]. 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 for embodied sequential decision-making tasks. This comparison helps evaluate the effectiveness and advantages of incorporating a pre-trained LLM into the planning process.

B. MiniGrid Experiments

The MiniGrid environment [13] consists of a collection of 2D grid-world environments with goal-oriented tasks. In these

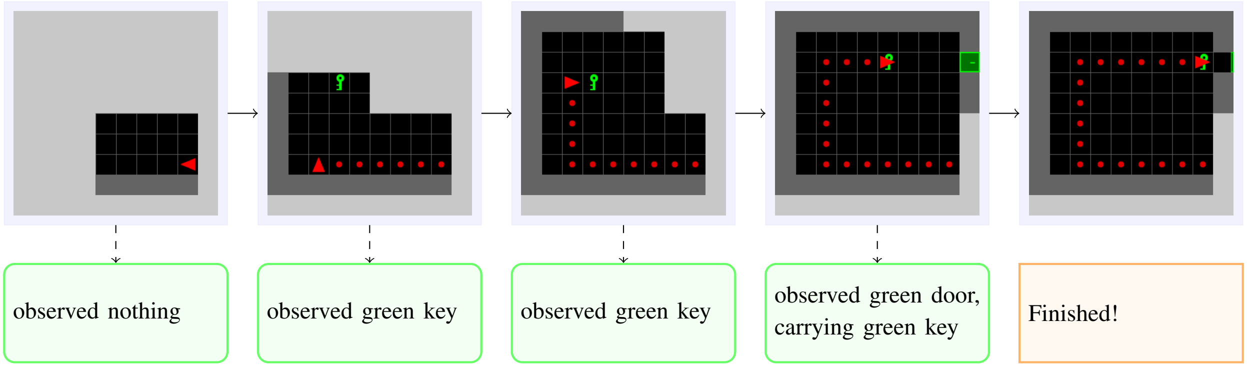


Fig. 3: An illustrative example of the partial observations and their corresponding text descriptions in environment *SimpleDoorKey*. The agent is illustrated with a red triangle, and the path it takes is illustrated with red dots. At the start of each episode, the agent is provided with only limited 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.

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”.

One important characteristic of our MiniGrid 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*.

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.

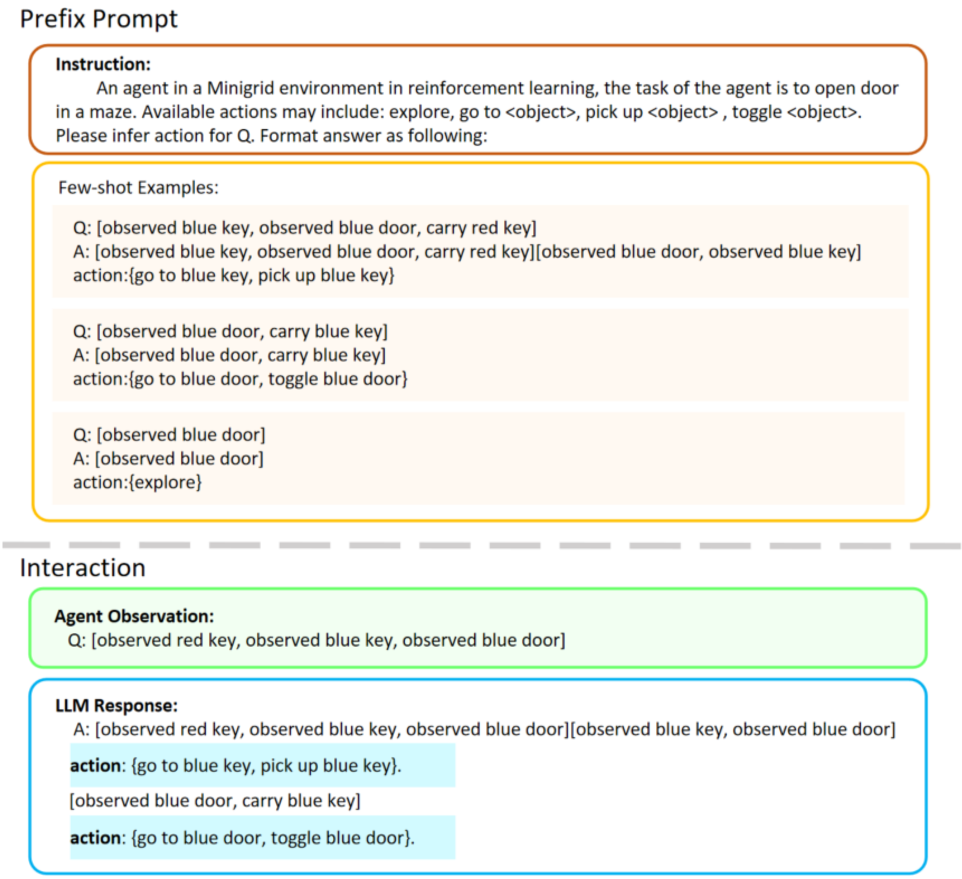
MovingObstacle adds another layer of complexity by introducing obstacles that move randomly within the room, potentially blocking the agent’s path. The agent needs to navigate in this dynamically changing environment and adapt its plans accordingly based on new observations.

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.

1) Implementation details of our approach on MiniGrid:

Planner As demonstrated in previous work [20], 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 ground the task knowledge and guide the LLM’s understanding of the specific task requirements. Furthermore, for the challenging reasoning task in the *ColoredDoorKey* environment, we utilize Chain-of-Thought prompts proposed by [6]. These prompts help the LLM to deal with complex reasoning tasks specific to the *ColoredDoorKey* environment. The few-shot examples provided in the prompts are used to anchor the agent’s knowledge about the task, such as understanding that a door can only be unlocked with a key of the same color, and provide constraints on the output formats and guidelines for generating appropriate plans. Note that the LLM planner must reason about the target task using its embedded knowledge and generalization capabilities to adapt to different scenarios with varying objects and colors. Fig. 4 provides an example of the prefix prompts and an interaction example in the *ColoredDoorKey* environment. It demonstrates how the LLM planner successfully generates a correct plan based on novel observations.

We employ two different versions of the Vicuna model (Vicuna-7b and Vicuna-13b) as LLM planners. The Vicuna models are part of a set of open-source LLMs trained by fine-tuning the LLaMa model [21]. We use the Vicuna-7b model for the *SimpleDoorKey*, *KeyInBox*, *RandomBoxKey*, and *MovingObstacle* environments, while for the more com-



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}.

Fig. 4: 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 task knowledge 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.

plex *ColoredDoorKey* environment we use the Vicuna-13b model. To enable interactions between the agent and the LLM planner, we design a communication application interface implemented using the fastapi framework in a RESTful API style. This interface allows for seamless interaction and information exchange between the agent and the LLM planner during the task execution. For more detailed information about our approach, implementation, and code, refer to our open-source code repository available at: https://github.com/ZJLAB-AMMI/LLM4RL.

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

- Explore: This option allows the agent to explore the environment, enabling it to gather information and uncover unexplored areas.
- 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.

Mediator As discussed in Section III-A, the mediator component consists of two distinct parts: an asking policy and a translator. In our experiments, we employ an expert translator and train a neural network to serve as the asking policy. Specifically, the asking policy 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.

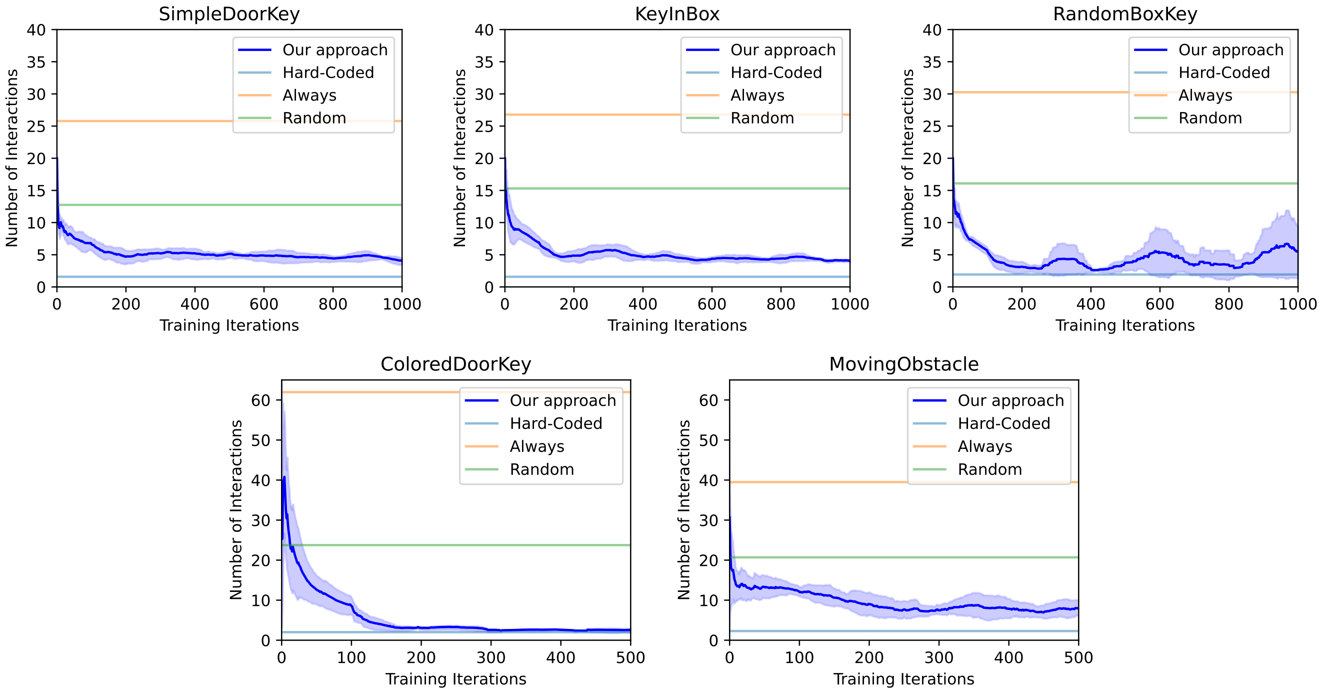


Fig. 5: The number of interactions with the LLM vs. the number of RL iterations used for learning the asking policy. It shows that, for every environment, the more thoroughly the asking policy is trained, the fewer interactions with the LLM planner (i.e., the less interaction costs) are required to complete a target task.

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.

As discussed in previous sections, we hypothesize that explicitly learning an asking policy within the planner-actor-mediator framework can benefit the agent in two ways: (1) avoiding wasting resources on non-informative interactions with LLM, and (2) improving task performance by interactively changing acting plans. With our designed experiments, we aim to investigate the following research questions:

2) Can our agent solve target tasks with less interaction costs: We compare our proposed approach *When2Ask* with several baseline methods to evaluate its effectiveness. We analyze the learning curves for both communication costs (Fig. 5) and task performances (Fig. 6) across all five environments. Additionally, we provide asymptotic performances in Table I. As is shown, our approach successfully reduces the number of interactions with the LLM while maintaining task performance across all environments. This reduction in communication cost indicates that our method effectively learns to minimize 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 interactions with the LLM to achieve successful task completion.

3) Can our agent actively ask 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 given time limit. In contrast, our proposed approach, 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*.

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 asking about the next steps 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 proposed approach in recognizing when to seek assistance from the LLM planner and making more efficient decisions based on the available information. By leveraging the embedded knowledge

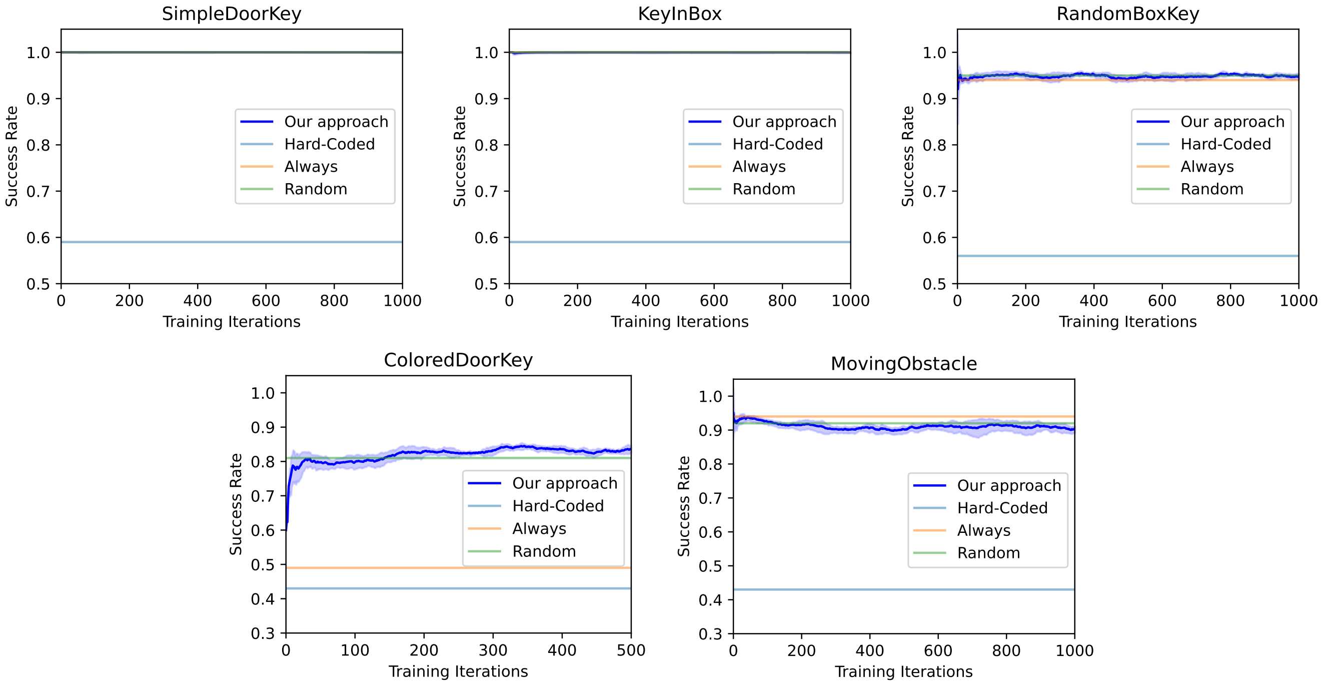


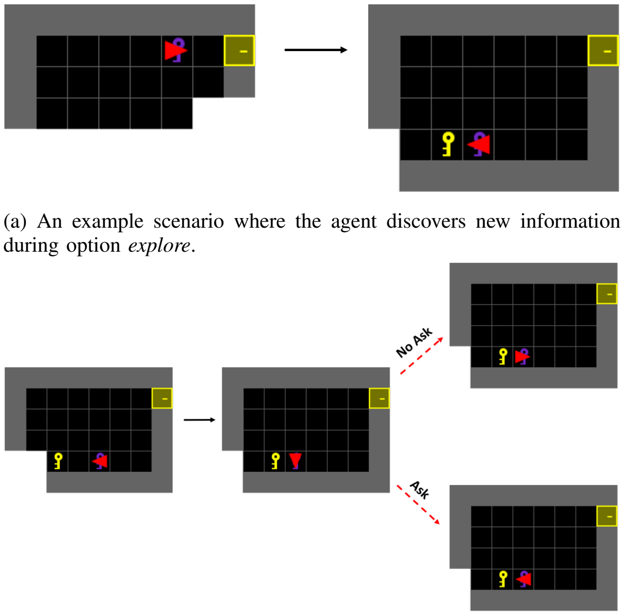
Fig. 6: Success rate of completing target tasks vs. the number of RL iterations used for learning the asking policy. It demonstrates that our approach consistently maintains a high success rate across all environments, and outperforms baseline methods in *ColoredDoorKey*.

TABLE I: 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%

of the LLM planner, our approach enables the agent to make informed choices that expedite task completion and improve overall performance.

4) Can our mediator perform robustly against uncertainties in other components: In the complex environment of *ColoredDoorKey*, 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 other components of the planner-Actor-Mediator framework have flaws. The learned asking policy enables the agent to make



(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.

Fig. 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.

more informed decisions based on its observations and context, leading to robust performance in scenarios where baseline methods may fail due to flaws of other components.

As depicted in Fig. 6, our proposed approach shows a gradual improvement in task success rates compared to the baseline methods. This trend indicates that the learned mediator within our approach has the capability to acquire knowledge about the behaviors of other components in the framework. As a result, the agent's performance becomes more robust, particularly in complex environments.

5) How does our agent perform compared to a baseline RL agent that does not use an LLM planner: To assess the importance of the reasoning ability of the LLM in our approach, we conduct an ablation study comparing it with a baseline RL agent that does not use an LLM planner. The RL baseline focuses on learning the planner, specifically policy-over-options, without any interaction with an LLM. The summarized results displayed in Table II demonstrate that even in the simplest environment, *SimpleDoorKey*, the RL baseline faces challenges in completing the task within a fixed number of training iterations. This suggests that learning how to solve these tasks from scratch is difficult for an RL agent. In embodied environments like these, agents must acquire skills such as exploration, reasoning about relationships between objects, and planning optimal actions to accomplish tasks successfully. By incorporating the LLM's assistance, an agent can leverage the world knowledge embedded in the language model, leading to a significant reduction in the difficulties associated with solving these tasks. Consequently, the outcomes of the ablation study support the notion that the reasoning ability provided by the pre-trained LLM plays a crucial role in achieving higher performance in complex environments.

TABLE II: Performance comparison between our agent and an RL agent that does not use LLM in the *SimpleDoorKey* environment.

Performance metric	RL	Our approach
Average Return ↑	0.0324	**0.7583**
Average # of state-transition timesteps ↓	98.36	**30.47**
Success rate ↑	12%	**100%**

C. Habitat Experiments

We further evaluate our approach with the Habitat environment [14]. Habitat is a simulation platform specifically designed for end-to-end development of embodied AI. 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. This task is depicted in Fig. 10.

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 [14].

In the *Pick&Place* environment, as solving the task requires progressively achieving several sub-goals, 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.

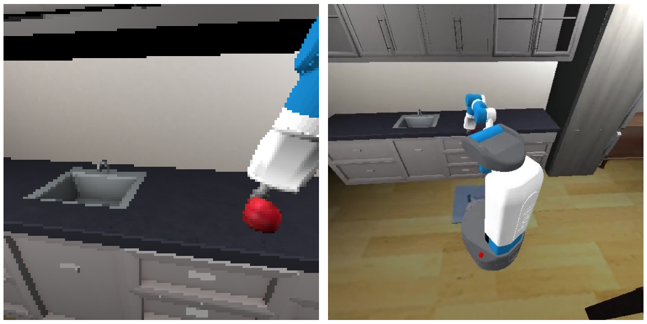
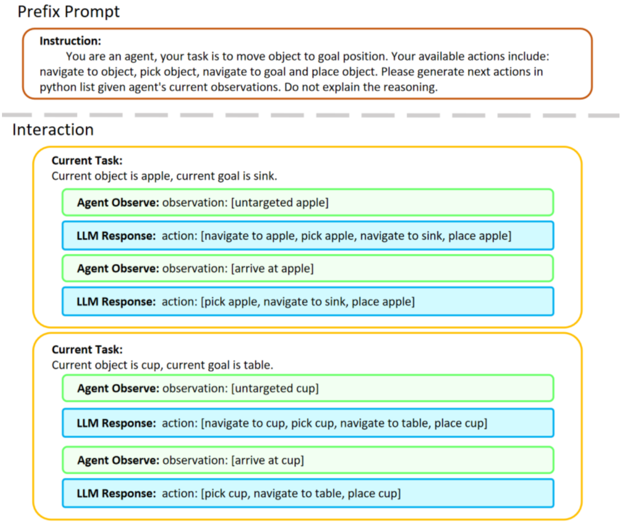


Fig. 8: Habitat environment. Left: The visual observation from the on board camera. Right: A view of acting robot and its workspace from a third-party camera. Note that the third-party camera mentioned is purely for illustrative purposes and is not utilized during either the training or testing phases.



Prefix Prompt

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.

Interaction

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]

Fig. 9: An example of the prompts and interactions for the habitat environment. Prefix prompt only contains a short task instruction.

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. 9.

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.

TABLE III: 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

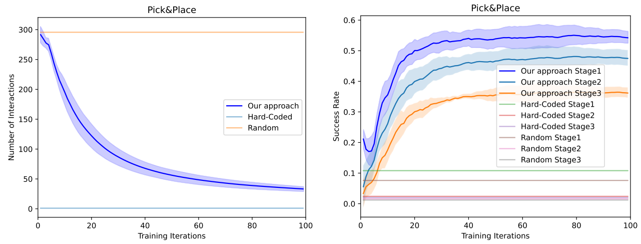


Fig. 10: The number of interactions with the LLM (left) and the stage success rates (right) vs. the number of training iterations used for learning the asking policy on the *Pick&Place* task.

Mediator In the Habitat environment, visual observations obtained from the on-board camera are utilized as inputs. To aid in the comprehension of these visual inputs by LLMs, we employ an expert translator that generates natural language descriptions listing the objects captured by the camera. Alternatively, pre-trained image translation models such as CLIP [22] can also be used for this purpose.

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*.

2) Experiment results on the Habitat experiments: 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 III and Fig. 10, our approach significantly outperforms baseline methods across all three stages. Particularly, compared to the hard-coded baseline where the preset plan is executed step-by-step, our approach addresses the "hand-off problem" [14] 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. 11, 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 incorporating intelligent interactions with the LLM planner, enabling the agent to adapt its actions based on dynamic information provided by the planner.

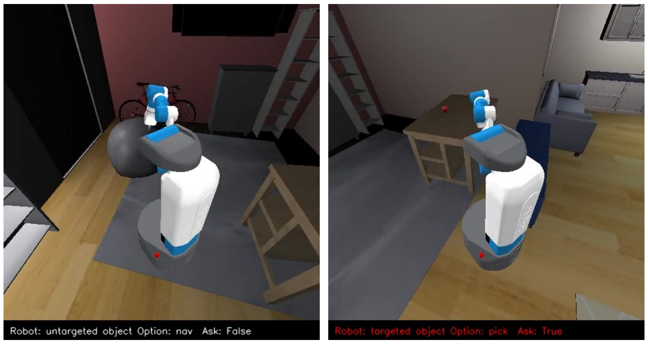

Robot: untargeted object Option: nav Ask: False
Robot: targeted object Option: pick Ask: True

Fig. 11: 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.

The obtained results demonstrate that the RL learned asking policy effectively establishes a connection between the world knowledge embedded within the LLMs and the local knowledge embedded within the pre-trained skills. 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 in the MiniGrid environments, particularly in the *ColoredDoorKey* scenario, where the RL learned asking policy enables the agent to outperform all baselines.

V. Conclusions and Future Works

In this paper, we aim to enhance the efficiency and cost-effectiveness of the interaction between an agent and an LLM in embodied environments. We consider cases wherein an LLM model is available while interacting with it is costly. We propose an RL-based mediator model within the Planner-Actor-Mediator framework [10]. Our model enables the agent to interact with the LLM in a more intelligent way than the baseline strategies. We evaluate our approach with both MiniGrid and Habitat environments. Results demonstrate that, with our approach, the agent can explore the environment and respond to perceived new information in a more reasonable way. Specifically, it learns when to initiate or maintain interaction with the LLM and when to rely on its own learned skills without requiring LLM interaction. Furthermore, we found that the agent exhibits greater robustness by maintaining only a few necessary interactions with the LLM, compared to frequent and intensive interactions. To summarize, our approach provides a cost-effective means to bridge the gap between world knowledge and task-specific knowledge by enabling the agent to interact with a pre-trained LLM to obtain valuable instructions.

In future work, one potential direction is to develop a fully learned mediator that serves as an optimal interface between LLMs and actors. This could involve training a translator specifically designed to provide the most accurate and informative text descriptions for LLMs. Additionally, while our current framework primarily leverages the reasoning abilities of LLMs, recent research has showcased their remarkable capabilities in other aspects, such as memorization, summarization, and few-shot learning [17], [23]. Exploring how to effectively utilize these attractive features of LLMs represents another exciting avenue for future exploration. For instance, investigating how LLMs can summarize an agent’s past experiences and contribute to the long-term credit assignment problem in RL [24] would be particularly interesting. One can also extend our approach to non-embodied environments, such as the fine-grained visual learning case considered in [25], where LLMs can be leveraged for commonsense knowledge reasoning.

References

[1] M. Deitke, D. Batra, Y. Bisk, T. Campari, A. X. Chang, D. S. Chaplot, C. Chen, C. P. D’Arpino, K. Ehsani, A. Farhadi *et al.*, “Retrospectives on the embodied ai workshop,” *arXiv preprint arXiv:2210.06849*, 2022.

[2] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, “Embodied question answering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1–10.

[3] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio, “Babyai: A platform to study the sample efficiency of grounded language learning,” *arXiv preprint arXiv:1810.08272*, 2018.

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

[5] 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.

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

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

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

[9] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, “Vima: General robot manipulation with multimodal prompts,” *arXiv preprint arXiv:2210.03094*, 2022.

[10] I. Dasgupta, C. Kaeser-Chen, K. Marino, A. Ahuja, S. Babayan, F. Hill, and R. Fergus, “Collaborating with language models for embodied reasoning,” *arXiv preprint arXiv:2302.00763*, 2023.

[11] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, “Voyager: An open-ended embodied agent with large language models,” *arXiv preprint arXiv:2305.16291*, 2023.

[12] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, “Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents,” *arXiv preprint arXiv:2302.01560*, 2023.

[13] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry, “Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks,” *CoRR*, vol. abs/2306.13831, 2023.

[14] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets *et al.*, “Habitat 2.0: Training home assistants to rearrange their habitat,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 251–266, 2021.

[15] R. S. Sutton, D. Precup, and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,” *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[16] D. Precup, *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[17] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, “A survey on large language model based autonomous agents,” *arXiv preprint arXiv:2308.11432*, 2023.

[18] T. Carta, C. Romac, T. Wolf, S. Lamprier, O. Sigaud, and P.-Y. Oudeyer, “Grounding large language models in interactive environments with online reinforcement learning,” *arXiv preprint arXiv:2302.02662*, 2023.

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

[20] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, “Rethinking the role of demonstrations: What makes in-context learning work?” *arXiv preprint arXiv:2202.12837*, 2022.

[21] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.

[22] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, “Learning transferable visual models from natural language supervision,” in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[23] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, “A survey of large language models,” *arXiv preprint arXiv:2303.18223*, 2023.

[24] A. Meulemans, S. Schug, S. Kobayashi, N. Daw, and G. Wayne, “Would i have gotten that reward? long-term credit assignment by counterfactual contribution analysis,” *arXiv preprint arXiv:2306.16803*, 2023.

[25] P. Zhang and B. Liu, “Commonsense knowledge assisted deep learning with application to size-related fine-grained object detection,” *arXiv preprint arXiv:2303.09026*, 2023.