

Large Language Model is a Good Policy Teacher for Training Reinforcement Learning Agents

Zihao Zhou^{1,*}, Bin Hu^{1,*}, Pu Zhang¹, Chenyang Zhao¹ and Bin Liu^{1,†}

¹Research Center for Applied Mathematics and Machine Intelligence, Zhejiang Lab
 {zhouzihao,hubin,puz,c.zhao,liubin}@zhejianglab.com

Abstract

Recent studies have shown that Large Language Models (LLMs) can be utilized for solving complex sequential decision-making tasks by providing high-level instructions. However, LLM-based agents face limitations in real-time dynamic environments due to their lack of specialization in solving specific target problems. Moreover, the deployment of such LLM-based agents is both costly and time-consuming in practical scenarios. In this paper, we introduce a novel framework that addresses these challenges by training a smaller scale specialized student agent using instructions from an LLM-based teacher agent. By leveraging guided actions provided by the teachers, the prior knowledge of the LLM is distilled into the local student model. Consequently, the student agent can be trained with significantly less data. Furthermore, subsequent training with environment feedback empowers the student agents to surpass the capabilities of their teachers. We conducted experiments on three challenging MiniGrid environments to evaluate the effectiveness of our framework. The results demonstrate that our approach enhances sample efficiency and achieves superior performance compared to baseline methods.

1 Introduction

Large Language Models (LLMs) have revolutionized the field of artificial intelligence. These models undergo training using vast amounts of text data from the internet, enabling them to exhibit remarkable capabilities such as natural language generation, question answering, and translation Brown *et al.* [2020]; Du *et al.* [2022]; Chiang *et al.* [2023]. Recent advancements have highlighted the LLM’s potential in capturing diverse aspects of the physical laws that govern our world, including spatial relationships Patel and Pavlick [2021], color perception Abdou *et al.* [2021], and even the affordances between entities and objects Ahn *et al.* [2022]. Furthermore, previous research has demonstrated the reasoning

abilities possessed by LLMs Radford *et al.* [2019]; Brown *et al.* [2020]; Wei *et al.* [2022]. By leveraging this prior knowledge, it becomes possible to propose effective action plans for embodied agents.

However, LLM-based agents struggle to generate end-to-end actions in interactive environments due to their lack of experience in real-world dynamic scenarios. This limitation arises from two key factors. Firstly, LLMs do not possess the appropriate task incentives during the training process. Secondly, these models lack the capability to actively interact with the environment and gather real-time data Carta *et al.* [2023]. A common solution to address this issue is to update the LLM-based agent with reinforcement learning (RL), where the agent interacts with the environment and learns from feedback. Moreover, the deployment of LLMs typically necessitates significant memory and computational resources. These requirements make the utilization of these models impractical and costly, particularly when considering their deployment on offline edge devices. As a result, employing a compact yet specialized deep neural network agent that possesses the capabilities of LLMs becomes more practical and suitable for real-world applications.

In this paper, we address the challenges associated with utilizing LLMs in dynamic environments. Specifically, we present a novel approach called LLM-based policy distillation framework that leverages the vast knowledge embedded within LLMs to expedite the training process of a specialized small-scale student agent for the target task. This framework not only incorporates guidelines from the LLM teacher but also enables the student agent to learn from online interactions with the environment. By doing so, the student agent can identify and correct any mistakes made by the LLM teacher, resulting in performance that surpasses that of its teacher. As a result, the locally trained student agent becomes independent of its LLM teacher and can be deployed on lightweight devices.

We evaluate the effectiveness of our approach using three partially observable MiniGrid environments Chevalier-Boisvert *et al.* [2018], which involve the agent exploring the environment and reacting to newly acquired information. Through these experiments, we demonstrate that our approach successfully leverages the prior knowledge contained within LLMs and achieves superior sample efficiency compared to tabula rasa reinforcement learning. Importantly,

*Equal contribution

†Corresponding author (Email:liubin@zhejianglab.com)

when the strategies provided by LLMs contain errors, our framework enables the student agent to rectify these errors and yield improved performance outcomes.

To summarize, our main contributions include:

- To overcome the limitations of LLMs in handling dynamic environment sequential decision-making problems, we propose a policy distillation strategy. In this approach, an LLM-based agent acts as the teacher, while a student agent is introduced, which learns from both the teacher agent and the feedback received from the environment simultaneously.
- We introduce an implementation of our strategy that includes a novel loss design, which dynamically balances the instructions from the teacher and the feedback from the environment. In our approach, we also consider the uncertainties associated with the instructions provided by the LLM teacher.
- We evaluate the effectiveness of our approach by conducting experiments on three distinct MiniGrid environments. The results demonstrate that leveraging the LLM-based teacher significantly accelerates the training process of the student agent. Furthermore, by incorporating feedback from the environment, the student agent is capable of rectifying the mistakes made by the teacher and ultimately surpassing its performance.

2 Related Work

In this paper, we consider an algorithmic agent operating in a dynamic environment. The agent is required to make a series of decisions and take actions based on the current state of the environment, employing a specific policy to successfully complete a designated task. We provide a brief overview of relevant research related to sequential decision making in dynamic embodied environments.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a widely-used algorithmic framework designed for sequential decision making in dynamic environments. In RL, we consider a Markov Decision Process (MDP) denoted by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where \mathcal{S} and \mathcal{A} denote the state and action spaces, respectively. The transition probability function is denoted as $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(\mathcal{S})$, and the reward function is denoted as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. Additionally, γ represents the discount factor. The primary objective of RL is to learn an optimal policy $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, which maximizes the expected cumulative return over time: $\max_{\pi} \mathbb{E}[\sum_t \gamma^t r_t]$. The parameter of the policy π is denoted as θ . A standard gradient-based RL algorithm minimizes a surrogate loss, $\mathcal{L}_{RL}(\theta)$, using gradient descent with respect to θ . This loss is estimated using sampled trajectories, where each trajectory consists of a sequence of environment states and agent actions.

In the RL literature, a closely related work to ours is “Kickstarting RL” Schmitt *et al.* [2018]. In this work, teacher agents are initially pre-trained and then used to train a new student agent. The goal is for the student agent to mimic the behavior of the teacher agent to improve sampling efficiency. However, unlike Kickstarting RL, our approach does

not require any pre-trained teacher agent specifically tailored to the target task. Instead, we directly leverage an LLM as the teacher agent. In other words, our approach can be viewed as an enhanced and more versatile version of Kickstarting RL that harnesses the world knowledge encoded in the LLM to teach the student agent to accomplish a specific target task.

2.2 LLM based Agents

LLMs have demonstrated remarkable reasoning capabilities, leading researchers to utilize them as foundational components for building LLM-based agents in various daily decision-making tasks Xi *et al.* [2023]; Yang *et al.* [2023]; Wang *et al.* [2023]. In addition, studies such as Huang *et al.* [2022] have shown that LLMs can generate high-level plans when prompted with natural language descriptions of the scene. However, since pre-trained LLMs lack grounding in specific target problems, they may propose actions that are incompatible with the acting agent or the environment. To address this limitation, Ahn *et al.* [2022] introduced the concept of grounding LLMs through an affordance function of pre-trained skills, which helps the LLMs develop feasible plans for the agents to execute. Furthermore, Carta *et al.* [2023] proposed an approach where the agent interacts with the environment and fine-tunes the LLMs using online collected data, enabling better adaptation to the specific task at hand.

While LLM-based agents have shown promising performances in various target tasks, they typically require significant computational resources in practical applications. To mitigate this high-cost issue, Hu *et al.* [2023] trains an individual module using RL to determine when to query LLMs, thereby avoiding unnecessary LLM queries and minimizing resource wastage. Additionally, Nottingham *et al.* [2023] optimizes the selection of information presented to LLMs, reducing the length of input contexts. Although these methods reduce the total number of queries and input tokens per query, they still necessitate online LLM queries during test time. Unlike the aforementioned approaches, our method focuses on distilling the task-relevant knowledge embedded within the LLM into a smaller and specialized policy model, specifically the student agent. This means that during testing, only the lightweight student agent needs to be deployed, eliminating the need for online LLM queries. As a result, our approach enables efficient and independent deployment of the student agent without relying on continuous access to the LLM.

2.3 LLM Assisted RL

Some research has explored leveraging the general knowledge embedded in LLMs to assist the standard RL process. For instance, studies such as Kwon *et al.* [2023] and Klissarov *et al.* [2023] tackle challenging scenarios where describing desired behaviors with a simple reward function is difficult. They employ LLMs to aid in reward assignment. Specifically, Kwon *et al.* [2023] uses LLMs as proxy reward functions to automatically annotate trajectory data with rewards. On the other hand, Klissarov *et al.* [2023] leverages an offline dataset of behaviors and utilizes LLMs’ preferences over pairs of randomly sampled trajectories to construct a reward model. Additionally, Du *et al.* [2023] and Colas *et al.*

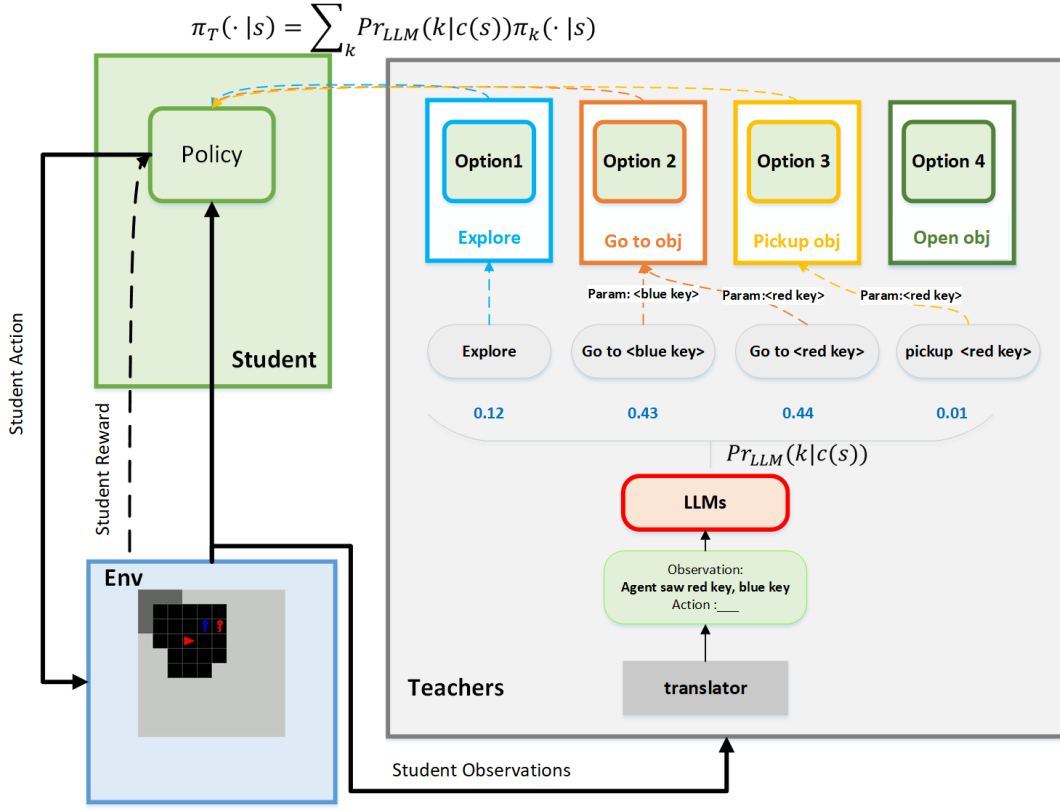


Figure 1: An overview of our approach using an embodied task with a MiniGrid environment Chevalier-Boisvert *et al.* [2018] as an example. For each sampled state, the LLM-based agent provides the student agent with a distribution over a set of suggested actions. The student agent is trained to optimize two objectives concurrently. The first objective is to maximize the expected return, following a traditional RL algorithm. The second objective is to adhere to the teacher’s instructions. Over time, the weight assigned to the second objective gradually decreases, mirroring the student’s increasing expertise and diminishing reliance on the teacher.

[2023] focus on learning various behaviors without relying on reward supervision. They use LLMs to generate innovative goals during exploration in the environment.

In contrast to these previous works, our approach concentrates on leveraging prior knowledge about the target task to enhance the initial exploration stage of an RL agent. This enables us to train the policy model with significantly less data, making the learning process more efficient.

3 Our Approach: LLM4Teach

Here we present our approach termed LLM for policy teaching, or LLM4Teach in short. The fundamental concept of our approach is to utilize a pre-trained LLM as a teacher agent, which then instructs a lightweight student agent to rapidly acquire a policy for real-time decision-making in order to achieve a specific embodied task. We enable the student agent to interact with the environment and receive feedback from these interactions to correct any errors indicated by the teacher agent. Once the training phase is ended, only the lightweight student agent is deployed during the testing phase, yet its ability to accomplish the target task surpasses that of its teacher agent.

The conceptual framework of our approach is illustrated in Figure 1. We describe the details from the perspective of the

student agent in the following subsections.

3.1 Requesting instructions from the teacher

Following Ahn *et al.* [2022], we assume that the teacher agent is equipped with a set of K fundamental option policies $\Pi : \{\pi_k : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})\}$, where $k \in \{1, 2, \dots, K\}$ denotes the option index. Given a state s , the student agent ask the LLM based teacher agent for instructions. The teacher agent first samples a high-level option π_k from the set Π , prompted by a textual description of the state $c(s)$. Then a valid action $a \sim \pi_k(s)$ is generated according to the selected option as the instruction provided by the teacher.

Generating uncertainty-aware soft instructions using LLM In principle, the policy learning from the teacher agent can be taken as a knowledge distillation process. As revealed in Hinton *et al.* [2015], taking account uncertainty in knowledge distillation is beneficial for improving sample efficiency and avoiding severe over-fitting Hinton *et al.* [2015]. Therefore, we consider three ways to let the LLM provide uncertainty-aware soft instructions to the student agent.

Specifically speaking, when the student agent sends an observation of the state s to the teacher agent, the teacher agent provides a soft decision or instruction $\pi_T(\cdot | s)$ in the follow-

ing manner:

$$\pi_T(\cdot|s) = \sum_k Pr_{LLM}(k|c(s))\pi_k(\cdot|s), \quad (1)$$

where $Pr_{LLM}(k|c(s))$ represents the probability of the LLM teacher selecting the k th option given the textual description $c(s)$ of the current state s , and $\pi_k(\cdot|s)$ denotes the policy associated with the k th option.

We consider three methods for calculating $Pr_{LLM}(k|c(s))$:

- **Statistical estimates:** Similar to Wang *et al.* [2022], we query the LLM multiple times with the same prompt to estimate the probability of each decision.
- **Prompt design:** By requesting the LLM to output the probability associated with its decision through prompt design.
- **Softmax of logits:** We calculate the logits of the words relevant to the strategy in the LLM’s output (such as ‘explore’ or ‘pick’) Ahn *et al.* [2022] and then apply the Softmax function to obtain the probabilities.

Compared to the **Softmax of logits** method, which requires calculating the logits word by word, the first and second methods are relatively easier to implement. In our experiments, we mainly utilized the **Statistical Estimates** approach. For more specific details, please refer to Appendix A.4.

3.2 Learning from the teacher’s instructions and the environmental feedbacks

To accelerate the stage of initial exploration stage, we kickstart the training of the student agent with the guidance from the teacher. Specifically, with sampled trajectory data and the corresponding teacher instructions, the student agent is trained to minimize the following loss:

$$\mathcal{L}(\theta) = \mathcal{L}_{RL}(\theta) + \lambda \mathbb{E}_{s \sim \pi_\theta} \mathcal{H}(\pi_T(\cdot|s) || \pi_\theta(\cdot|s)), \quad (2)$$

where λ is the temperature that controls the weighting of environmental feedback and teacher instructions. Intuitively, after the student has learned sufficient knowledge from the teacher, it should pay more attention to the environment feedback over the teacher’s. To achieve this, we linearly decay the temperature λ , encouraging the student agent to learn from RL loss and deviate from the teacher agent as the training proceeds. Eventually, λ decays to 0, which means that teacher instructions are no longer needed. Removing the teacher instructions also helps us to avoid unnecessary cost induced by communication overhead with the LLM. In practice, we found that it helps to keep the coefficient as a small constant value λ_c for a few iterations before completely removing it, in order to stabilize the training. In summary, our kickstarting coefficient schedule is designed as follow:

$$\lambda_i = \begin{cases} \lambda_0 - ki & \text{if } i < i_1 \\ \lambda_c & \text{if } i_1 < i < i_2 \\ 0 & \text{otherwise} \end{cases}, \quad (3)$$

where i is the training iteration, k controls the speed of the decay, λ_0 is the initial kickstarting temperature, and i_1, i_2 indicates when to switch to the constant temperature and remove teacher instructions, respectively.

Algorithm 1 Learning from LLM-based teacher

Require: LLM, pre-trained option policies $\{\pi_k\}$

Initialize student agent θ

```

1: for  $i = 1, 2, \dots, \text{MaxIterNum}$  do
2:   Collecting rollouts following student policy and stores
   in buffer  $\mathcal{D}$ 
3:   for each transition  $(s, a, r) \in \mathcal{D}$  do
4:      $\omega_k = LLM(s, k)$ 
5:      $\pi_T(\cdot|s) = \sum_k \omega_k(s) \pi_k(\cdot|s)$ 
6:   end for
7:    $\lambda_i = \text{scheduler}(i)$ 
8:   for each gradient step do
9:      $\theta \leftarrow \theta - \alpha \nabla_\theta (\mathcal{L}_{RL} + \lambda_i \mathcal{L}_{KS})$ 
10:  end for
11: end for
```

4 Experiments

In this section, we present implementation details of our method in the context of our experiments. In our experiments, we designed 3 partially observable environments, based on MiniGrid Chevalier-Boisvert *et al.* [2018]. These environments require the capability to perceive, reason, and control in order to solve target tasks. To begin with, we introduce the testing environments. Then we present the implementation details of our method. Finally, we present the baseline methods involved in our experiments.

4.1 Environments

In MiniGrid environments, an agent must navigate in a 2D grid room and interact with specific objects, in order to complete different tasks, such as “*open the red door*” or “*put the green ball next to the yellow box*”. The agent is equipped with only a limited view range, such that it also needs to first explore the environment and collect useful information for further planning. In this work, we consider 3 distinct environments: *SimpleDoorKey*, *LavaDoorKey* and *ColoredDoorKey*. **SimpleDoorKey** The room contains only one key and one exit door. The objective for the agent is to effectively explore the surroundings, locate both the target door and the key, pick up the key, and ultimately unlock the designated door. This is the most basic environment, where all expert option policies are generated.

LavaDoorKey Some floor grids are filled with hazardous lava. Stepping into any lava grid will lead to an immediately failure of the task. Therefore, the agent needs to transfer the knowledge from the basic environment and learn to generalize to this new task from environmental feedback.

ColoredDoorKey The room encompasses multiple keys and only one exit door. Each key and the corresponding door are color-coded, necessitating the use of a key that matches the door’s color for successful unlocking. Due to the complexity of this task and the introduction of the new rule, LLMs may fail to ground to provide erroneous strategies.

All environments are procedurally generated, with the grid layout (such as room size, key, and door locations) being randomized upon each environment reset. Additionally, a separate test set comprising 1000 randomly selected seeds is re-

Prefix Prompt

Instruction:

In a locked 2D grid room, there is an agent whose task is to open the door. The door can only be opened with a key of the same color as the door. The agent can perform the following actions: explore, go to an object, pick up an object, drop an object, or open an object. Agent can only hold one object. Please clarify which object the agent has seen and is holding given the information provided. Then, please determine what the agent should do in its current state. Finally, please select the correct agent action. Do not provide reasoning.

One-shot Example:

User: Agent sees box, holds nothing

Assistant:

1. What agent sees and holds: Agent sees a box, holds nothing. Key and door are not observed yet.
2. What should agent do: Key maybe hidden in the box, agent may go to box and open it to check.
3. Action: {go to <box>, open <box>}.

Interaction

Observation: Agent sees red key, red door, holds blue key.

Response:

1. What agent sees and holds: Agent sees a red key and a red door, and holds a blue key.
2. What should the agent do: Agent should drop the blue key and pick up the red key to open the red door.
3. Actions: {drop <blue key>, go to <red key>, pick up <red key>}

Figure 2: Example of the prefix prompt and one interaction for task *ColoredDoorKey*. The Prefix prompt consists of two blocks: the instruction block briefly introduces the target problem and the CoT reasoning process; and the example block provides one arbitrary example of the response format.

served for evaluation purposes in each environment.

4.2 Implementation Details

LLM-based teacher We use ChatGLM-turbo Du *et al.* [2022] as the LLM and use Chain-of-thought (CoT) Wei *et al.* [2022] style prompts to empower our teacher agent with complex reasoning capabilities. More specifically, we ask the LLM to firstly summarize the scene; secondly reason about what to do next; and finally output its decision. Additionally, we provide one arbitrary example in the prompt to help it understand the reasoning process and the correct output format. Figure 2 provides an example of the dialogues generated by the LLM, in the task of *ColoredDoorKey*. In the case when the LLM returns with a decision that is not executable, a "wait" action is put in place.

In our experiments, we record the LLM response strategies given different state descriptions throughout the training process, and use the statistical estimates for the uncertainty of LLM’s decisions.

Option policies For all tasks, we implement 5 human expert option policies, including {*explore*, *go to an object*, *pickup an object*, *drop an object*, *open an object*}. These option policies are gathered within the basic *SimpleDoorKey* task. Each option returns a Dirac delta distribution over

the actions given a state. More details are included in Appendix A.1.

Kickstarting coefficient schedule To find the best scheduler for λ , we set $\lambda_0 = 10$ and sweep over 6 different combination of k and i_2 . In the main experiment results, we report the combination with the best performances: $k = 0.01$ and $i_2 = 3000$.

4.3 Baselines

Tabula rasa RL We compare our approach with an RL baseline learned from scratch. Specifically, we train a policy model with the same architecture as our student agent, using RL losses only. With this comparison, we aim to demonstrate the effectiveness of introducing the LLM-based teacher agent in addressing exploration difficulties and improving overall sample efficiency.

LLM-based agent We also compare the performances of our student agent with its LLM-based teacher agent, in order to investigate whether the proposed framework can help the student agents to learn capabilities beyond their teachers.

5 Experiment results

5.1 Does the LLM-based teacher help in solve complex tasks efficiently?

We compare our approach with an RL baseline to investigate the importance of the reasoning ability of the pre-trained LLMs. As summarized in Figure 3, even in the simplest environment, *SimpleDoorKey*, the RL baseline struggles to complete the task using a fixed number of training iterations. This suggests that it is challenging for an RL agent to learn to solve these tasks from scratch.

In these embodied environments, the agent needs to learn how to explore the environment, reason about the relationships between different objects, and plan optimal actions to complete tasks. With the help of the LLMs, an agent can take advantage of the world knowledge embedded in the LLMs to significantly reduce the difficulties in solving these tasks. Therefore, the results suggest that the reasoning ability of the pre-trained LLMs is crucial for achieving higher performance in complex environments.

5.2 Does the student agent learn beyond the capability of the teacher agent?

As described in previous section, it is important for the student agent to learn from the environment alongside with the teacher instructions, in order to surpass the capability of its teacher. In this section, with environment *ColoredDoorKey* and *LavaDoorKey*, we demonstrate that with our approach, the student agent can learn corrected behaviors and generalize to previously unseen entities, respectively.

Correcting teacher’s mistakes In *ColoredDoorKey*, LLM-based agents may suggest inefficient or incorrect options, because they are not well grounded under the target environment and expert option policies. For example, when the agent describes the current state as "saw a red key, held a blue key", the LLM would suggest "drop the blue key", despite "explore" being the de facto best choice. As a result, the LLM-based teacher agent achieves only an average return

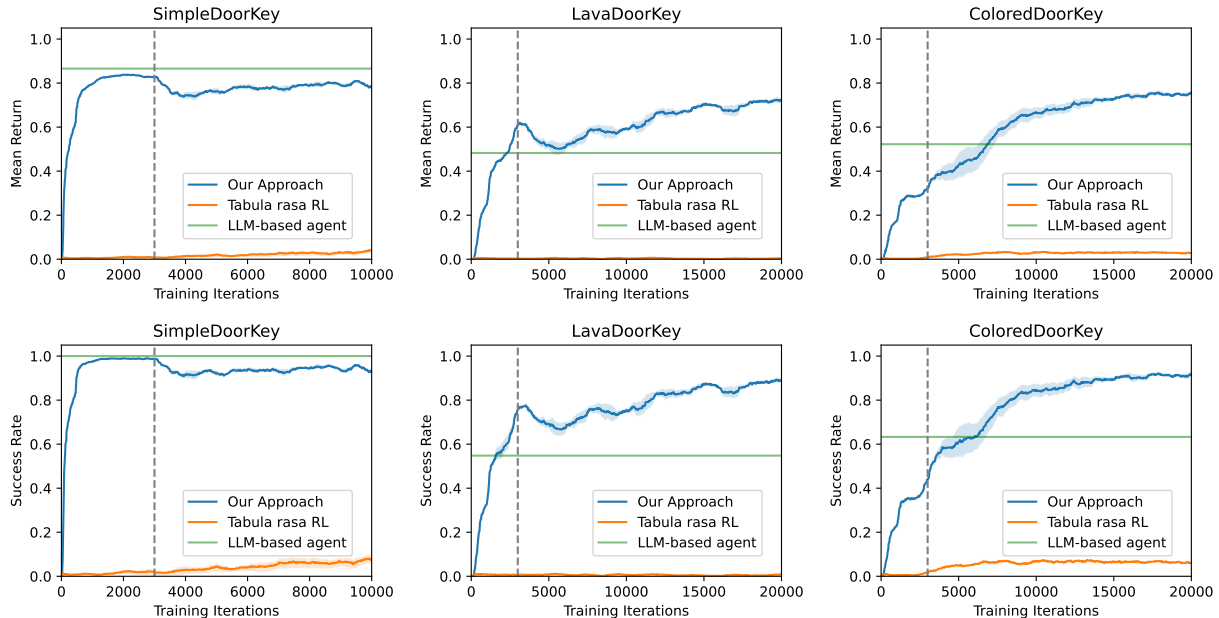


Figure 3: The average returns (top row) and success rates (bottom row) during training of all compared methods across three environments. The dotted vertical line indicates the point at which the teacher policy is removed, i.e., $\lambda_i = 0$. With our approach, the student agent can effectively leverage the knowledge of LLMs-based teacher to bootstrap the early learning stage. In *LavaDoorKey* and *ColoredDoorKey*, the student agent learns to outperform LLM-based agent by learning from environment feedback with an RL loss.

of 0.52. As shown in the right column of fig. 3, with our approach, the student agent is able to learn from environmental feedback and resolve such issues, resulting in a significant improvement in performance. The student achieves an average asymptotic return of 0.77.

Generalizing to unseen scenarios In *LavaDoorKey*, as the expert option policies are gathered in the *SimpleDoorKey* environment, where there is no lava grid. The student agent needs to generalize to the new environment, escaping the locked room while avoiding the lava grids. As shown in the middle column of fig. 3, the student policy in our approach can extract the useful knowledge from the teacher and rapidly adapt itself in previously unseen scenarios. Asymptotically, our student agent achieves a success rate of 0.87, compared with 0.55 of an LLM-based agent.

6 Conclusions

For real-time sequential decision-making problems in dynamic environments, an intelligent agent based on LLMs inherits the drawbacks of the LLM itself, such as hallucinations, high deployment costs, and low communication efficiency. On the other hand, RL, as the most popular paradigm for handling sequential decision-making problems, faces limitations such as low exploration efficiency and high sample complexity, particularly in scenarios with sparse rewards. To address these challenges, we propose a novel computational paradigm where an LLM serves as the policy teacher for RL agents.

By incorporating the LLM as a teacher, our approach, LLM4Teach, bypasses the issues of low exploration efficiency and high sample complexity. The RL based student agent interacts with the environment and collects environ-

mental feedback, enabling it to rectify mistakes made by the LLM-based teacher agent. This integration allows us to leverage the respective advantages of LLMs and RL.

Furthermore, our approach only utilizes the LLM during the training phase for policy distillation. During the testing phase, only the lightweight student agent is deployed online. This design choice reduces the computational burden and ensures efficient deployment of the learned policy.

We conducted a proof-of-concept study on our approach using three partially observable MiniGrid environments Chevalier-Boisvert *et al.* [2018]. The results demonstrate the effectiveness of utilizing a pre-trained LLM as a policy teacher for training RL agents. In future work, we plan to further evaluate our approach with larger LLMs and more complex tasks. This will provide a deeper understanding of its capabilities and potential for real-world applications.

References

- Mostafa Abdou, Artur Kulmizev, Daniel Hershcovich, Stella Frank, Ellie Pavlick, and Anders Søgaard. Can language models encode perceptual structure without grounding? a case study in color. *arXiv preprint arXiv:2109.06129*, 2021.
- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- 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, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for gymnasium, 2018.
- Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.
- Cédric Colas, Laetitia Teodorescu, Pierre-Yves Oudeyer, Xingdi Yuan, and Marc-Alexandre Côté. Augmenting autotelic agents with large language models. *arXiv preprint arXiv:2305.12487*, 2023.
- Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335, 2022.
- Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*, 2023.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. Enabling efficient interaction between an agent and an llm: A reinforcement learning approach. *arXiv preprint arXiv:2306.03604*, 2023.
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.
- Martin Klissarov, Pierluca D’Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.
- Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.
- Kolby Nottingham, Yasaman Razeghi, Kyungmin Kim, JB Lanier, Pierre Baldi, Roy Fox, and Sameer Singh. Selective perception: Optimizing state descriptions with reinforcement learning for language model actors. *arXiv preprint arXiv:2307.11922*, 2023.
- Roma Patel and Ellie Pavlick. Mapping language models to grounded conceptual spaces. In *International Conference on Learning Representations*, 2021.
- 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.
- Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- 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*, 2023.
- 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.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.
- Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

A Appendix

A.1 Options set

In this section, we provide a detailed description of the option set, with an emphasis on the termination conditions used in the LLM-based teacher baseline presented in Section 4.3.

For our MiniGrid environments, we employ the following set of options: $\{ \textit{explore}, \textit{go to an object}, \textit{pickup an object}, \textit{drop an object}, \textit{open an object} \}$. All options can be initiated from any state, i.e., $\mathcal{I}_\omega = \mathcal{S}$ for any option. Additionally, all options have a maximum length of 100 steps, after which the agent will terminate the option.

- $\{\textit{explore}\}$ option: During exploration, the agent follows a fixed strategy of scanning the unexplored grid row-by-row and finishes when it has observed walls forming a closed area.
- $\{\textit{go to an object}\}$ option: the agent plans the path to the target object using the A^* algorithm and terminates the option upon reaching the target.
- $\{\textit{pickup an object}\}$ option: the agent will attempt to pick up the target object in front of it if it is not already holding another object; otherwise, it will first drop the current object at the nearest position before picking up the new one.
- $\{\textit{drop an object}\}$ option: the agent will drop the current object at the nearest position.
- $\{\textit{open an object}\}$ option: it is a one-step action that attempts to interact with the object in front of the agent.

A.2 LLM-based agent performances

The LLM-generated plans may contain various types of errors given observations. In our research, we analyzed the error strategies proposed by LLMs, and identified them into three primary categories:

- **Incorrect strategies:** which are executable but lead to task failure. (such as moving into the lava).
- **Inefficient strategies:** which are executable but are not necessary. These strategies will increase the number of steps required to complete the tasks, potentially resulting in time-out errors. (e.g. keep exploring while the correct key and door have been found, repeatedly discarding the held key and picking up the observed key, rather than exploring the door.)
- **Inconsistent strategies:** which are not executable due to non-compliance with behavioral logic or contextual constraints. (e.g. attempting to pick up a new key while not dropping the holding key).

We have statistically investigated the probabilities of these three distinct errors occurring on the *ColorDoorKey* task and presented the results in the following table.

In the experiment, in order to ensure that the teacher’s policy can be implemented smoothly, for the first **incorrect strategies** error, we let the environment directly feedback the task termination, and for the **inconsistent strategies** error, we set the agent to not perform any action.

LLMs	Correct	Incorrect	Inefficiency	Inconsistent
Vicuna-7B	0.441	0.400	0.092	0.0667
ChatGLM-12B	0.558	0.217	0.025	0.200
Vicuna-13B	0.584	0.216	0.092	0.108
Vicuna-33B	0.691	0.025	0.167	0.117
ChatGLM-130B	0.750	0.133	0.042	0.075

Table 1: Probabilities of correct and three distinct error strategies provided by different LLMs occurring on the *ColorDoorKey* task.

Variable	Value
Number of trajectories per iteration	10
Number of epochs per iteration	3
Minibatch size	128
Entropy loss coefficient	0.001
Value function loss coefficient	0.5
Discount factor	0.99
Learning rate	0.001
Clipping parameter	0.2
Maximum gradient norm	0.5

Table 2: PPO hyperparameters

A.3 PPO Hyperparameters

A.4 LLMs’ decisions uncertainty

We give three ways to estimate LLMs’ uncertainty value for decision-making $Pr_{\text{LLM}}(k|c(s))$, namely **Statistical Estimates (SE)**, **Prompt Design (PD)**, and **Softmax of Logits (SoL)**. We tested the performance of three methods in *SimpleDoorKey*, as shown in the table below. The results show that

Performance	SE	PD	SoL
<i>Num of timesteps</i>	22.28	26.36	22.48
<i>Mean Return</i>	0.866	0.841	0.865
<i>Success rate</i>	100%	99.6%	100%

Table 3: Performance summary of LLMs’ uncertainty estimation methods in *SimpleDoorKey*

Statistical Estimates and **Softmax of Logits** are better, but considering that calculating logits word by word requires additional computational overhead during implementation, we use the first method to estimate in the main text.

A.5 Test result details

Here we present corresponding 1000 test numerical results shown in Figure 3, see Tables 4-6.

A.6 Ablation experiments

LLMs interactive parameter schedule design

Here, we show a result of decay-with-platform versus decay-without-platform versus always-constant. We designed ablation experiments to prove how to design LLMs interactive parameters that are reasonable. They are: 2000 steps with constant 0.1 ($k = 0, t = 2000$); 1000 steps linearly attenuate from 10 to 0 ($k = 0.01, t = 1000$); 2000 steps linearly attenuate from 10 to 0 ($k = 0.005, t = 2000$); 1000 steps attenuate linearly from 10 to 0.1 and then maintain 0.1 for 1000 steps and then set to 0 ($k = 0.01, t = 2000$), where

Performance	RL only	LLM agent	Ours
<i>Num of timesteps</i>	143.99	22.28	30.49
<i>Mean Return</i>	0.046	0.866	0.813
<i>Success rate</i>	10.1%	100%	96.4%

Table 4: The summary of the performance of our method and base-lines approach in *SimpleDoorKey*

Performance	RL only	LLM agent	Ours
<i>Num of timesteps</i>	149.61	15.79	36.84
<i>Mean Return</i>	0.003	0.482	0.746
<i>Success rate</i>	5%	54.6%	90.9%

Table 5: The summary of the performance of our method and base-lines approach in *LavaDoorKey*

Performance	RL only	LLM agent	Ours
<i>Num of timesteps</i>	145.84	90.73	36.27
<i>Mean Return</i>	0.032	0.402	0.776
<i>Success rate</i>	6.6%	46.8%	94%

Table 6: The summary of the performance of our method and base-lines approach in *ColoredDoorKey*

k is the decline slope and t is the termination step. Judging from the experimental results in Figure 4, only the last parameter setting can be successful. This is because in the short period of time when the LLM-based teacher is removed, students will have a period of policy oscillation. By allowing students to adapt to the LLM-based teacher’s small influence first and then removing it, they can spend this period of learning more smoothly.

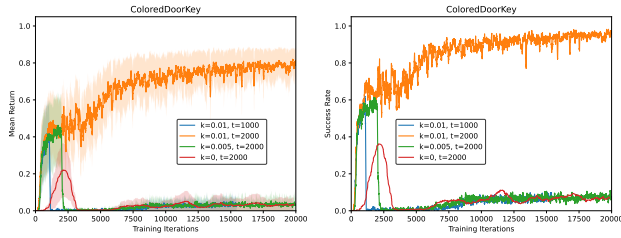


Figure 4: The training mean return (left) and success rates (right) of all compared parameter schedule design in *ColoredDoorKey* environments. The comparison shows that our coefficient schedule design enables successful distillation learning.

Correct LLM decision-making errors We specially selected some environments seeds where LLM-based teacher strategies are prone to errors and observed the changes in the success rate of the student agent in these environments as the training progressed. From the results in the Figure 5, we can see that the student agent corrects the mistakes of the teacher’s strategy during the distillation process, thereby achieving performance that surpasses the teacher’s.

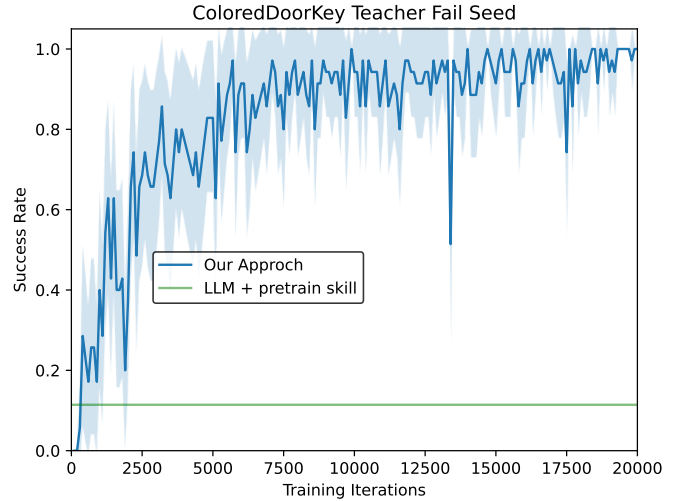


Figure 5: After 2000 steps away from LLM guidance, the success rates in these environments are greater than those of teachers and gradually increase.