AdaRefiner: Refining Decisions of Language Models with Adaptive Feedback

Wanpeng Zhang
School of Computer Science
Peking University
wpzhang@stu.pku.edu.cn

Zongqing Lu[†]
Peking University
BAAI
zongqing.lu@pku.edu.cn

Abstract

Large Language Models (LLMs) have demonstrated significant success across various domains. However, their application in complex decision-making tasks frequently necessitates intricate prompt engineering or fine-tuning, leading to challenges in unseen downstream tasks and heavy demands on computational resources. Meanwhile, Reinforcement Learning (RL) has been recognized as effective in decision-making problems but struggles in environments with sparse rewards, such as open-world games. To overcome these challenges, we introduce AdaRefiner, a novel framework designed to enhance the synergy between LLMs and RL feedback. The key component of AdaRefiner is a lightweight Adapter Language Model (LM), which automatically refines task comprehension based on feedback from RL agents. This method mitigates the need for intricate prompt engineering and intensive LLM fine-tuning while maintaining the LLMs' generalization abilities and enhancing their decision-making capabilities in downstream tasks. Empirical evaluations of AdaRefiner on 22 diverse tasks within the open-world game *Crafter* have demonstrated its superior effectiveness, especially in guiding agents towards higher-level and common-sense skills. Our work makes contributions to the automatic self-refinement of LLMs with RL feedback, offering a more adaptable and efficient solution for complex decision-making problems. The code is available at `https://github.com/PKU-RL/AdaRefiner`.

1 Introduction

The rapid development of Large Language Models (LLMs), trained on massive corpora, has opened new frontiers in various fields, leveraging their ability to process and generate text (Wei et al., 2022). Notably, LLMs have demonstrated impressive performance in decision-making problems (Yao et al., 2023; Shinn et al., 2023; Sun et al., 2023). However, recent studies highlight that directly applying LLMs to complex decision-making tasks often necessitates intricate prompt engineering and external feedback (Wang et al., 2023a; Wu et al., 2023b; Wang et al., 2023b). Such task-specific designs pose challenges in transferring these methods to different scenarios. Some studies have explored the use of task-related data to fine-tune LLMs to improve decision-making capabilities (Nottingham et al., 2023; Feng et al., 2023). However, such approaches often encounter practical challenges, such as inaccessible LLM weights or intensive computational demands. Moreover, fine-tuning LLMs may lead to decreases in their generalization capabilities (Wang et al., 2022), making their deployment across diverse environments challenging. These challenges underscore the need for a more adaptable and generalizable approach.

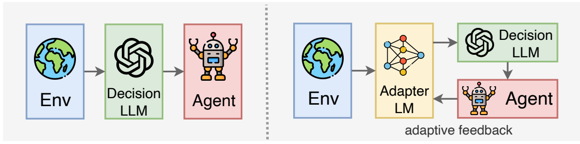


Figure 1: Core differences between AdaRefiner (*right*) and typical LLM-based methods (*left*). The key distinction is the integration of Adapter LM, which enhances the synergy between LLMs and adaptive feedback.

Before the emergence of LLMs, Reinforcement Learning had been recognized for its impressive capabilities in decision-making problems (Mnih et al., 2015; Silver et al., 2017). The strength of RL is most evident when agents consistently receive clear and dense rewards that guide them toward the targeted behaviors (Ladosz et al., 2022; Eschmann, 2021). However, designing such reward functions is far from straightforward. It often requires meticulous engineering and access to a comprehensive set of task-specific information. This challenge becomes even more pronounced in naturally sparse-

[†]Corresponding Author

Findings of the Association for Computational Linguistics: NAACL 2024, pages 782–799

reward environments. In such contexts, integrating LLMs to assist RL agents has emerged as a promising direction (Du et al., 2023). Despite the potential of this approach, LLMs may face difficulties in understanding specific environments (Bommasani et al., 2021; Ahn et al., 2022). This limitation undermines their efficacy in assisting RL agents.

In this paper, our goal is to enhance LLMs to better understand specific environments without relying on demanding prompt engineering or directly fine-tuning LLMs, while assisting RL agents with complex decision-making tasks. To this end, we propose a novel framework, AdaRefiner, where the LLM provides guidance to the RL agent who selects fine-grained actions to accomplish tasks. Simultaneously, the RL agent contributes adaptive feedback, enriching the LLM's understanding of the environment through an adjustable module.

As illustrated in Figure 1, the core feature of AdaRefiner is the integration of a lightweight Adapter LM. This Adapter LM, enriched with feedback and information from the RL agent, automatically prompts a Decision LLM, like GPT-4 (OpenAI, 2023). It enables a refined understanding of the environment and agents' learning capabilities without the need to alter the Decision LLM's parameters. This approach maintains the generalization abilities of LLMs while providing targeted assistance for RL agents with specific tasks. By the synergy of LLMs and RL feedback, AdaRefiner addresses the limitations of existing methods, setting a new paradigm in the integration of advanced LLMs with reinforcement learning.

In the experiments, AdaRefiner is evaluated on 22 tasks within the *Crafter* environment (Hafner, 2021). The results not only demonstrate AdaRefiner's superior performance compared to state-of-the-art baselines but also highlight its ability to guide agents towards common-sense behaviors.

Our key contributions are summarized as follows: **1)** We propose a novel framework that aligns LLMs with downstream tasks and guides agents to effectively learn complex tasks without the need for intricate prompt engineering or intensive fine-tuning; **2)** We design the Adapter LM that correlates its own update with the learning progress of the agent and automatically generates appropriate prompts for the Decision LLM, thereby forming a feedback loop together with LLMs and RL agents; **3)** We thoroughly evaluate our framework's efficacy on 22 diverse tasks and provide a comprehensive analysis of the experimental results.

2 Related Work

Large Language Models (LLMs). Recent advancements in natural language processing have been significantly shaped by the emergence of LLMs. The GPT series, notably, has garnered attention for its broad task versatility, while other models like PALM and LaMDA have also contributed to the field with their unique capabilities (Chowdhery et al., 2022; Thoppilan et al., 2022). A pivotal development in the evolution of LLMs is the implementation of instruction tuning (Ouyang et al., 2022), which has markedly enhanced adaptability in complex scenarios, particularly in zero-shot and few-shot learning applications. The open sourcing of some LLMs (Zeng et al., 2022; Touvron et al., 2023a) has spurred efforts in task-specific fine-tuning (Wu et al., 2023a). While this approach often boosts task performance, it can simultaneously reduce the models' generalization abilities (Wang et al., 2022). Our work navigates this challenge by dynamically fine-tuning a lightweight Adapter LM via real-time feedback from RL agents, aiming to strike a balance between task-specific improvement and broad applicability. This method tailors the LLM for specific tasks while maintaining LLM's broad adaptability to new environments, addressing a key limitation in current applications.

LLMs for RL. Incorporating language models to represent goals in RL utilizes the extensive knowledge of LLMs trained on large corpora. The use of LM-encoded goal descriptions has been shown to significantly improve the generalization capabilities of instruction-following agents (Chan et al., 2019; Hill et al., 2020). This is achieved by enabling agents to interpret and act upon complex instructions more effectively. Furthermore, pre-trained LLMs provide nuanced guidance through sub-goals and sub-policies, enhancing agent strategies and decision-making in various scenarios (Lynch and Sermanet, 2020; Sharma et al., 2021). Subsequent research efforts have linked these sub-policies to address more intricate tasks in RL environments (Huang et al., 2022a,b). Several methods also leverage LLMs to generate intrinsic rewards, boosting the efficiency and effectiveness of RL learning (Choi et al., 2022; Du et al., 2023). However, the application of these methods in simple text-based games often does not transfer well to more complex and dynamic environments, leading to scalability and generalization issues (Zhong et al., 2021; Wang and Narasimhan, 2021). Our work addresses these challenges by making LLMs more adaptable and practical for use in sophisticated environments. The AdaRefiner framework is specifically designed to enhance the flexibility and effectiveness of LLMs, providing tailored assistance to RL agents in navigating and mastering complex decision-making tasks.

LLMs for Open-World Games. Open-world games pose unique challenges, such as managing long horizons (Hafner, 2021) and balancing multiple objectives (Wang et al., 2023c). These complexities require sophisticated decision-making strategies. While some studies have explored using LLMs for planning and guiding RL agents (Du et al., 2023; Yuan et al., 2023; Tsai et al., 2023), their approaches often depend on human-generated trajectories as context. This dependency can limit the agent's performance in unseen scenarios, making them less effective compared to recent RL algorithms (Hafner et al., 2023) that operate independently of LLMs. Additionally, methods that solely rely on LLMs for decision-making (Wu et al., 2023b; Wang et al., 2023a) often have designs that are intricately tailored to specific environments or require expert-level prior knowledge. This specificity can make them less transferable to different tasks. In contrast, our AdaRefiner avoids such complexity. Its straightforward and flexible design enables it to adapt to a variety of tasks and environments, addressing the key limitations of current LLM applications in open-world games.

3 Methodology

3.1 Problem Formulation

In our study, the primary goal is to leverage LLMs to enhance the decision-making capabilities of RL agents in complex environments. We consider a partially observable Markov decision process (POMDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \Omega, \mathcal{O}, R, \gamma)$. Here, $s \in \mathcal{S}$ and $a \in \mathcal{A}$ denote the state and action, respectively. The transition probability $\mathcal{P}(s'|s, a)$ represents the environment dynamics, where s' is the state following action a from state s. The observation $o \in \Omega$ is obtained through function $\mathcal{O}(o|s, a)$, and R is the reward function, with γ as the discount factor. We can use $\tau \doteq \{o_0, a_0, r_0, \ldots, o_t, a_t, r_t, \ldots\}$ to represent a sequence of data as a trajectory.

Under this setting, we employ LLMs to generate sub-goals g, aiding agents in decision-making processes. These sub-goals are designed to provide intermediate targets, enhancing the agent's ability in complex scenarios. Our objective is to develop a policy, denoted as $\pi(a|o, g)$, which maximizes cumulative reward by effectively integrating these sub-goals. The specific mechanics of how LLMs assist in generating these sub-goals and their exact role in the decision-making process will be detailed in subsequent sections.

3.2 Key Idea and Overall Framework

Pre-trained LLMs demonstrate impressive zero-shot language understanding capabilities across diverse tasks. This proficiency can be leveraged to help agents quickly comprehend complex environments, thus mitigating exploration dilemmas in RL. By prompting LLMs, we obtain sub-goals in textual format, which are then embedded with the agent's observations to inform the policy $\pi(a|o, g)$. This process aids agents in making more informed decisions based on the contextual guidance provided by these sub-goals.

Despite their generalization capabilities, LLMs may not always have a comprehensive understanding of specific tasks, leading to potential mismatches between the generated guidance and the environment's realities. Directly using LLM-generated guidance may not result in coherent or relevant advice. While fine-tuning LLMs with task-specific data is a typical solution, it can be computationally intensive and may also lead to catastrophic forgetting of pretrained knowledge. Moreover, fine-tuning black-box models like GPT-4 is infeasible due to restricted access to their weights.

Given these challenges, we focus on adding adjustable modules to help LLMs adapt to environments, rather than modifying the LLMs directly. A key insight is that even a lightweight LM, with the right fine-tuning, can excel at particular tasks (Zhang et al., 2023; Li et al., 2023). This motivates us to propose AdaRefiner, as illustrated in Figure 2. The core component of AdaRefiner is a lightweight Adapter LM which bridges the gap between specific environments and the Decision LLM's capabilities. The Adapter LM first processes the environmental inputs and the agent's status, automatically generating tailored prompts that include summaries and suggestions. These prompts are then fed into the Decision LLM, which produces final sub-goals. The Adapter LM thus acts as an intermediary, ensuring that the Decision LLM receives contextually relevant information, enabling it to provide accurate and useful guidance to the agent.

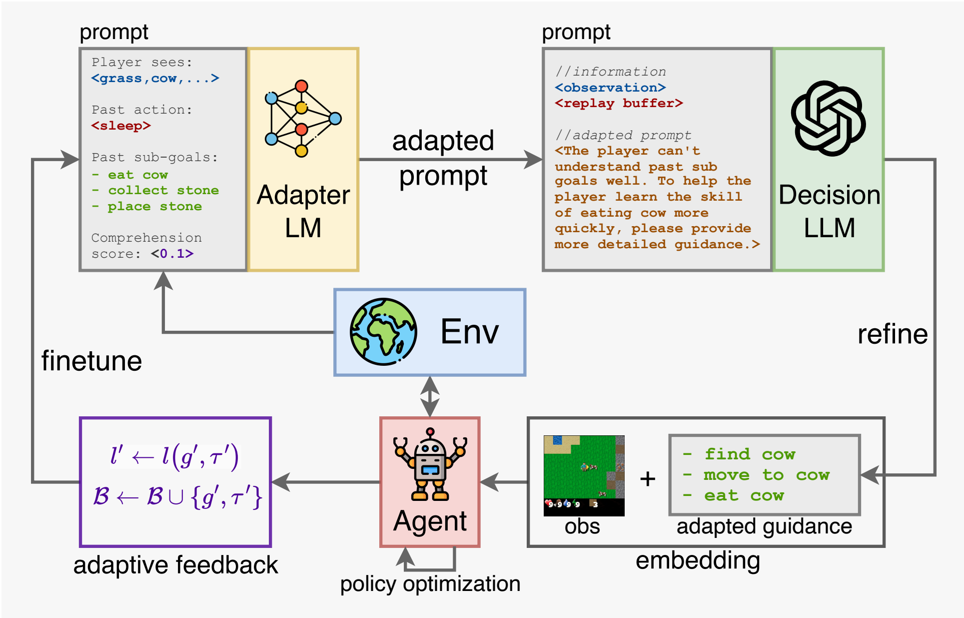


Figure 2: Overall framework of AdaRefiner. In addition to receiving inputs from the environment and historical information, the prompt of the Adapter LM incorporates a comprehension score. This score computes the semantic similarity between the agent's recent actions and the sub-goals suggested by the LLM, determining whether the agent currently comprehends the LLM's guidance accurately. Through the agent's feedback and continuously fine-tuning the Adapter LM, we can keep the LLM always attuned to the actual circumstances of the task. This, in turn, ensures that the provided guidance is the most appropriate for the agents' prioritized learning.

3.3 Adapter LM

The Adapter LM processes two types of input information: environmental information and the agent's comprehension level of language guidance. The environmental information, sourced from the game engine or visual descriptors (Radford et al., 2021), includes critical information such as object properties and the current status of the agent. The agent's comprehension level of language guidance is quantified using a cosine similarity score l, calculated between the suggested sub-goals and the agent's trajectories, represented as:

$$l \doteq \cos(g, \tau) = \frac{f_{\text{emb}}(g) \cdot f_{\text{emb}}(\tau)}{\|f_{\text{emb}}(g)\| \cdot \|f_{\text{emb}}(\tau)\|}. \quad (1)$$

Here, f_{emb} represents the embedding function, with SentenceBert (Reimers and Gurevych, 2019) employed in our implementation. A higher score l suggests that the agent's actions are more closely aligned with the sub-goals, indicating a better comprehension of the provided guidance.

The Adapter LM then utilizes the comprehension score l and environmental information to generate $\texttt{prompt}_{\texttt{a}}(\mathcal{B}, l)$, where $\mathcal{B}$ is a replay buffer of the agent's historical contexts and $\texttt{prompt}_{\texttt{a}}(\cdot)$ is the prompt template for Adapter LM. After analyzing the prompt, the Adapter LM synthesizes the information to assist the Decision LLM, which is responsible for overall decision-making. The output from the Adapter LM, represented as $c \sim \mathcal{M}_{\text{a}}(\texttt{prompt}_{\texttt{a}}(\mathcal{B}, l))$, is then used to inform the Decision LLM. Here, $\mathcal{M}_{\text{a}}$ represents the Adapter LM. By providing tailored information through the adapted $\texttt{prompt}_{\texttt{d}}(\mathcal{B}, c)$, the Decision LLM is better equipped to generate appropriate sub-goals $g \sim \mathcal{M}_{\text{d}}(\texttt{prompt}_{\texttt{d}}(\mathcal{B}, c))$. Here, $\mathcal{M}_{\text{d}}$ represents the Decision LLM and $\texttt{prompt}_{\texttt{d}}(\cdot)$ is the prompt template for the Decision LLM. Details of these prompts is available in Appendix D.

3.4 Training Procedure

The training process of our framework is designed to coordinate the learning of RL agents and the fine-tuning of the Adapter LM. In other words, the Adapter LM is continuously updated to refine its comprehension of the environment and the agent in parallel with the RL agent's exploration and data collection. Specifically, the RL agent receives suggested sub-goals $g \sim \mathcal{M}_{\text{d}}(\texttt{prompt}_{\texttt{d}}(\mathcal{B}, c))$ from the Decision LLM, which are then provided to the policy $\pi(a|o, g_{\text{emb}})$ for training. Here, g_{emb} is the text embedding produced by f_{emb}. The agent's actions and the resultant trajectories provide an updated comprehension score $l' = \cos(g', \tau')$, where g', τ' is the new sub-goals and trajectories. This score and collected information are then used to compose a linguistic data pair $< \texttt{prompt}_\texttt{a}(\mathcal{B}, l'), c >$ for supervised fine-tuning of the Adapter LM. Then the replay buffer will be updated as $\mathcal{B} \leftarrow \mathcal{B} \cup \{g', \tau'\}$ This iterative procedure allows the Adapter LM to continuously refine its self-awareness and generate more effective summaries c, which affects the quality of guidance for the RL agent.

Considering the computational costs and the nature of open-world game environments, we query the language models at predetermined intervals instead of every step. This strategy ensures a balance between consistent guidance and computational efficiency. The fine-tuning of the Adapter LM is also conducted at specific intervals for the same reason. In line with our claim that only a lightweight Adapter LM is needed, we utilize the 4-bit quantized version of the Llama2-7B model (Touvron et al., 2023b) as the base model (Jiang et al., 2023) and employ QLoRA (Dettmers et al., 2023) for efficient fine-tuning. And we choose OpenAI's GPT-4 as the default Decision LLM. These choices will be further discussed and analyzed in Section 4. For policy learning, we adopt the classic Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017). It is worth noting that our framework is designed to be compatible with a variety of standard RL algorithms, not only limited to PPO.

Specific parameters and settings are detailed in Appendix B. The complete procedure can be found in Appendix A.

4 Experiment

Our experiments primarily aim to validate the following claims: **1)** The integration of the Adapter LM can enhance LLM's comprehension of downstream tasks and the agent's understanding capability, resulting in more meaningful guidance; **2)** Agents trained under the AdaRefiner framework can exhibit superior performance and demonstrate higher-level decision-making capabilities.

4.1 Experiment Settings

Our experiments are conducted in the *Crafter* environment (Hafner, 2021), a widely used benchmark with 22 different tasks for evaluating the decision-making capabilities of agents in open-world games.

Environment Details. *Crafter* features a 64×64 grid map populated with various objects (*e.g.*, grass, water, wood) and entities (*e.g.*, player, zombie, skeleton). Agents in this environment have access to a local 9×7 area for observation, presenting a challenge in terms of limited information and requiring effective decision-making for long-term survival and resource management. In *Crafter*, agents are not bound to a single main task. Instead, they are expected to master a range of skills to accomplish 22 different tasks, including tasks such as collecting resources, crafting tools, and surviving against environmental hazards. This variety tests the agents' ability to learn and adapt to diverse challenges, aligning well with our objective to enhance their decision-making capabilities through the AdaRefiner framework.

Evaluation Metrics. In *Crafter*, the performance of an agent is evaluated using three metrics: reward, success rate, and overall score. The reward is designed to reflect the agent's skills. Each time an agent unlocks a new achievement, it receives a $+1$ reward. Additionally, the agent is rewarded with $+0.1$ or penalized with -0.1 for every gain or loss of a health point, respectively. The success rate is defined as the proportion of episodes in which agents complete a achievement. Completing the same achievement multiple times within an episode does not affect the success rate. The overall score averages the success rates ($s_i \in [0, 100]$) of the 22 achievements in log-space as follows (known as the geometric mean): $S \doteq \exp\left(\frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + s_i\right)\right) - 1$, where $N = 22$ is the total number of achievements.

Prompt Design. The prompt design for the Adapter LM is crafted to encapsulate critical information for decision-making. It includes observations of objects and the agent's status obtained from the game engine, along with the comprehension score l. The format is: "`Player sees: <observations>; Player status: <status>; Past action: <past actions>; Past sub-goals: <last suggested sub-goals>; Comprehension score: <l>. Analyze the environment and the player's understanding capability, then generate concise summaries and suggestions about this player.`" For the Decision LLM, we construct the prompt based on the Adapter LM's output: "`<output of the Adapter LM>. Based on the provided information, suggest 3 sub-goals that the player should accomplish next.`"

4.2 Baselines

To demonstrate the effectiveness of our AdaRefiner framework, we conduct comparative analyses against a diverse set of methods:

LLM-based Methods: We compare AdaRefiner with LLM-based methods such as Reflexion (Shinn et al., 2023), ReAct (Yao et al., 2023), and Vanilla GPT-4. Reflexion and ReAct leverage chain-of-thought prompts for decision-making tasks. Considering that LLM-based methods do not accept image input, we additionally include the coordinates of objects in the prompt for fair comparisons. We also maintain consistency in the prompts used across all methods (detailed in Appendix D). The LLM used in Reflexion and ReAct is the same as the Decision LLM in AdaRefiner, *i.e.*, GPT-4 by default. These comparisons aim to demonstrate how the integration of LLMs with adaptive feedback can provide a more comprehensive approach to decision-making.

RL Methods: We also benchmark against RL methods such as DreamerV3 (Hafner et al., 2023), Rainbow (Hessel et al., 2018), PPO (Schulman et al., 2017), RND (Burda et al., 2019), and Plan2Explore (Sekar et al., 2020). DreamerV3 is notable for its performance in model-based RL. Rainbow is a classic algorithm that achieves great performance in many games. RND and Plan2Explore are known for the intrinsically motivated exploration. PPO, which is also adopted in AdaRefiner, serves to highlight the added value of LLMs in the same RL setup.

Additional References: We include random policy, human expert performance (Hafner, 2021), SPRING (Wu et al., 2023b) that provides GPT-4 with domain-specific prior knowledge (*i.e.*, research papers about the game engine), and Reflexion with *gpt-4-vision* (including both coordinates and image inputs), as additional references to showcase performances enhanced with different information and knowledge.

4.3 Results and Analysis

The comparison includes some methods for which open-source codes are unavailable. For these algorithms, we rely on the performance metrics reported in respective papers, ensuring that the comparisons are as consistent as possible in terms of experimental setup and evaluation criteria. For RL baselines, we set the training to 1 million steps, following the standard set in the *Crafter* paper (Hafner, 2021). However, LLM-based baselines do not include a training phase and instead focus on leveraging pre-trained LLMs. To facilitate a fair comparison, we also present a version of AdaRefiner trained for 5 million steps to assess its asymptotic performance. This extended training is essential for evaluating AdaRefiner's full potential and maintaining comparability with baselines.

Method Type	Method	Score (%)	Reward
Ours	AdaRefiner (@5M)	**28.2** $\pm$ 1.8	**12.9** $\pm$ 1.2
	AdaRefiner (@1M)	15.8 $\pm$ 1.4	12.3 $\pm$ 1.3
LLM-based methods	Reflexion (GPT-4)	11.7 $\pm$ 1.4	9.1 $\pm$ 0.8
	ReAct (GPT-4)	8.3 $\pm$ 1.2	7.4 $\pm$ 0.9
	Vanilla GPT-4	3.4 $\pm$ 1.5	2.5 $\pm$ 1.6
RL methods	DreamerV3	14.5 $\pm$ 1.6	11.7 $\pm$ 1.9
	PPO	4.6 $\pm$ 0.3	4.2 $\pm$ 1.2
	Rainbow	4.3 $\pm$ 0.2	5.0 $\pm$ 1.3
	Plan2Explore	2.1 $\pm$ 0.1	2.1 $\pm$ 1.5
	RND	2.0 $\pm$ 0.1	0.7 $\pm$ 1.3
Addtional references	Human Experts	50.5 $\pm$ 6.8	14.3 $\pm$ 2.3
	SPRING (+prior)	27.3 $\pm$ 1.2	12.3 $\pm$ 0.7
	Reflexion (GPT-4-Vision)	12.8 $\pm$ 1.0	10.3 $\pm$ 1.3
	Random	1.6 $\pm$ 0.0	2.1 $\pm$ 1.3

Table 1: Performance comparison between AdaRefiner and baselines in terms of score and reward metrics. AdaRefiner is trained with 5 million and 1 million steps. All results are derived from 500 inference episodes. Note that $\pm$ captures standard deviations.

Method	Achievements (out of 22)	Achievement Depth (max 8)
AdaRefiner	**21**	**7**
DreamerV3	19	6
Reflexion	17	5

Table 2: Numbers and depths of achievements that can be completed by different methods. The achievement depth refers to the number of prerequisite steps required to complete each task, with a maximum value of 8.

Results in Table 1 show that AdaRefiner with 1 million training steps outperforms all baselines. In comparisons with RL methods, the integration of LLM demonstrates a clear advantage in learning effectiveness. The performance of AdaRefiner compared to Reflexion and ReAct underscores that prompts generated automatically by the Adapter LM can enhance the decision-making capabilities of LLMs in downstream tasks more effectively than traditional prompt engineering techniques. This efficiency, combined with the adaptability of AdaRefiner, establishes it as a highly practical and powerful framework in complex decision-making environments.

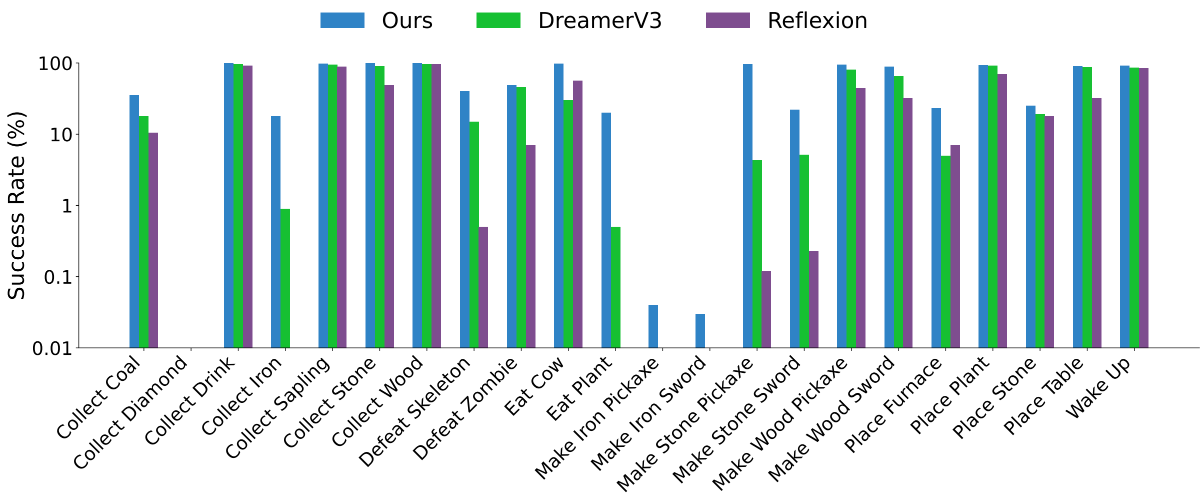


Figure 3: Success rates of unlocking 22 different achievements in log scale. AdaRefiner outperforms the two top-performing baselines. Notably, AdaRefiner is the only method that successfully completes the level-7 tasks "Make Iron Pickaxe" and "Make Iron Sword".

Additionally, it is noteworthy that AdaRefiner with 5 million training steps slightly outperforms the SPRING (+prior) method, which necessitates providing task-related papers and engaging in a 9-round chain-of-thought questioning process. This indicates that AdaRefiner can achieve better performance simply through a comprehensive understanding of adaptive feedback, without the need for external expert-level knowledge and complex prompt engineering.

Moreover, the augmented Reflexion, which utilizes *gpt-4-vision*, showes some improvements over the original version. However, even with the inclusion of additional input information, Reflexion still exhibited a significant performance gap compared to AdaRefiner. This indicates that merely employing pretrained large multimodal models with image understanding capabilities does not guarantee improved performance on downstream tasks, further underscoring the efficacy of AdaRefiner's framework design.

To study the breadth of abilities learned by different methods, we compare AdaRefiner with two top-performing baselines, DreamerV3 and Reflexion. We investigate their success rates on 22 specific achievements in *Crafter*. Both AdaRefiner and DreamerV3 are trained for 5 million steps. Figure 3 illustrates that AdaRefiner has the highest success rates across all tasks. Moreover, as shown in Table 2, AdaRefiner completes the largest number of achievements and is the only method that reaches level-7 difficulty. Specifically, AdaRefiner is notably the only method capable of accomplishing level-7 tasks "Make Iron Pickaxe" and "Make Iron Sword". These tasks are particularly hard due to their prerequisite conditions and rarity in the game. This result underscores the importance of a comprehensive understanding of environments in developing versatile agents.

4.4 Ablation Study

To investigate the contribution of various components in the AdaRefiner framework, a series of ablation studies are conducted.

Decision LLM Variants. We first investigate the performance of using different Decision LLMs. By replacing GPT-4 with GPT-3.5 in the Decision LLM, we observe a slight decrease in performance, as shown in the first two rows of Table 3. This result suggests that AdaRefiner using an LLM with less capability still maintains a comparable level to other baselines, achieving level-6 tasks. This demonstrates that the success of AdaRefiner is primarily attributed to its framework design, rather than the use of more advanced GPT-4 as the Decision LLM. In contrast, when comparing the two versions of Reflexion under the same Decision LLMs, significant performance gaps are observed, further underscoring the superiority of our framework.

Adapter LM Variants. To study the contribution of the Adapter LM to AdaRefiner, we design two variants as shown in the middle three rows of Table 3. The first variant, *AdaRefiner w/o l-score*, excludes the comprehension score from

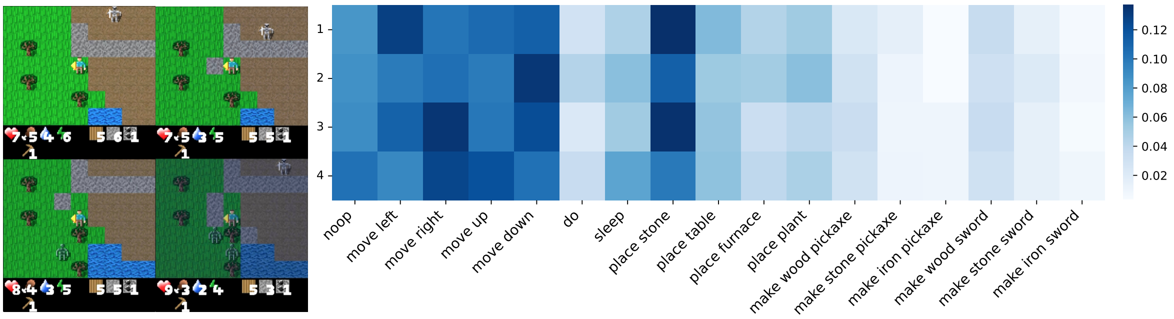


Figure 4: (*left*) Frames from an episode in the game, the order is from top left to bottom right. (*right*) The probabilities of actions in the agent's policy corresponding to each frame.

Method (@5M steps)	Score (%)	Reward	Achievement Depth
AdaRefiner	**28.2** $\pm$ 1.8	**12.9** $\pm$ 1.2	7
AdaRefiner (GPT-3.5)	23.4 $\pm$ 2.2	11.8 $\pm$ 1.7	6
Reflexion (GPT-4)	11.7 $\pm$ 1.4	9.1 $\pm$ 0.8	5
Reflexion (GPT-3.5)	8.9 $\pm$ 1.7	7.2 $\pm$ 1.1	4
AdaRefiner w/o l-score	13.4 $\pm$ 1.9	9.2 $\pm$ 1.6	5
AdaRefiner w/o Adapter LM	9.6 $\pm$ 1.7	8.7 $\pm$ 1.4	5
GPT-4 + GPT-4	7.5 $\pm$ 0.8	5.2 $\pm$ 1.5	4
Llama2-7B + GPT-4	7.1 $\pm$ 1.0	4.7 $\pm$ 1.5	4
AdaRefiner w/ binary score	18.7 $\pm$ 2.4	11.0 $\pm$ 1.6	6

Table 3: Ablation study of AdaRefiner. The results illustrate the impact of various components.

both the prompts and the fine-tuning process. This variant experiences a notable performance decline, highlighting the critical role of the comprehension score in refining the Adapter LM with task objectives. It appears that merely using task data for fine-tuning does not sufficiently enhance decision-making capabilities. Another variant, *AdaRefiner w/o Adapter LM*, retains the comprehension score but removes the Adapter LM. This setup leads to an even more pronounced decrease in performance, indicating that simply providing comprehension scores as inputs is not enough to significantly increase decision-making effectiveness. It demonstrates that the Adapter LM, when fine-tuned with comprehension scores, plays a pivotal role in enhancing the overall decision-making capabilities.

Feedback from RL. To demonstrate the significance of integrating adaptive feedback from RL, we compare two variants that remove adaptive feedback from RL and rely solely on the Decision LLM for action decisions. In these variants, PPO and corresponding feedback are removed, and the Adapter LM is used only for inference, without any fine-tuning. The results are shown in the last three rows of Table 3. The first variant, named *Llama2-7B + GPT-4*, shows a significant decrease in performance. This underscores the critical role of incorporating adaptive feedback from RL for the Adapter LM to accurately perceive and adapt to the environment. Another variant, *GPT-4 + GPT-4*, which utilizes GPT-4 as the Adapter LM for inference, exhibits similar performance, further suggesting that simply increasing the capacity of LLMs is insufficient. These comparisons demonstrate that the synergy between LLMs and RL feedback is crucial to the efficacy of AdaRefiner.

Fine-grained Comprehension Score. To verify the necessity of using a fine-grained comprehension score, we investigate the impact of the score's format on performance. Specifically, we compare the performance with a variant, *AdaRefiner w/ binary score*, which assigns a score of 1 to entries above a 0.5 similarity threshold and 0 to others. The results clearly show that replacing the comprehension score with a binary score leads to a significant decrease in performance. This indicates that a finer-grained similarity score is more effective in aiding the Adapter LM to understand the agent's capabilities, showcasing the Adapter LM's sensitivity to score values.

4.5 Guidance and Agent Behaviors

We further investigate how AdaRefiner enhances the agent's comprehension and learning. As shown on the left side of Figure 4, in a scenario where enemies gradually appear, AdaRefiner receives environmental information and suggests the agent to "place stone to build shelter, collect food and drink, avoid combat". The policy visualized on the right side of Figure 4, reveals a high probability of "place stone" following this guidance. Notably, five basic actions controlling the player's movement also maintain high probabilities. This pattern likely reflects the inherent design of RL algorithms to encourage exploration, leading agents to consistently engage in common and easily executed actions. Actions less relevant to the provided guidance exhibit lower probabilities, indicating the agent's ability to prioritize actions based on AdaRefiner's suggestions. For more detailed analyses, statistical tests, and further demonstrations, please refer to Appendix C.

4.6 Consistent Increment of Performance and Agent's Comprehension.

To further validate the efficacy of AdaRefiner in providing effective guidance for the agent, we investigate the correlation between the learning curve and the comprehension score during training. Figure 5 illustrates this relationship, showing that there is a consistent increase in the comprehension score as training progresses. This increment suggests an improvement in the agent's understanding of the language guidance, which in turn enhances the overall performance. The results demonstrate that the agent is not just following instructions more accurately but is also integrating this guidance more effectively into its decision-making process.

4.7 Behavior Statistics

To better quantify the guidance provided by AdaRefiner and the common-sense behavior exhibited by the agent, we have adopted a setting similar to that used in existing work (Du et al., 2023). Specifically, we categorized each instruction and actual agent action into three groups:

- **No Common-Sense** (where behavior significantly deviates from typical human common sense, *i.e.*, *suggesting the agent to fight with enemies when its health is low*);
- **Impossible** (where the resources and conditions do not support the behavior in game engine);
- **Reasonable** (all remaining behaviors not included in the first two categories).

The results are shown in Table 4, suggesting that in most scenarios (83.8% and 78.6%), the guidance and agent's actions are reasonable. While we acknowledge that there may be subjectivity in this assessment, we believe the results can still show the general tendencies of AdaRefiner in guiding agent's behavior.

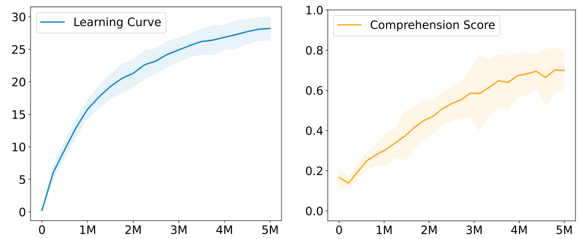


Figure 5: Learning curve (*left*) and comprehension score (*right*) of AdaRefiner.

	Guidance	**Action**
No Common-Sense	12.8%	21.4%
Impossible	3.4%	0%
Reasonable	83.8%	78.6%

Table 4: Statistical tests on common-sense behaviors. The results are based on 500 inference samples.

5 Conclusions

In this study, we introduce AdaRefiner, a novel framework that synergizes LLMs with adaptive feedback, leveraging an Adapter LM as a crucial intermediary. AdaRefiner, rigorously tested across 22 diverse tasks in the *Crafter* environment, not only outperforms state-of-the-art baselines but also steers agents towards learning higher-level skills and exhibiting common-sense behaviors. Ablation studies further validate the significance of each component, particularly emphasizing the Adapter LM's role in refining decision-making. These results highlight AdaRefiner's potential in advancing LLMs' capabilities in complex open-world games, and open up avenues for further research in LLM's decision-making capabilities.

Limitations

The primary limitation of AdaRefiner is that it still requires a certain level of pre-trained knowledge of the Adapter LM. If a smaller language model is used as the Adapter LM, its language understanding ability may not be sufficient to provide the necessary analysis and summarization for the environment and agent. Additionally, although AdaRefiner substantially improves the performance, all methods including AdaRefiner fall short in the most difficult level-8 task "Collect Diamond." This gap points to a need for further improvements in current methods to tackle more complex tasks.

Nevertheless, the uncovering of knowledge from LLMs by the Adapter LM demonstrates promising prospects for filling the gap in LLMs' performances across various tasks. In future work, we will continue to explore this characteristic of the Adapter LM while also attempting to integrate LLM with RL algorithms more closely to address these limitations in complex environments.

Ethical Considerations

While the natural language guidance generated by LLMs exhibits strong common-sense capabilities, there is a possibility that they might contain or produce harmful information. Though no such concerns were observed during evaluations in simulated environments like *Crafter*, it is imperative to address these potential risks when transferring AdaRefiner to more open and real-world settings in the future. Mitigating these risks can be achieved by adding additional instructions in prompts, fine-tuning with curated data, and post-processing the generated text. Adopting these measures ensures that AdaRefiner functions effectively and safely in its intended roles.

Acknowledgements

This work was supported in part by NSF China under grant 62250068.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do as I can, not as I say: Grounding language in robotic affordances. In *CoRL*.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. Exploration by random network distillation. In *Seventh International Conference on Learning Representations*, pages 1–17.

Harris Chan, Yuhuai Wu, Jamie Kiros, Sanja Fidler, and Jimmy Ba. 2019. Actrce: Augmenting experience via teacher's advice for multi-goal reinforcement learning. *arXiv preprint arXiv:1902.04546*.

Kristy Choi, Chris Cundy, Sanjari Srivastava, and Stefano Ermon. 2022. LMPriors: Pre-trained language models as task-specific priors. *arXiv preprint arXiv:2210.12530*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. 2023. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*.

Jonas Eschmann. 2021. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, pages 25–33.

Yicheng Feng, Yuxuan Wang, Jiazheng Liu, Sipeng Zheng, and Zongqing Lu. 2023. Llama rider: Spurring large language models to explore the open world. *arXiv preprint arXiv:2310.08922*.

Danijar Hafner. 2021. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. 2020. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion*.

Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2023. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. *arXiv preprint arXiv:2308.12060*.

Corey Lynch and Pierre Sermanet. 2020. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. 2023. Do embodied agents dream of pixelated sheep?: Embodied decision making using language guided world modelling. *arXiv preprint arXiv:2301.12050*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. 2020. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR.

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. 2021. Skill induction and planning with latent language. *arXiv preprint arXiv:2110.01517*.

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Chen Feng Tsai, Xiaochen Zhou, Sierra S Liu, Jing Li, Mo Yu, and Hongyuan Mei. 2023. Can large language models play text games well? current state-of-the-art and open questions. *arXiv preprint arXiv:2304.02868*.

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

HJ Wang and Karthik Narasimhan. 2021. Grounding language to entities and dynamics for generalization in reinforcement learning. *arXiv preprint arXiv:2101.07393*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023b. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

Yihan Wang, Si Si, Daliang Li, Michal Lukasik, Felix Yu, Cho-Jui Hsieh, Inderjit S Dhillon, and Sanjiv Kumar. 2022. Preserving in-context learning ability in large language model fine-tuning. *arXiv preprint arXiv:2211.00635*.

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023a. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Yue Wu, So Yeon Min, Shrimai Prabhumoye, Yonatan Bisk, Ruslan Salakhutdinov, Amos Azaria, Tom Mitchell, and Yuanzhi Li. 2023b. Spring: Gpt-4 out-performs rl algorithms by studying papers and reasoning. *arXiv preprint arXiv:2305.15486*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. 2023. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*.

Victor Zhong, Austin W Hanjie, Sida I Wang, Karthik Narasimhan, and Luke Zettlemoyer. 2021. Silg: The multi-environment symbolic interactive language grounding benchmark. *arXiv preprint arXiv:2110.10661*.

Appendices

A Pseudo Code for AdaRefiner

Algorithm 1 Pseudo Code for AdaRefiner

Init: Policy π; Buffer $\mathcal{B}$; Supervised fine-tuning (SFT) buffer $\mathcal{D}$; LLM generation interval N_{gen}; SFT interval N_{sft}.
$o_0 \leftarrow \texttt{env.reset}(),\ l_0 \leftarrow 0$
for $t = 0, 1, \ldots$ **do**
    // generate with Adapter LM and LLM
    if $t \,\%\, N_{\text{gen}} = 0$ **then**
        $c_t \leftarrow \mathcal{M}_{\text{a}}(\texttt{prompt}_{\texttt{a}}(\mathcal{B}_t, l_t))$
        $g_t \leftarrow \mathcal{M}_{\text{d}}(\texttt{prompt}_{\texttt{d}}(\mathcal{B}_t, c_t))$
    else
        $c_t \leftarrow c_{t-1},\ g_t \leftarrow g_{t-1}$
    end if
    // interact with the environment
    $a_t \sim \pi(a_t | o_t, f_{\text{emb}}(g_t))$
    $o_{t+1} \leftarrow \texttt{env.step}(a_t)$
    // update buffer and policy
    $\mathcal{B}_{t+1} \leftarrow \mathcal{B}_t \cup (o_t, a_t, o_{t+1}, r_t, g_t)$
    $\pi_{t+1} \leftarrow \texttt{RL\_Update}(\pi_t, \mathcal{B}_{t+1})$
    // update SFT buffer
    $l_{t+1} \leftarrow \cos(f_{\text{emb}}(g_t), f_{\text{emb}}(\tau)),\ \tau \sim \mathcal{B}_{t+1}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup [\texttt{prompt}_{\texttt{a}}(\mathcal{B}_t, l_{t+1}), c_t]$
    // SFT Adapter LM (with interval N_{sft})
    if $t \,\%\, N_{\text{sft}} = 0$ **then**
        $\texttt{SFT}(\mathcal{M}_{\text{a}}; \mathcal{D})$
    end if
end for

B Implementation Details

B.1 RL Algorithm

We use the classic PPO algorithm for policy learning in AdaRefiner, and the hyperparameters are shown in Table 5. It is worth noting that AdaRefiner can be flexibly combined with various RL algorithms and is not limited to PPO.

Hyperparameter	Value
policy learning rate	7e-4
update epoch	16
γ	0.97
ε	1e-8
clip ratio	0.1
optimizer	`Adam`

Table 5: Hyperparameters for PPO.

B.2 Adapter LM

We use open-source Llama2-7B weight as initial weight for the Adapter LM. In order to reduce computational resources and time consumption, we perform 4-bit quantization on it. The SFT parameters of the Adapter LM are shown in Table 6.

Hyperparameter	Value
quant type	`nf4`
learning rate	2e-4
batch size	4
gradient accumulation step	1
weight decay	1e-3
max grad norm	0.3
warmup ratio	0.3
lora alpha	16
lora dropout	0.1
lora r	64
N_{gen} (w/ GPT3.5)	10
N_{gen} (w/ GPT4)	20
N_{sft}	1e3

Table 6: Hyperparameters for Supervised Fine-Tuning.

B.3 Decision LLM

We call the API interfaces of OpenAI's `gpt-4` and `gpt-3.5-turbo` models. The API parameters used are shown in Table 7.

Hyperparameter	Value
temperature	0.5
top_p	1.0
max_tokens	100

Table 7: Hyperparameters for LLM.

B.4 Text Embedding

For text embedding, we choose the open-source `paraphrase-MiniLM-L6-v2` model as the encoder.

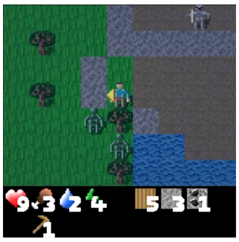
Case	Description	Explanation
	The agent tends to place stones between itself and monsters to avoid combat at night (the number of monsters will increase).	Frequent combats are not conducive to maintaining health and can delay other tasks such as resource collection. Therefore, the agent chooses to avoid combat at the appropriate time.
	The agent does not immediately place a workbench to craft tools and unlock achievements when it has abundant resources, but instead places the workbench when moving to resource-rich areas.	Placing the workbench in resource-rich areas can reduce the distance between collecting resources and crafting items, thus improving efficiency.

Table 8: Case study on agent behaviors grounded in common sense. These behaviors demonstrate the ability of the Adapter LM in uncovering human knowledge behind LLMs.

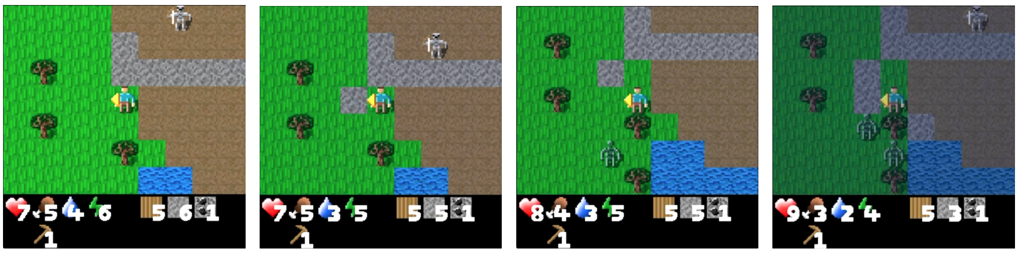


Figure 6: Case details of avoiding combat.

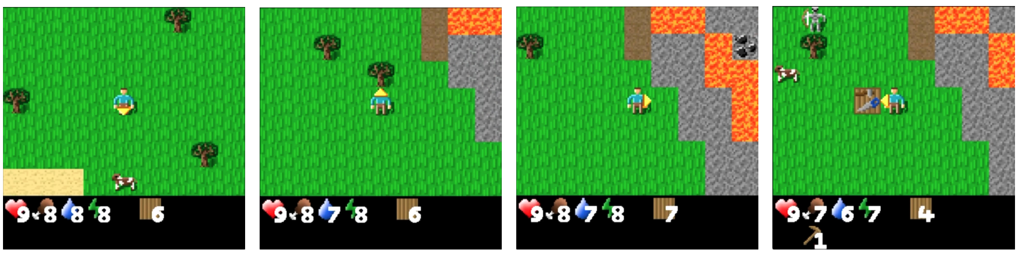


Figure 7: Case details of resource planning.

C Agent Behaviors Grounded in Common Sense

As discussed in Section 4.5, the policy trained by AdaRefiner exhibits behaviors like avoiding combat. Although this may result in a partial performance decrease for the achievements"Defeat Skeleton" and "Defeat Zombie", it could be more advantageous for survival and better completion of other tasks. In this sense, AdaRefiner demonstrates behaviors that align with human common sense. We further analyze additional replays and find other cases of human-like behavior in the policy trained by AdaRefiner, as shown in Table 8.

In the two cases, AdaRefiner demonstrates behaviors such as using stones to block monsters and extend survival time, as well as placing workbenches in resource-rich areas for more efficient resource utilization. These behaviors are not observed or reported in other baselines or in the version of AdaRefiner w/o Adapter LM. This further demonstrates that the Adapter LM can better capture the agent's learning ability and uncover common-sense knowledge behind LLMs, prompting them to provide more useful and reasonable guidance for better decision-making.

C.1 Details of Avoiding Combat

As shown in Figure 6, it is approaching night and the number of monsters is increasing. The agent starts early to strategically place stones in suitable terrain, successfully building a shelter that can keep the monsters outside and extend its survival time.

C.2 Details of Resource Planning

As shown in Figure 7, even though the agent has enough wood to make a workbench, its observations do not reveal abundant resources. Therefore, instead of rushing to make a workbench, it waits until more resources are discovered before making one nearby. This strategy can optimize the efficiency of resource collecting and item crafting.

D Full Prompt Details

In the following, we provide detailed prompts as well as corresponding example outputs for different methods.

D.1 Example for Adapter LM

System message:

```
You are a professional game
analyst. A player is playing a
game similar to Minecraft.
Available actions are:
<move_left, move_right, move_up,
move_down, do, sleep, place_stone
, place_table, place_furnace,
place_plant, make_wood_pickaxe,
make_stone_pickaxe,
make_iron_pickaxe,
make_wood_sword, make_stone_sword
, make_iron_sword>.

You will get the player's
observation, status information,
and its comprehension score of
language guidance (between 0 and
1). You are collaborating with
another analyst, and you will be
asked to provide concise
summaries and suggestions about
this player.
```

Example prompt:

```
Player sees: <grass, water, cow>

Player status: <7 health, 5 food,
 6 drink, 4 energy>

Past action: <sleep>

Past sub-goals:
- eat cow
- collect stone
- place stone

Comprehension score: <0.131>

Analyze the environment and the
player's understanding capability
,then generate concise summaries
and suggestions about this player
.
```

Example output:

```
The player seems to struggle with
 understanding past sub-goals,
possibly indicating an early
stage in the learning process. To
 help the agent learn the skill
of eating cow more quickly,
```

```
please provide more detailed
guidance.
```

D.2 Example for Decsion LLM

System message:

```
You are a professional game
analyst. A player is playing a
game similar to Minecraft.
Available actions are:
<move_left, move_right, move_up,
move_down, do, sleep, place_stone
, place_table, place_furnace,
place_plant, make_wood_pickaxe,
make_stone_pickaxe,
make_iron_pickaxe,
make_wood_sword, make_stone_sword
, make_iron_sword>.

You will get analysis about this
player from another analyst, and
you will be asked to provide the
next sub-goals for this player.
```

Example prompt:

```
Player sees: <grass, water, cow>

Player status: <7 health, 5 food,
 6 drink, 4 energy>

Past action: <sleep>

Past sub-goals:
- eat cow
- collect stone
- place stone

Analysis: <The player seems to
struggle with understanding past
sub-goals, possibly indicating an
 early stage in the learning
process. To help the agent learn
the skill of eating cow more
quickly, please provide more
detailed guidance.>

Based on the provided information
, suggest 3 sub-goals that the
player should accomplish next.
```

Example output:

```
find cow, move to cow, eat cow
```

D.3 Example for Decsion LLM in *AdaRefiner w/o Adapter LM*

System message:

```
You are a professional game
analyst. A player is playing a
game similar to Minecraft.
Available actions are:
<move_left, move_right, move_up,
move_down, do, sleep, place_stone
, place_table, place_furnace,
place_plant, make_wood_pickaxe,
make_stone_pickaxe,
make_iron_pickaxe,
make_wood_sword, make_stone_sword
, make_iron_sword>.

You will get necessary
information and player's
comprehension score of language
guidance (between 0 and 1). You
will be asked to provide the next
 sub-goals for this player.
```

Example prompt:

```
Player sees: <grass, tree, water>

Player status: <6 health, 7 food,
 3 drink, 1 energy>

Past action: <move_up>

Past sub-goals:
- eat cow
- collect stone
- place stone

Comprehension score: <0.165>

Based on the provided information
, suggest 3 sub-goals that the
player should accomplish next.
```

Example output:

```
collect stone, make stone sword,
make stone pickaxe
```

D.4 Example for Vanilla GPT-4

System message:

```
You are a professional game
analyst. A player is playing a
game similar to Minecraft.
Available actions are:
<move_left, move_right, move_up,
move_down, do, sleep, place_stone
, place_table, place_furnace,
place_plant, make_wood_pickaxe,
make_stone_pickaxe,
make_iron_pickaxe,
make_wood_sword, make_stone_sword
, make_iron_sword>.

You will get necessary
information and player's
comprehension score of language
guidance (between 0 and 1). You
will be asked to provide the next
 action for this player. You are
only allowed to choose the
available action.
```

Example prompt:

```
You see (object with coordinate)
<cow(2,2), tree(3,1), ...>

Player status: <8 health, 8 food,
 8 drink, 6 energy>

Past action: <make_wood_pickaxe>

Comprehension score: <0.275>

Based on the provided information
, provide the next action for
this player.
```

Example output:

```
move_right
```

D.5 Example for Reflexion

```
Environment: You are playing a
game similar to Minecraft.
Available actions are
<move_left, move_right, move_up,
move_down, do, sleep, place_stone
, place_table, place_furnace,
place_plant, make_wood_pickaxe,
make_stone_pickaxe,
make_iron_pickaxe,
make_wood_sword, make_stone_sword
, make_iron_sword>.

Task: Unlock following
achievements
<Collect Coal, Collect Diamond,
Collect Drink, Collect Iron,
Collect Sapling, Collect Stone,
Collect Wood, Defeat Skeleton,
Defeat Zombie, Eat Cow, Eat Plant
, Make Iron Pickaxe, Make Iron
Sword, Make Stone Pickaxe, Make
Stone Sword, Make Wood Pickaxe,
Make Wood Sword, Place Furnace,
Place Plant, Place Stone, Place
Table, Wake Up>

You see (object with coordinate)
<water(-1,2), cow(2,3), cow
(-3,-1), ...>

Your status:
<9 health, 3 food, 2 drink, 6
energy>
```

```
think: Considering the current
status, I need to find some food
and drink.

think: There are water at (-1,2),
 I need to move to it.

action: move_left
```

```
You see (object with coordinate)
<water(0,2), cow(1,3), cow(-2,-1)
, ...>

Your status: <9 health, 3 food, 2
 drink, 6 energy>
```

```
think: Now I move one step to the
 left, my status has not changed,
 water is at (0,2). Next, I need
to move up.

action: move_up
```

D.6 Example for ReAct

```
You are playing a game similar to
 Minecraft. Available actions are
<move_left, move_right, move_up,
move_down, do, sleep, place_stone
, place_table, place_furnace,
place_plant, make_wood_pickaxe,
make_stone_pickaxe,
make_iron_pickaxe,
make_wood_sword, make_stone_sword
, make_iron_sword>.

Your task is to: Unlock following
 achievements
<Collect Coal, Collect Diamond,
Collect Drink, Collect Iron,
Collect Sapling, Collect Stone,
Collect Wood, Defeat Skeleton,
Defeat Zombie, Eat Cow, Eat Plant
, Make Iron Pickaxe, Make Iron
Sword, Make Stone Pickaxe, Make
Stone Sword, Make Wood Pickaxe,
Make Wood Sword, Place Furnace,
Place Plant, Place Stone, Place
Table, Wake Up>

You see (object with coordinate)
<tree(1,-1), cow(1,4), ...>

Your status: <9 health, 7 food, 8
 drink, 7 energy>
```

```
think: My status is good, I need
to collect some wood to make
tools. First I need to move to
the tree at (1,-1).

actions: move_right, move_down
```

E Compute Resource Details

The hardware resources we used are listed in Table 9. For each seed, the average GPU running speed is approximately 30K steps/hour.

CPU	GPU	RAM
Intel Xeon 8280@2.7GHz	Nvidia A100 (40GB)	256GB
Intel I9-12900K@3.2GHz	Nvidia RTX 3090 (24GB)	128GB

Table 9: Computational resources for our experiments.

F Licenses

In our code, we have used the following libraries which are covered by the corresponding licenses:

- Crafter (MIT license)
- OpenAI GPT (CC BY-NC-SA 4.0 license)
- Llama 2 (Llama 2 license)
- SentenceTransformer (Apache-2.0 license)