Improving Retrospective Language Agents via Joint Policy Gradient Optimization

Xueyang Feng[1], **Bo Lan**[2], **Quanyu Dai**[3], **Lei Wang**[1], **Jiakai Tang**[1],
Xu Chen[1*], **Zhenghua Dong**[3], **Ji-Rong Wen**[1]
[1]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
[2] School of Software and Microelectronics, Peking University, Beijing, China
[3] Huawei Noah's Ark Lab, China
`{xueyangfeng, xu.chen}@ruc.edu.cn`

Abstract

In recent research advancements within the community, large language models (LLMs) have sparked great interest in creating autonomous agents. However, current prompt-based agents often heavily rely on large-scale LLMs. Meanwhile, although fine-tuning methods significantly enhance the capabilities of smaller LLMs, the fine-tuned agents often lack the potential for self-reflection and self-improvement. To address these challenges, we introduce a novel agent framework named RetroAct, which is a framework that jointly optimizes both task-planning and self-reflective evolution capabilities in language agents. Specifically, we develop a two-stage joint optimization process that integrates imitation learning and reinforcement learning, and design an off-policy joint policy gradient optimization algorithm with imitation learning regularization to enhance the data efficiency and training stability in agent tasks. RetroAct significantly improves the performance of open-source models, reduces dependency on closed-source LLMs, and enables fine-tuned agents to learn and evolve continuously. We conduct extensive experiments across various testing environments, demonstrating RetroAct has substantial improvements in task performance and decision-making processes.

1 Introduction

Achieving independent, autonomous agents capable of thinking, reasoning, and dynamically interacting with their environment has long been a fundamental goal for researchers in the field of artificial intelligence. In recent years, with the emergence of the powerful capabilities of Large Language Models (LLMs) (Zhao et al., 2023), researchers have started to utilize these models in building advanced intelligent agents (Wang et al., 2024; Yao et al., 2023a; Shinn et al., 2024). These agents demonstrate advanced capabilities in understanding complex language inputs, engaging in intricate planning (Ahn et al., 2022; Wei et al., 2022; Wang et al., 2022; Huang et al., 2022; Yao et al., 2024) and utilizing tools (Qin et al., 2023; Schick et al., 2024; Shen et al., 2023; Kong et al., 2023).

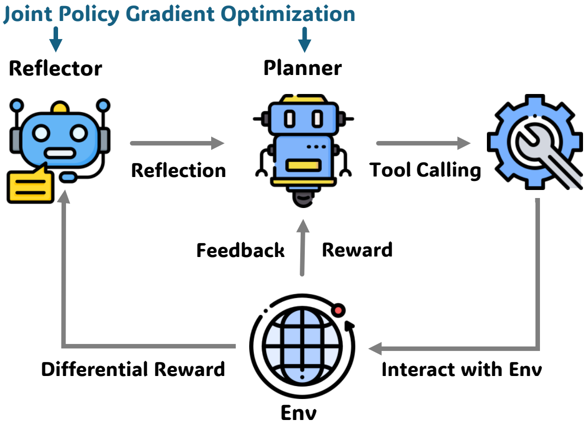

Figure 1: Overview of retrospective language agent. The planner analyzes task requirements, calls external tools, and gathers feedback. If planning fails, the reflector intervenes to adjust the strategy until the issue is resolved. Through joint strategy optimization, RetroAct continually enhances both the planner and reflector to tackle complex tasks more effectively.

Since LLMs are not initially intended for agent tasks, they need to be adjusted to carry out such tasks efficiently. Currently, there are two primary adaptation paradigms: (1) *Prompt-based Agent*: In this paradigm, LLMs leverage their in-context learning abilities to adapt to new environments (Brown et al., 2020). Many researchers have designed self-reflection mechanisms, allowing agents to explicitly reflect on feedback from the environment and incorporate these reflections into subsequent trials for iterative self-improvement (Shinn et al., 2024; Madaan et al., 2024; Yao et al., 2023b). However, these methods typically rely on large-scale LLMs, leading to substantial costs and delays. Moreover, most smaller LLMs often exhibit insuf-

---

[*] Corresponding Authors.

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 112–141
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

Name	Planner			Reflector		
	Prompt	IL	RL	Prompt	IL	RL
ReAct (Yao et al., 2023a)	✓	✗	✗	✗	✗	✗
Self-Refine (Madaan et al., 2024)	✓	✗	✗	✓	✗	✗
Reflexion (Shinn et al., 2024)	✓	✗	✗	✓	✗	✗
Retroformer (Yao et al., 2023b)	✓	✗	✗	✓	✓	✓
FireAct (Chen et al., 2023a)	✓	✓	✗	✗	✗	✗
ArCHer (Xi et al., 2024)	✓	✓	✓	✗	✗	✗
RetroAct	✓	✓	✓	✓	✓	✓

Table 1: Related work on Language Agent. Prompt, IL, RL stand for Prompt-based Method, Imitation Learning-based Method, and Reinforcement Learning-based Method.

ficient performance and robustness when deployed as agents (Chen et al., 2023a). (2) *Agent Fine-tuning*. This paradigm primarily includes using imitation learning to fine-tune smaller LLMs with expert datasets, enabling agents to learn specific tasks from the data (Zeng et al., 2023; Chen et al., 2023a), as well as employing reinforcement learning to allow agents to explore and learn on training sets (Zhou et al., 2024). However, these fine-tuned agents depend on the knowledge acquired through fine-tuning and lack the ability to continuously learn new information and self-improve in testing environments without updating parameters. As shown in Table 1, there is no research on how to simultaneously enhance an agent's task-planning abilities and self-reflection capabilities from failures, nor is there a comprehensive framework that combines IL and RL to jointly fine-tune these two capabilities of the agent.

In this work, we propose a novel agent framework called RetroAct, which jointly optimizes the task-planning and self-reflection capabilities of open-source LLMs. This approach eliminates the dependency on closed-source models during inference while retaining the ability for continuous reflection and evolution. Specifically, we developed a two-phase joint optimization process that integrates imitation learning and reinforcement learning. First, we use imitation learning to distill the planning and reflection capabilities of large-scale LLMs into smaller LLMs. Then, we propose an off-policy joint policy gradient optimization algorithm with imitation learning regularization to enhance data efficiency and training stability in the reinforcement learning process. During the joint optimization process, the planner and reflector can mutually facilitate each other and collectively improve the overall performance of the agent, showcasing the unique advantages of joint optimization.

We conduct experiments on three representative agent tasks: Complex Reasoning (HotpotQA (Yang et al., 2018)), Embodied Decision (ALFWorld (Shridhar et al., 2020)), and Interactive Programming (InterCode (Yang et al., 2023)), based on the *Llama-7b* and *Llama-13b* (Touvron et al., 2023). Through joint optimization, our model achieves performance improvements ranging from 22.3% to 348.3% on these tasks, attaining performance comparable to or exceeding that of ChatGPT (OpenAI, 2022). Moreover, we further validate the mutual promotion between the planner and reflector. We also demonstrate that a single model can concurrently learn both planning and reflection capabilities, albeit with a slight decrease in performance. Finally, we conduct systematic ablation studies to demonstrate the importance of each module in our approach.

To summarize, our contributions are shown in the following:

- We propose a language agent framework that jointly optimizes task-planning and self-reflective evolution capabilities. This framework enables agents built on open-source models to improve performance through fine-tuning and empowers them to continuously learn and adapt to the environment.
- We design an off-policy joint policy gradient algorithm with an imitation learning regularization term, which improves data efficiency and training stability.
- We validate the effectiveness of our proposed methods through extensive experiments across multiple representative testing environments, demonstrating substantial improvements in task performance and decision-making processes.

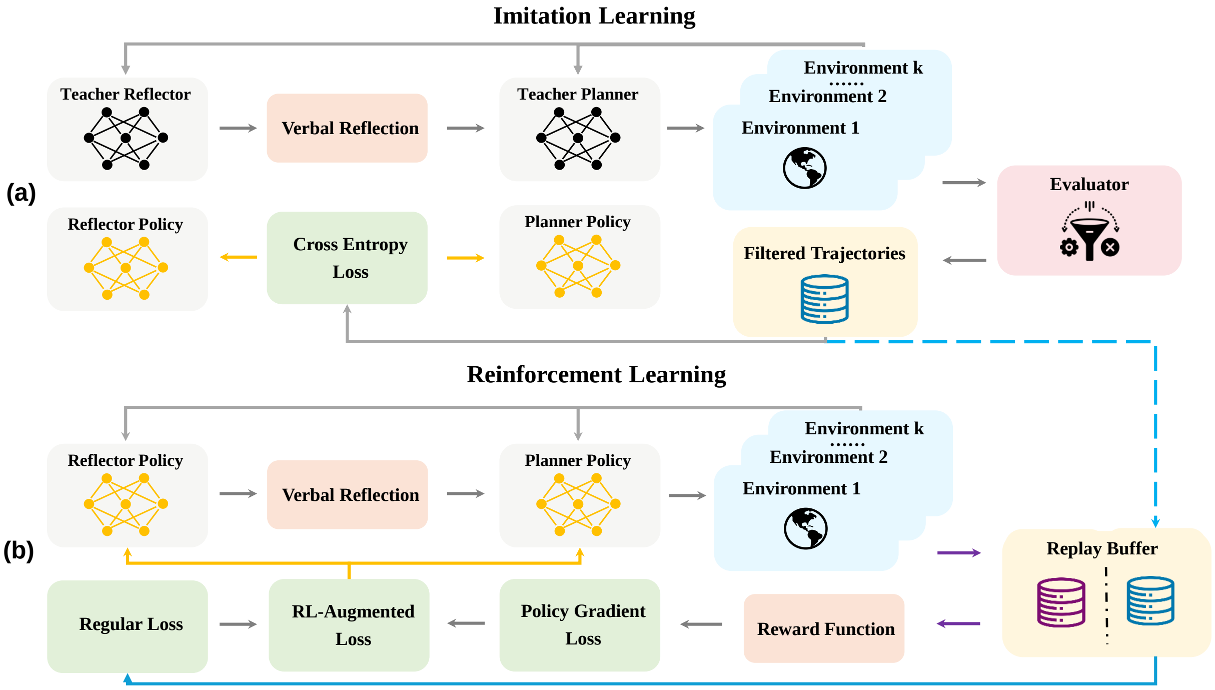

Figure 2: Schematic of Joint Policy Gradient Optimization for Retrospective Language Agent. Our approach is divided into two stages: (a) **Imitation Learning**: We use expert models to generate expert trajectories, employ evaluators to filter out these trajectories, and then use them to fine-tune the student models. (b) **Reinforcement Learning**: The planner and reflector are jointly optimized through the off-policy reinforcement learning algorithm with the imitation learning regularizer.

2 Approach

Existing methods for agent-tuning primarily rely on knowledge acquired during the fine-tuning process, lacking the ability for real-time self-reflection and self-improvement. To address these limitations, we propose RetroAct, a novel approach that jointly optimizes LLMs' task-planning and self-reflection capabilities. Our approach significantly enhances the performance of open-source LLMs through a combination of imitation learning (IL) and reinforcement learning (RL). This eliminates the dependency on large-scale LLMs in agent tasks and retains the potential to evolve in new environments without parameter updates. Figure 2 illustrates the overall framework of our RetroAct. In this section, we frame the process of LLM-based agent task-solving as a Markov Decision Process (MDP) and construct our method within this MDP framework.

2.1 Preliminaries: LLM-based Agent for Task-Solving

In this study, we frame LLM-based agent task-solving within the Markov Decision Process (MDP) framework, represented as the tuple (S, A, P, O, R). Here, S denotes the set of states, A represents the available actions for the agent, $P : S \times A \times S$ defines transition probabilities between states given action, O is the environmental feedback, and $R : S \times A \to R$ is the reward function, which assigns values to actions taken in different states. The LLM-based agent $\pi(a|s)$ aims to choose actions that maximize rewards. Notably, $s \in S$, $a \in A$, and $o \in O$ are all represented in natural language. Typically, $r \in R$ is sparse, with values mostly zero except for specific states, where rewards are obtained upon the conclusion of particular trajectories, such as success or failure. With a slight abuse of notation, a typical execution trajectory, consisting of n steps, is denoted as $\tau = \{s_0, a_0, o_0, \ldots, s_n, a_n, o_n, r\}$.

2.2 Agent Architecture

The RetroAct agent architecture is comprised of two language model components: a planner LLM denoted as π and a reflector LLM denoted as μ. The planner directly interacts with the environment and generates thoughts and actions, while the reflector generates verbal reflections to help the planner iteratively improve the plan. It is similar to the Reflexion (Shinn et al., 2024).

Planner Model The *planner model* π resembles a policy model in reinforcement learning, employed to generate an action a_t at a specified step t and given state s_t. The state s_t is textual, composed of task prompts, environmental descriptions, and historical interactions. The action a_t comprises two distinct components: *Thought* and *Action* (Yao et al., 2023a). *Thought* denotes the agent's explicit thought process about the task; *Action* refers to the actual interactive responses of the agent, such as utilizing tools and executing tasks. The planner can be formulated as follows:

$$a_t \sim \pi(\cdot|s_t) \tag{1}$$

Reflector Model The *reflector model* μ resembles a reward model in reinforcement learning. When facing complex tasks, LLM-based agent often engages in multiple interactions with the environment to accomplish them. Tasks may fail either due to execution errors or upon encountering specific constraints. When the agent fails in the k-th ($0 \leq k \leq n$) trial, the unsuccessful trajectory τ^k is presented to the reflector μ to produce verbal reflections denoted as f^k. These verbal reflections act as semantic gradient signals to improve the planner model without parameter updates. The reflector can be formulated as:

$$f^k \sim \mu(\cdot|\tau^k) \tag{2}$$

The initial state of the new trial is adjusted based on the initial state of the previous trial and the received feedback, i.e., $s_0^{k+1} = s_0^k + f^k$. Consequently, the trajectory for trial $k+1$ becomes $\tau^{k+1} = \{s_0^{k+1}, a_0^{k+1}, o_0^{k+1}, \ldots, s_n^{k+1}, a_n^{k+1}, o_n^{k+1}, r^{k+1}\}$. The overall goal is to iteratively refine the policy model π through the feedback provided by μ, aiming to optimize decisions and maximize cumulative rewards across trials.

2.3 Imitation Learning

Despite the ability of few-shot examples and self-reflection mechanisms to adapt agents to particular tasks, these methods heavily rely on more powerful LLMs and often underperform with smaller LLMs. To overcome this issue, we first create expert datasets using large-scale LLMs. Then, we use these datasets to fine-tune the planner and reflector components of smaller LLMs. This approach enables smaller LLMs to enhance their performance by learning from the demonstrations of the larger LLMs, facilitating rapid adaptation to new tasks.

Expert Data Collection We utilize a powerful LLM, Mixtral-8*7b (Jiang et al., 2024) with few-shot examples, serving as the teacher agents, which is denoted as π_{expert} and μ_{expert}. These teacher agents engage extensively with various environments in the training sets to generate expert trajectories for fine-tuning. We implement different rule-based evaluators for planner and reflector trajectories across multiple datasets (see Appendix C.5 for more information). These evaluators help us filter out positive examples from the expert data to create fine-tuning datasets D^{IL}_{planner} and $D^{IL}_{\text{reflector}}$.

Imitation Learning The training objective is to closely align the distribution $\pi(a|s)$ of the planner and the distribution $\mu(f|\tau)$ of the reflector with the expert model's action distribution. This optimization objective can be expressed as:

$$\mathcal{L}^{\text{IL}}_{\text{planner}} = \mathbb{E}_{s\sim D^{\text{IL}}_{\text{planner}}}\left[-\pi_{\text{expert}}(a|s)\log \pi_\theta(a|s)\right], \tag{3}$$

$$\mathcal{L}^{\text{IL}}_{\text{reflector}} = \mathbb{E}_{s\sim D^{\text{IL}}_{\text{reflector}}}\left[-\mu_{\text{expert}}(f|\tau)\log \mu_\phi(f|\tau)\right]. \tag{4}$$

2.4 Joint Policy Gradient Optimization

Although IL has demonstrated notable efficacy, it still has several limitations. Firstly, IL relies on expert demonstrations, meaning the agent can only learn behaviors in datasets, making it difficult to surpass expert performance. Secondly, IL lacks the ability to learn from broader reward signals, such as negative feedback, which results in suboptimal outcomes when dealing with complex tasks. To this end, we use a joint policy gradient algorithm to simultaneously optimize both the planner and reflector. Leveraging trial-and-error and broad reward signals to learn environmental information, the planner and reflector could surpass expert demonstrations and achieve superior policies.

Reward Shaping The reward function for the planner R_π is the environment-provided reward R_τ, which is defined according to the datasets, e.g., task completion rate (more details are in Appendix C.5). Moreover, following the Retroformer (Yao et al., 2023b), we design a reward function R_μ for the reflector, defined as the difference in environmental rewards between the trial conducted after reflection by the reflector and the last failed trial. In conclusion, the reward functions of the planner and reflector in the k-th trial are as follows:

$$R_{\pi_k} = R_{\tau_k}, \tag{5}$$

$$R_{\mu_k} = (R_{\tau_{k+1}} - R_{\tau_k}). \tag{6}$$

Off-Policy Joint Policy Gradient Optimization In LLM-based agent tasks, online RL algorithms like Policy Gradient often suffer from poor performance due to low sample efficiency and training instability, while the training process is further hindered by high inference costs and significant latency. To address these issues, we design an off-policy joint policy gradient algorithm inspired by PPO-Clip (Schulman et al., 2017). Specifically, we construct replay buffers to store historical trajectories and use these data in each iteration to perform off-policy optimization. The optimization objective is formulated as follows:

$$\mathcal{L}^{\text{RL}}_{\text{planner}} = \mathbb{E}_{s\sim D^{\text{RL}}_{\text{planner}}}\left[-\sum_{a} w_{\pi}(s,a)R_{\pi}(s,a)\right], \tag{7}$$

$$\mathcal{L}^{\text{RL}}_{\text{reflector}} = \mathbb{E}_{s\sim D^{\text{RL}}_{\text{reflector}}}\left[-\sum_{a} w_{\mu}(\tau,f)R_{\mu}(\tau,f)\right], \tag{8}$$

where w_π and w_μ are importance sampling weights that adjust for the difference between the policies under parameter updates and the behavior policies, π_{beh} and μ_{beh}, which generate the data. These weights help mitigate the distribution shift in off-policy data (Chen et al., 2021). Given that importance sampling weights can introduce significant variance (Schulman et al., 2017), we mitigate this by applying a clipping function, which limits the importance sampling term to the interval $\{1-\epsilon, 1+\epsilon\}$:

$$w_{\pi}(s,a) = \text{Clip}\left(\frac{\pi_{\theta}(a|s)}{\pi_{\text{beh}}(a|s)}, 1-\epsilon, 1+\epsilon\right)$$

$$w_{\mu}(\tau,f) = \text{Clip}\left(\frac{\mu_{\phi}(f|\tau)}{\mu_{\text{beh}}(f|\tau)}, 1-\epsilon, 1+\epsilon\right)$$

Notably, we adopt a related technique (Chen et al., 2024, 2021) to control the optimization speed by adjusting the gradient clipping coefficient, ensuring it remains within a reasonable range. When the optimization speed is too fast or too slow, the clipping coefficient helps regulate gradient changes, improving training stability. In contrast, standard PPO halts optimization when the discrepancy becomes too large, which can lead to the waste of a significant amount of samples in off-policy/offline scenarios. Additionally, the knowledge acquired during RL may conflict with the knowledge learned during IL, causing the model to forget previous knowledge and leading to a performance decrease. Therefore, we introduce a regularizer based on imitation learning to mitigate this effect and stabilize the training process. The resulting augmented RL loss functions for the planner and the reflector can be written as:

$$\mathcal{L}^{\text{RL}}_{\text{planner, augmented}} = \mathcal{L}^{\text{RL}}_{\text{planner}} + \lambda_{\pi}\mathcal{L}^{\text{IL}}_{\text{planner}}, \tag{9}$$

$$\mathcal{L}^{\text{RL}}_{\text{reflector, augmented}} = \mathcal{L}^{\text{RL}}_{\text{reflector}} + \lambda_{\mu}\mathcal{L}^{\text{IL}}_{\text{reflector}}, \tag{10}$$

where λ_π and λ_μ are regularization weights that balance the influence of the RL objectives and the imitation learning objectives.

3 Experiments

In this section, we systematically explore the performance of our proposed method. Our experimental design revolves around four key questions: **Q1**: How does RetroAct perform compared with the existing prompt-based and fine-tuning method? What are the benefits of joint optimization in improving the learning process of both the planner and reflector? **Q2**: How about integrating the planner and reflector within the same open-source model? Can the model effectively learn both planning and self-reflection capabilities simultaneously? **Q3**: How do the optimized planner and reflector respectively influence the behavior of RetroAct? **Q4&Q5**: How does the reinforcement learning process and imitation learning regularization impact RetroAct?

3.1 Environmental Settings

We select three representative agent environments: **(1) Complex Reasoning: HotpotQA** (Yang et al., 2018) is a multi-turn QA dataset. Following ReAct (Yao et al., 2023a) and ChatCoT (Chen et al., 2023b), we reconstruct the HotpotQA environments. The agent needs to call an external retriever based on SimCSE (Gao et al., 2021) multiple times to obtain the necessary information for answering complex questions. **(2) Embodied Decision: ALFWorld** (Shridhar et al., 2020) is a text-based environment designed to simulate real-world interactions through embodied agents. In this setting, agents are tasked with executing a sequence of natural language actions informed by surrounding environment feedback to accomplish complex goals. **(3) Interactive Coding: InterCode** (Yang et al., 2023) is a framework for evaluating language agents capable of interactive coding. In this work, we utilize the InterCode-SQL to thoroughly assess the agent's interactive SQL query generation capability. Moreover, We discuss in detail the selection criteria and configuration settings of the baselines and evaluation metrics in Appendix C.

Model	Method	HotpotQA			ALFWorld			InterCode			Avg
		IR	FR	AR	IR	FR	AR	IR	FR	AR	
Llama-7b	ReAct	39.7	39.7	39.7	8.96	8.96	8.96	14.24	14.24	14.24	21.15
	Reflexion	39.7	58.49	54.39	8.96	21.64	18.28	14.24	32.16	28.40	30.14
	SFT	55.58	55.58	55.58	72.39	72.39	72.39	30.67	30.67	30.67	52.65
	SFT+EI	58.22	58.22	58.22	68.66	68.66	68.66	34.86	34.86	34.86	53.91
	SFT+RL	60.70	60.70	60.70	80.60	80.60	80.60	39.42	39.42	39.42	60.24
	Ours	**60.92**	**71.51**	**67.90**	**82.84**	**97.01**	**93.28**	**41.12**	**54.17**	**51.46**	**68.91**
Llama-13b	ReAct	43.99	43.99	43.99	28.36	28.36	28.36	26.83	26.83	26.83	32.94
	Reflexion	43.99	62.39	59.73	28.36	46.27	39.70	26.83	43.00	38.79	43.01
	SFT	58.99	58.99	58.99	**79.10**	79.10	79.10	43.17	43.17	43.17	60.75
	SFT+EI	60.23	60.23	60.23	71.64	71.64	71.64	**47.01**	47.01	47.01	59.63
	SFT+RL	**61.90**	61.90	61.90	77.61	77.61	77.61	42.25	42.25	42.25	60.59
	Ours	57.99	**70.11**	**66.58**	78.36	**91.04**	**87.39**	44.30	**61.83**	**58.38**	**68.11**
ChatGPT	ReAct	41.84	41.84	41.84	52.24	52.24	52.24	62.47	62.47	62.47	52.32
	Reflexion	41.84	68.61	62.12	52.24	79.10	71.04	62.47	69.00	68.63	63.34

Table 2: Experimental results on HotpotQA, ALFWorld, InterCode. **Avg** is the average accuracy of all tasks. **IR**, **FR**, and **AR** stand for initial reward, final reward, and average reward, respectively. The best results and second best results are **bold** and underlined, respectively.

3.2 Main Experiment on Multi-Agent (Q1)

In this section, we implement the planner and reflector as two different LLMs, thereby instantiating RetroAct as a multi-agent framework. We then compare it with baseline agents based on prompts and fine-tuning, respectively. Comparison of RetroAct agents with other baseline agents across three environments and two base LLMs are shown in Table 2. Overall, our method demonstrates significant advantages compared to baseline methods. Additionally, there is mutual facilitation between the reflector and the planner during the joint optimization process.

Baseline Analysis (1) For prompt-based methods, *ChatGPT* agents outperform *Llama* agents in all environments, demonstrating that agents based on open-source LLMs perform significantly worse than those based on closed-source LLMs, indicating that current open-source LLMs are not yet sufficient for handling complex agent tasks; (2) Reflexion-based agents significantly improve both average and final rewards, even with the same initial reward. Fine-tuning methods show clear advantages over prompt-based methods, underscoring the potential of self-reflection to enhance agent performance. However, current fine-tuning approaches rarely consider the joint optimization of task planning and self-reflection capabilities; (3) Benefiting from self-exploration, EI and RL methods, particularly in the 7b model, achieve better performance than SFT, with broader reward signals making RL algorithms more effective in agent tasks. Additionally, we demonstrate in the Appendix D.1 our specifically designed off-policy RL is better suited for agent tasks than the standard PPO algorithm.

RetroAct Result Compared to prompt-based baselines, our approach significantly enhances the agent performance based on the same size LLMs. More notably, RetroAct based on Llama-7b exceeds the reflexion agent based on ChatGPT by an average of 8%. The sustained superior performance of RetroAct demonstrates its effectiveness in enhancing task planning and self-reflection capabilities through the knowledge gained from fine-tuning. Compared to the best fine-tuning baseline methods, RetroAct achieves a 13.4% performance improvement on average. Specifically, existing fine-tuning baseline methods do not include a reflector and cannot continuously self-improve. As a result, after multiple iterations of trials and reflections, RetroAct shows significant advantages in both final rewards and average rewards. Furthermore, even in terms of initial rewards, RetroAct slightly outperforms the RL baseline that optimizes the planner alone. This result underscores the **mutual facilitation between the reflector and the planner during the joint optimization process**. Interestingly, our approach brings more significant improvements to lower-performing base LLMs. This could be attributed to the fact that larger base models have already developed more advanced planning and reflection abilities during the pre-training phase. As

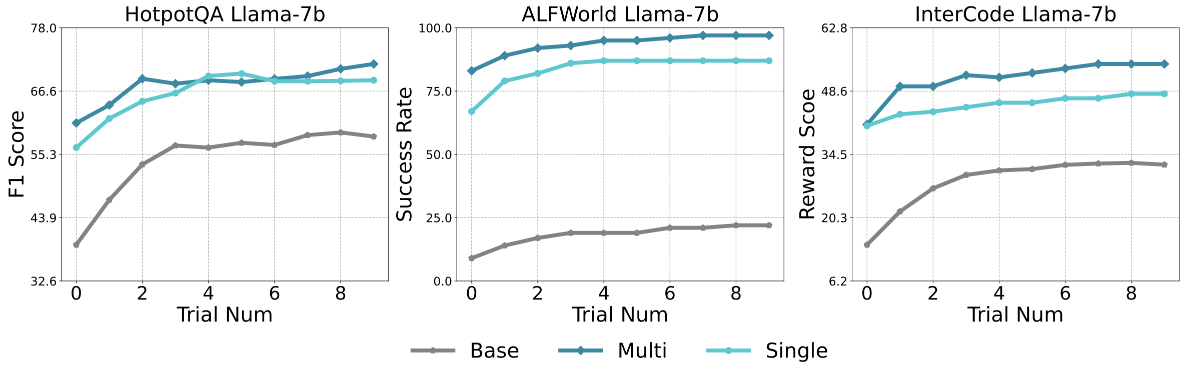

Figure 3: Multi-Agent vs Single Agent (Q2)

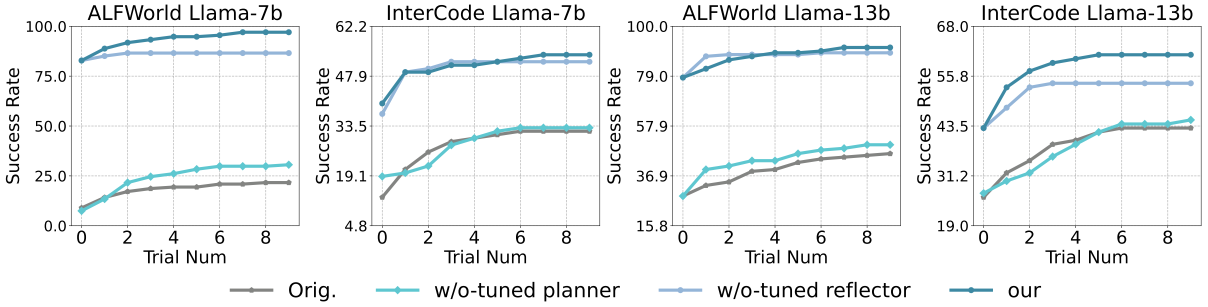

Figure 4: Effectiveness of Optimized Planner and Reflector (Q3)

a result, larger models derive less benefit from further training through supervised fine-tuning and reinforcement learning with expert or exploratory samples. This phenomenon is similar to what was observed in (Yuan et al., 2023).

3.3 Comparison of Multi-Agent and Single-Agent (Q2)

In the last experiment, we validate the effectiveness of instantiating RetroAct as a multi-agent framework. In this section, we will further explore whether a single agent can simultaneously learn task-planning and self-reflection through fine-tuning. The results in Figure 3 show that in HotpotQA, the single agent has almost no performance loss. In ALFWorld and InterCode, although our method is still effective, it has about 10% performance loss compared to the multi-agent method. We posit that these findings are due to the difference in **trajectories consistency** between the planner and reflector. In HotpotQA, there is a high similarity between the planning and reflection tasks, as both involve standard natural language processing tasks. Consequently, the single agent is capable of effectively managing these tasks simultaneously. However, the ALFWorld and InterCode environments exhibit significant differences in task types: the planner in ALFWorld primarily utilizes specific natural language instruction sets, whereas the planner in InterCode involves SQL commands (we provide some trajectories in the appendix D.4, C.4). Meanwhile, reflection tasks in both environments are executed using conventional natural language. Compared to the multi-agent, the single-agent may lead to specific knowledge conflicts when learning task planning and self-reflection, resulting in decreased performance.

3.4 Effectiveness of Optimized Planner and Reflector (Q3)

In this section, we explore how the optimized planner and reflector respectively influence the behavior of RetroAct. Specifically, we separately removed the optimization from the planner and the reflector, which allowed us to study the interaction between the optimized planner and the unoptimized reflector, and vice versa. The experimental results are shown in Figure 4. Overall, removing the optimization from either the planner or the reflector negatively impacts performance. If the planner is not optimized, the agent's performance in the first trial will degrade to the base model level. Although the optimized reflector can generally help the planner improve through trial and error, this initial perfor-

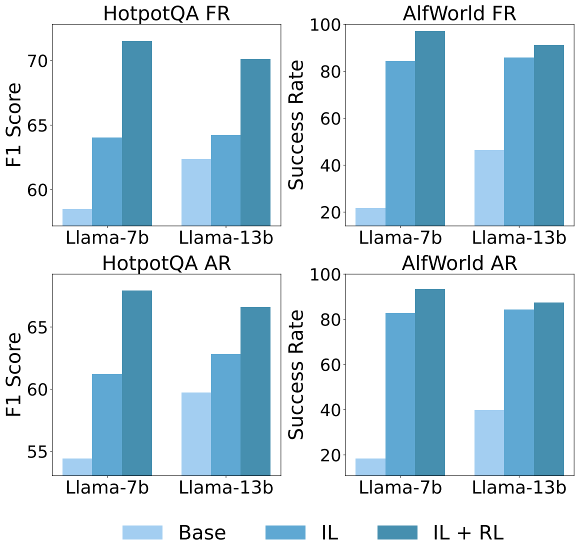

Figure 5: Effectiveness of Reinforcement Learning (Q4)

mance loss is significant and cannot be recovered through self-reflection. When the reflector is not optimized, the well-optimized planner performs well in the first trial. However, due to the reflector not being sufficiently compelling, the potential for performance improvement through self-reflection is less than with our jointly optimized agent.

Notably, compared to the base model, optimizing the planner alone is more efficient than optimizing the reflector. We attribute this to two main reasons. First, the optimization of the planner is a direct optimization of the tasks, making it more straightforward and effective. In contrast, the optimization of the reflector is essentially tuning the planner's prompts, which is less advantageous compared to directly updating the parameters. Second, there is an imbalance in the amount of data available for the planner and the reflector (we provide a comparison of effective data amounts for imitation learning in the Appendix C), leading to the planner often being better trained.

Therefore, we conclude that for smaller LLMs, simultaneously optimizing the planner and the reflector is optimal. Additionally, optimizing the planner alone is more effective than optimizing the reflector alone. Optimizing the reflector alone is preferable when the planner cannot be optimized or the cost is too high. This finding supports the claims made in previous studies (Chen et al., 2023a; Yao et al., 2023b). To more specifically describe how the optimized planner and reflector work together to better complete tasks, we provide several cases in Appendix D.4.

λ	Initial Reward	Final Reward	Average Reward
0.0	52.72	67.71	64.15
1.0	56.52	68.60	66.24
2.0	55.41	63.19	62.67

Table 3: Hyperparameter Analysis on λ.

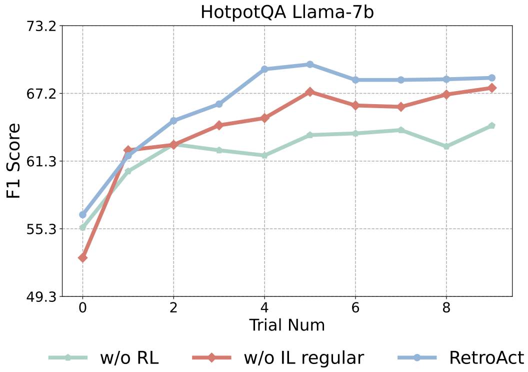

Figure 6: Effectiveness of Imitation Learning Regularization (Q5)

3.5 Effectiveness of Reinforcement Learning and Regularization (Q4&Q5)

In this section, we explore how each component of our proposed off-policy joint reinforcement learning impacts the overall performance of RetroAct.

Effectiveness of RL phase Overall, as shown in Figure 5, we find that our RL algorithm outperformed IL in three different environments. We attribute these performance improvements to the agent's ability in RL to discover optimal behaviors beyond the constraints of the imitation learning dataset through exploration. Additionally, the performance improvements are more pronounced in HotpotQA and InterCode, likely because these datasets provide broader reward signals, whereas the binary reward signals in ALFWorld limit the effectiveness of RL. These findings underscore the importance of reward signal design in RL, suggesting that broader reward signals can better aid the agent in learning environmental information.

Effectiveness of IL Regularization By setting λ to 0, we remove the imitation learning regularization from the RL phase. As shown in Figure 6, eliminating this regularization objective led to some performance degradation, which demonstrates the role of regularization to retain the knowledge acquired during the imitation learning phase. Moreover, using the RL objective independently still resulted in acceptable performance, confirming the effectiveness of the RL objective. Additionally, we analyze the effect of the hyperparameter λ, which controls the weight of the regularization term, under different values. As shown in Table 3, we find that setting the hyperparameter around 1.0 yields the best performance. When the hyperparameter value is too small, conflicts arise between the knowledge learned during the reinforcement learning (RL) phase and the knowledge acquired during the imitation learning (IL) phase, leading to performance degradation. On the other hand, when the hyperparameter value is too large, IL regularization dominates, causing the model to be overly constrained by the IL rules, thereby limiting the benefits of RL and reducing overall performance.

4 Conclusion

In this work, we introduce RetroAct, an agent framework that utilizes imitation learning and off-policy reinforcement learning to jointly optimize the task-planning and self-reflective capabilities of open-source LLMs. RetroAct significantly enhances the performance of these models, reducing the reliance on closed-source LLMs in agent tasks. We conduct extensive experiments across various agent environments, demonstrating the substantial improvements over existing baselines. Future work may involve designing more complex reinforcement learning systems, such as training reward models for the planner and reflector to provide more fine-grained reward signals.

5 Ethical Considerations and Limitations

Ethical Considerations Our work leverages the powerful capabilities of Large Language Models (LLMs) to build advanced intelligent agents. While the potential benefits of these advancements are substantial, it is crucial to consider their broader impact and ethical implications. We summarize the broader impact and ethics statement of our research as follows:

- **Bias and Fairness**: We utilize reinforcement learning to enable open-source LLMs to learn and adapt to domain-specific agent tasks. If there are biases and discrimination in specific tasks, the agent may exacerbate these issues. Therefore, ensuring fairness in the application field is vitally important.

- **Safety**: Despite significant efforts to align LLMs with safety standards, their deployment as agents raises additional safety concerns. Agents must avoid invoking harmful tools when interacting with external systems. Implementing constrained reinforcement learning can help ensure that agents do not engage in harmful actions, thereby enhancing their safety and reliability.

- **Data Security**: Although we have made every effort to review and verify the data we release, some security concerns may still remain. Ensuring the integrity and security of the data is crucial to prevent potential misuse or vulnerabilities.

Limitation Our current approach is the reliance on rewards directly provided by the environment. While this method is straightforward and effective to a certain extent, it may not always provide the most granular and informative feedback necessary for optimal performance. A more sophisticated approach would involve training separate reward models for the planner and the reflector, which could offer more detailed and tailored reward information for each component. This could potentially enhance the agent's ability to fine-tune its actions and reflections, leading to better overall performance and adaptability in complex tasks. We encourage future work to explore the development and integration of these reward models to further improve the effectiveness of our proposed approach.

Acknowledgement

This work is supported in part by National Natural Science Foundation of China (No. 62422215 and No. 62472427), Beijing Outstanding Young Scientist Program NO.BJJWZYJH012019100020098, Intelligent Social Governance Platform, Major Innovation & Planning Interdisciplinary Platform for the “DoubleFirst Class” Initiative, Renmin University of China, Public Computing Cloud, Renmin University of China, fund for building world-class universities (disciplines) of Renmin University of China, Intelligent Social Governance Platform, Huawei Innovation Research Programs. We gratefully acknowledge the support from Mindspore[1], CANN(Compute Architecture for Neural Networks) and Ascend AI Processor used for this research.

References

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. Preprint, arXiv:2005.14165.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023a. Fireact: Toward language agent fine-tuning. arXiv preprint arXiv:2310.05915.

Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi. 2021. Top-k off-policy correction for a reinforce recommender system. Preprint, arXiv:1812.02353.

Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Chatcot: Tool-augmented chain-of-thought reasoning on\\chat-based large language models. arXiv preprint arXiv:2305.14323.

Zhipeng Chen, Kun Zhou, Wayne Xin Zhao, Junchen Wan, Fuzheng Zhang, Di Zhang, and Ji-Rong Wen. 2024. Improving large language models via fine-grained reinforcement learning with minimum editing constraint. arXiv preprint arXiv:2401.06081.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. arXiv preprint arXiv:2104.08821.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. arXiv preprint arXiv:2309.17452.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024a. Teaching large language models to reason with reinforcement learning. arXiv preprint arXiv:2403.04642.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024b. Teaching large language models to reason with reinforcement learning. Preprint, arXiv:2403.04642.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2022. Large language models are reasoning teachers. arXiv preprint arXiv:2212.10071.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. Preprint, arXiv:2106.09685.

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

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. Preprint, arXiv:2401.04088.

Yilun Kong, Jingqing Ruan, Yihong Chen, Bin Zhang, Tianpeng Bao, Shiwei Shi, Guoqing Du, Xiaoru Hu, Hangyu Mao, Ziyue Li, et al. 2023. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. arXiv preprint arXiv:2311.11315.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient

---

[1] `https://www.mindspore.cn`

memory management for large language model serving with pagedattention. Preprint, arXiv:2309.06180.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. arXiv preprint arXiv:2305.20050.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Systems, 36.

OpenAI. 2022. Chatgpt.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch via self-planning. arXiv preprint arXiv:2401.05268.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. 2023. Tool learning with foundation models. arXiv preprint arXiv:2304.08354.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36.

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face. In Advances in Neural Information Processing Systems.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 36.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. arXiv preprint arXiv:2010.03768.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. Preprint, arXiv:2403.02502.

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

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6):1–26.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Zhiheng Xi, Wenxiang Chen, Boyang Hong, Senjie Jin, Rui Zheng, Wei He, Yiwen Ding, Shichun Liu, Xin Guo, Junzhe Wang, Honglin Guo, Wei Shen, Xiaoran Fan, Yuhao Zhou, Shihan Dou, Xiao Wang, Xinbo Zhang, Peng Sun, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Training large language models for reasoning through reverse curriculum reinforcement learning. Preprint, arXiv:2402.05808.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. Preprint, arXiv:2306.14898.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. arXiv preprint arXiv:1809.09600.

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024. React meets actre: Autonomous annotations of agent trajectories for contrastive self-training. arXiv preprint arXiv:2403.14589.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. Advances in Neural Information Processing Systems, 36.

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

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. 2023b. Retroformer: Retrospective large language agents with policy gradient optimization. arXiv preprint arXiv:2308.02151.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. Preprint, arXiv:2308.01825.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. arXiv preprint arXiv:2310.12823.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. arXiv preprint arXiv:2303.18223.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024. Archer: Training language model agents via hierarchical multi-turn rl. arXiv preprint arXiv:2402.19446.

Contents

1 Introduction **1**

2 Approach **3**
2.1 Preliminaries: LLM-based Agent for Task-Solving . . . 3
2.2 Agent Architecture . . . 3
2.3 Imitation Learning . . . 4
2.4 Joint Policy Gradient Optimization . . . 4

3 Experiments **5**
3.1 Environmental Settings . . . 5
3.2 Main Experiment on Multi-Agent (Q1) . . . 6
3.3 Comparison of Multi-Agent and Single-Agent (Q2) . . . 7
3.4 Effectiveness of Optimized Planner and Reflector (Q3) . . . 7
3.5 Effectiveness of Reinforcement Learning and Regularization (Q4&Q5) . . . 8

4 Conclusion **9**

5 Ethical Considerations and Limitations **9**

A Algorithm **14**

B Related Works **14**

C More Detailed Experimental Settings **15**
C.1 Evaluation Detials . . . 15
C.2 Baselines . . . 15
C.3 Expert Dataset . . . 15
C.4 Prompt Details . . . 16
C.5 Evaluation Metrics and Reward Function . . . 17
C.6 Training Details . . . 18

D More Experiments **20**
D.1 Comparison of Standard PPO and our proposed Off-policy policy gradient optimization . 20
D.2 Main Experiments on Multi-Agent (Complete Results and Additional Metrics) . . . 20
D.3 Error Analysis in ALFWorld . . . 20
D.4 Case Study . . . 21

A Algorithm

Algorithm 1 Practical Framework

```
Initialize parameters $\theta, \phi, \lambda, \alpha$
Initialize replay buffer $D^{\text{IL}}_{\text{planner}}, D^{\text{IL}}_{\text{reflector}}, D^{\text{RL}}_{\text{planner}}, D^{\text{RL}}_{\text{reflector}}$
## IL Data Collection
Initialize Expert Language Agent $\pi_{\text{expert}}, \mu_{\text{expert}}$
for each environment in trainset do
    for each trial do
        Update the initial state
        Generate $\tau^k = \{s^k_0, a^k_0, o^k_0, \ldots, s^k_n, a^k_n, o^k_n\}$ using $\pi_{\text{expert}}$
        Generate verbal reward $f^k \sim \mu_{\text{expert}}(\cdot|\tau^k)$
    end for
    Filter expert trajectories using evaluators and add to $D^{\text{IL}}_{\text{planner}}, D^{\text{IL}}_{\text{reflector}}$
end for
## IL training
Supervised Fine-tuning of $\pi_\theta$ and $\mu_\phi$ using $D^{\text{IL}}_{\text{planner}}$ and $D^{\text{IL}}_{\text{reflector}}$, respectively (Eq. 3 and 4)
for each iteration do
    ## RL exploration
    for each environment in trainset do
        for each trial do
            Update the initial state
            Generate $\tau^k = \{s^k_0, a^k_0, o^k_0, \ldots, s^k_n, a^k_n, o^k_n\}$ using $\pi_\theta$, add to replay buffer $D^{\text{RL}}_{\text{planner}}$
            Generate verbal reward $f^k \sim \mu_\phi(\cdot|\tau^k)$, add to replay buffer $D^{\text{RL}}_{\text{reflector}}$
        end for
    end for
    ## RL training
    Update $\theta$ and $\phi$ using off-policy policy gradient optimization (Eq. 9 and 10)
end for
```

The algorithm for RetroAct method is shown in Algorithm 1.

B Related Works

Retrospective language agent Benefitting from the LLMs' in-context learning capabilities, they can summarize environmental feedback into natural language-based reflections and use these reflections to improve performance in subsequent trials. Self-Refine (Madaan et al., 2024) processes results from environmental interactions, using these outcomes to improve performance. Reflexion (Shinn et al., 2024) involves introspection about feedback from the environment, generating reflective experiences that enhance reasoning abilities. Retroformer (Yao et al., 2023b) introduces a framework by learning a retrospective model to enhance LLM-based agents, which automatically adjusts language agent prompts based on environmental feedback through policy gradient. Nonetheless, these approaches typically rely on large-scale LLMs, resulting in significant costs and delays. Moreover, most smaller LLMs typically exhibit insufficient performance and robustness when deployed as agents.

Language Agent Fine-tuning To address the reliance of agent tasks on large-scale LLMs, the agent-tuning method is a standard solution. Researchers leverage powerful closed-source LLMs or human experts to generate expert trajectories, acting as a dataset for fine-tuning smaller open-source LLMs through imitation learning (Ho et al., 2022; Chen et al., 2023a; Zeng et al., 2023; Gou et al., 2023; Yang et al., 2024). Moreover, AutoAct (Qiao et al., 2024) enhances the planning, reflection and action capabilities of multi-agent systems through self-synthetic trajectories. However, its reflection is only a summary of the current situation and does not have the ability to continue learning and evolution. Compared to imitation learning, reinforcement learning aims to learn through self-exploration and trial-and-error without relying on the guidance of external experts. Archer (Xi et al., 2024) employs a hierarchical RL approach with two parallel RL algorithms to enhance the planning ability of LLM-based agents. Several other works (Lightman et al., 2023; Chen et al., 2024; Havrilla et al., 2024a) propose intricate reward models with more fine-grained reward signals to improve performance. In summary, these methods focus on training planner models through fine-tuning, but the fine-tuned agents usually struggle to retain their capability for learning and adaptation to the environment.

In summary, while these methods enhance planning models through fine-tuning, this process often diminishes the agent's ability for self-reflection.

C More Detailed Experimental Settings

C.1 Evaluation Detials

For HotpotQA and InterCode, we select 100 tasks as the test environments. For ALFWorld, we use 134 "out-of-distribution" tasks as the test environments. In all environments, we conduct ten rounds of trials and reflections. In terms of evaluation metrics, we report the *F1 Score* for HotpotQA, the *Success Rate* for ALFWorld, and the *Reward Score (IoU)* for InterCode in the main text. Additionally, we provide the *Exact Match Score* for HotpotQA and the *Success Rate* data for InterCode as supplementary experiments in the Appendix D. In tables, we report the initial reward (**IR**) to measure the planner's performance on the first trial, as well as the final reward (**FR**) and average reward (**AR**) to evaluate the performance of both the planner and reflector comprehensively.

C.2 Baselines

ReAct (Yao et al., 2023a) is a method that combines reasoning and action in language models, enhancing performance in understanding and decision-making tasks by alternately generating reasoning trajectories and task-specific actions.

Reflexion (Shinn et al., 2024) builds on the ReAct framework, allowing language agents to learn from past errors by converting feedback into textual summaries, providing context for future tasks to improve performance through self-reflection.

SFT (Chen et al., 2023a) is a fine-tuning method for language agents that enhances performance by utilizing expert trajectories, significantly improving efficiency and accuracy compared to prompt-based models. **We use the FireAct method to fine-tune the language agent separately on each dataset**.

EI (Havrilla et al., 2024b)(Expert Iteration) is a strong baseline that involves using a model initialized with SFT to generate data, filtering successful samples, and then using them to further fine-tune the model

RL We design an *off-policy* policy gradient algorithm to fine-tune the planner model as a baseline. This algorithm follows the same reinforcement learning (RL) framework as used in RetroAct, incorporating *off-policy* RL with imitation learning regularization. A detailed comparison between this algorithm and the standard PPO algorithm is provided in Appendix D.1, further justifying our choice of this baseline.

C.3 Expert Dataset

Environment and Dataset	Positive Examples (Planner)	Positive Examples (Reflector)
HotpotQA	6956	1304
ALFWorld	693	221
InterCode	866	97

Table 4: Imitation Learning Dataset

In our research, we collect an imitation learning dataset using expert models. The dataset encompasses three different environments. We employ our custom evaluator to retain the positive examples from this dataset. Detailed information is provided in Table 4.

C.4 Prompt Details

Across three datasets, Our prompt design on **HotpotQA** and **ALFWorld** follow (Yao et al., 2023a; Shinn et al., 2024), given that the aforementioned approach has not been previously tested on **InterCode**, we proceed by designing our prompts in accordance with the underlying design philosophy of the method described above.

Prompt C.1: Intercode Few-shot Planner Generation

#Problem description

Interact with a MySQL Database system using SQL queries to answer a question.

Use interleaving Thought, Action, Observation steps.

Thought can reason about the current situation, and Action can be two types:

(1) execute, which executes SQL code on the MySQL Database system

(2) submit, which indicates that the previous observation is the answer

#Few-shot description

Here are some examples.

Question: What are the names and grades for each high schooler?

Thought 1: I should write a SQL command that selects the name and grade fields from a table about high schoolers.

Action 1: execute [SELECT_ name, grade FROM high_schoolers]

Observation 1: Error executing query: Table “network_1.high_schoolers” doesn’t exist

......

Thought x: ...This should be the answer.

Action x: submit

#Target task information

Here is the task:

Question: What are the names of poker players, ordered ascending by the number of final tables they have made?

Thought 1:

Prompt C.2: Intercode Few-shot Reflector Generation

#Problem description

You will be given the history of a past experience in which you were placed in an environment and given a programming task to complete. You were unsuccessful in completing the task. Do not summarize your environment, but rather think about the strategy and path you took to attempt to complete the task. Devise a concise, new plan of action that accounts for your mistake with reference to specific actions that you should have taken. For example, if you tried A and B but forgot C, then devise a plan to achieve C with environment-specific actions. You will need this later when you are solving the same task. Give your plan after “Plan”.

#Few-shot description

Here are some examples:

[# the history of the failure trajectory

Question:

Thought x:

Action x:

Observation x:

......

]

STATUS: FAIL

New plan: I was unsuccessful in finding the most common bond type because I did not consider the possibility of null or empty values in the "bond_type" column. In the future, I should modify my SQL query to exclude these values. The revised plan would be to first check the tables in the database, then check the columns in the "bond" table, and finally execute a SQL query to find the most common bond type, excluding null or empty values. The new SQL query would be: ...

#Target task information

Here is the task:

[# A trajectory that requires reflection

Question:

Thought x:

Action x:

Observation x:

......

STATUS: FAIL

]

Plans from past attempts:

### reflection of the past failure trajectory in this task ###

New plan:

C.5 Evaluation Metrics and Reward Function

We summarize the reward functions and evaluation metrics for the three datasets in a table 5. Additionally, we provide the final evaluation results for all these metrics.

HotpotQA We employ two primary metrics to assess the performance of models: **Exact Match (EM)** and the **F1 Score**. The EM score is a strict metric that measures whether the normalized predicted answer

Environment and Dataset	Evaluator	Planner Reward Function
HotpotQA	EM Score (Eq. 11)	F1 Score (Eq. 12)
ALFWorld	Success Rate (Eq. 13)	
InterCode	Success Rate (Eq. 14)	Reward Score (Eq. 15)

Table 5: Evalator and Reward Function

exactly matches the normalized gold answer. It is defined as:

$$EM = \begin{cases} 1 & \text{if normalized prediction = normalized gold} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

The F1 Score is the harmonic mean of precision and recall. Precision (precision), recall (recall), and the F1 score ($F1$) are calculated as follows:

$$\text{precision} = \frac{\text{num\_same}}{\text{prediction}}, \quad \text{recall} = \frac{\text{num\_same}}{\text{gold}}, \quad F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{12}$$

These metrics effectively evaluate the accuracy and reliability of answers generated by models in the HotpotQA dataset. When building the expert data set for imitation learning, we use the EM metric to filter the expert data. When performing reinforcement learning, we use the F1 score as the planner's reward.

ALFWorld Since ALFWorld only provides information on failure and success, we use whether the agent completes the task in the environment as the evaluation criterion and reward function.

$$SR = \begin{cases} 1 & \text{if agent completes the task} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

InterCode We employ two primary metrics to assess the performance of models: **Success Rate (SR)** and the **Reward Score**. The SR score is a strict metric that measures whether the SQL operations completed according to the task requirements. It is defined as:

$$SR = \begin{cases} 1 & \text{if the result of the program execution matches the gold answers} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

The execution outcome of all SQL queries is a list of records. In order to more accurately and meticulously evaluate the results of the agent's command execution, we employ the same method as (Yang et al., 2023), utilizing Intersection over Union (IoU), or more formally the Jaccard Index, to quantify the accuracy of the latest output generated by the agent in comparison to the gold standard output. Given the agent's latest execution output A and the gold answer's execution output G, the reward function is as follows:

$$\mathcal{R} = \frac{A \cap G}{A \cup G} \times \left(\frac{\text{kendalltau}(A \cap G, G \cap A) + 1}{2} \right) \tag{15}$$

C.6 Training Details

Model Details We use Llama-chat-7b and Llama-chat-13b models in our experiments respectively. During training, we do not introduce additional prompt templates. Moreover, although calculating loss only in the model-generated parts of the trajectory is the optimal choice, splitting the trajectory significantly increases training costs. To balance training costs, we directly use the entire trajectory for auto-regressive training. We find that this approach does not result in significant performance loss while reducing training costs.

Inference Details During testing, we fix the *temperature* parameter of all models to 0.0. This eliminates any randomness in the local models, ensuring the reproducibility of experiments and confirming that improvements in reflection are not due to randomness. In reinforcement learning training, we set the *temperature* to 1.0 to allow for exploration. We use the *vllm* (Kwon et al., 2023) framework to accelerate all inference processes. For HotpotQA, we limit the number of steps for a single trial to 5; for ALFWorld, we set the limit to 50; and for InterCode, we set the limit to 10.

Training Details We implement LoRA based on PEFT (Hu et al., 2021) and set $r_{LoRA} = 8$ and $\alpha_{LoRA} = 16$ for training in all experiments. We implement our own off-policy reinforcement learning algorithm based on the transformers (Wolf et al., 2020) and open-source the code at `https://anonymous.4open.science/r/RetroAct-04E8.`

System Specifications The system specifications for our experiments is shown in Table 6.

Name	**Details**
CPU	Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz
GPU	4 * Nvidia A800 80GB PCIE
Memory	1TB RAM
Python	Version 3.9
Transformers	Version 4.38.2

Table 6: System Specifications

Hyperparameters We set the regularization coefficient in RL, $\lambda_{planner} = 1.0$, $\lambda_{reflector} = 1.0$ and the reward coefficient for the reflector, $\alpha = 1.0$. Additionally, we conduct hyperparameter searches for the baseline IL and RL methods, as well as our RetroAct method, within the range of learning rates $\{5e-05, 1e-04, 3e-04\}$ and epochs $\{3, 5\}$. In our experiments, due to the small data volume in the ALFWorld and InterCode datasets, we find that using a large batch size leads to insufficient update steps, resulting in severe underfitting. To maintain uniform settings, we use a batch size of 1 across all datasets. However, we strongly recommend increasing the batch size in scenarios with sufficient data volume to improve training stability.

D More Experiments

D.1 Comparison of Standard PPO and our proposed Off-policy policy gradient optimization

The standard PPO algorithm for LLM was originally designed for standard Reinforcement Learning with Human Feedback (RLHF) tasks, particularly suitable for preference optimization scenarios. In such cases, since it's not feasible to obtain real-time human feedback during training, it relies on pre-labeled preference datasets to train a reward model, which then guides the reinforcement learning process. However, in LLM-Agent tasks, the agent interacts with the environment multiple times and can directly obtain rewards from it. Additionally, training a reward model in multi-step tasks is particularly challenging. We conduct an initial evaluation of the standard PPO algorithm's effectiveness in agent tuning tasks using the HotpotQA dataset:

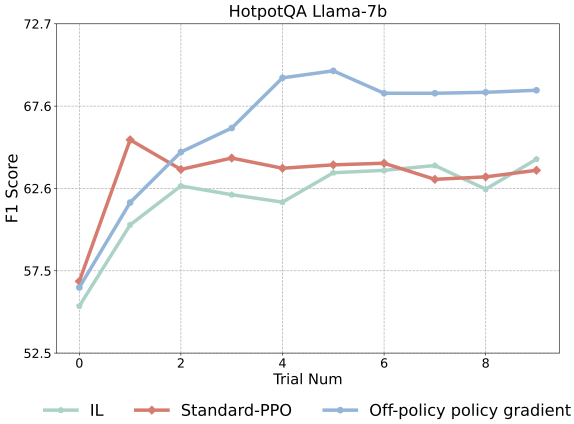

Figure 7: Experiments on Multi-Agent (Complete Results)

In the figure 7, the performance of the standard PPO algorithm and imitation learning is similar, both lower than the Off-policy policy gradient algorithm we designed. In fact, we find that the standard PPO can easily cause the agent to lose the ability to follow instructions and fail to complete the task correctly. We analyze the reasons as follows: Standard PPO for RLHF (von Werra et al., 2020) operates on a token-level reward mechanism, where the environment reward (provided by the reward model) only applies to the last token of a sequence, while the intermediate tokens rely on rewards provided by the continually learning critic model. Therefore, to achieve the desired results with PPO, typically, extensive training time and a large amount of data are required, along with thorough training of the reward model, critic model, and policy model to ensure proper convergence of the final objective. However, standard agent tasks often struggle to provide the high-quality data needed for this process.

The standard token-level PPO algorithms, while theoretically capable of providing more fine-grained supervision signals, face significant challenges in achieving good convergence in agent tuning tasks. As demonstrated in (Song et al., 2024), PPO, when used as a baseline, exhibit poor performance, making it a less meaningful and costly choice as a general RL baseline. Therefore, to make the experimental comparison more valuable, we use our designed off-policy policy gradient algorithm with imitation learning regularization as a more challenging RL baseline in our paper.

D.2 Main Experiments on Multi-Agent (Complete Results and Additional Metrics)

In this section, we present the complete data of our method and baseline methods in Figure 8. It can be observed that, whether in terms of rewards or additional metrics, the overall trends are consistent with the conclusions drawn in the main text. Our method outperforms the baseline methods significantly and is comparable to methods based on closed-source models.

D.3 Error Analysis in ALFWorld

RetroAct significantly outperforms the baseline agent by completing 130 out of 134 tasks based on Llama-7b. In this section, we present a classification of ALFWorld trajectories by reason of failure in

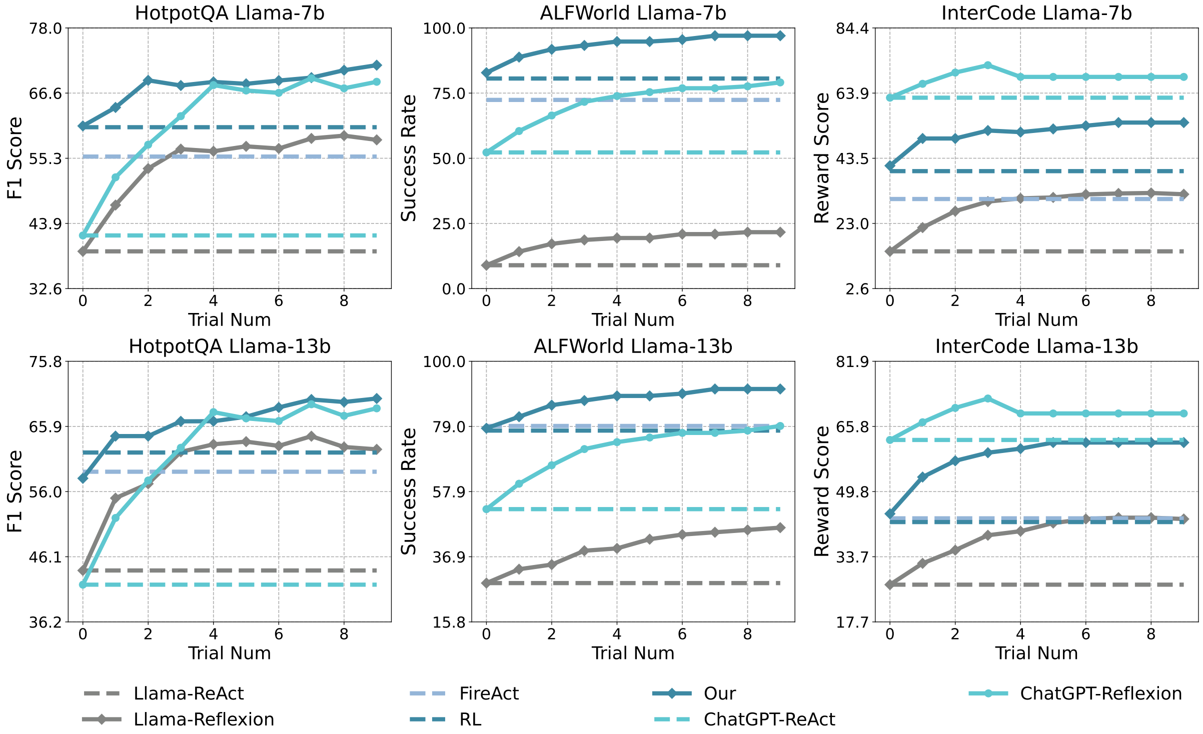

Figure 8: Experiments on Multi-Agent (Complete Results)

Figure 9, following (Shinn et al., 2024). The reasons can be summarized into two points: **Hallucination**: The agent attempts to pick up non-existent items at a location and holds onto imaginary objects; **Inefficient planning**: The agent formulates plans that lack common sense and fails to make accurate judgments based on environmental feedback.

By analyzing the original trajectories, we find that fine-tuning and integrating prior knowledge effectively improves the rationality of the agent's actions, allowing it to track the placement of objects better. This underscores the crucial role of fine-tuning in enhancing the agent's capabilities. Moreover, by summarizing experiences, adjusting plans, and attempting multiple iterations, RetroAct completes most of the previously failed tasks. This process highlights the essential role the self-reflection plays in improving RetroAct's planning.

D.4 Case Study

In this section, we conduct a detailed and in-depth case study of our model. We deliberately select complex tasks from the original data that require multiple reflections to succeed. These cases comprehensively verify that our method can simultaneously enhance the model's planning and reflection capabilities.

HotpotQA In HotpotQA, we deliberately select a challenging task requiring multi-step complex reasoning. The agent is tasked with answering the question: "When Copsi was made Earl of Northumbria, he went back to reside in a town at the confluence of which two rivers?" To answer this task, the agent needs to follow the correct reasoning process: **(1)** first, use a tool to search for Copsi and obtain information about his life; **(2)** then, correctly extract the town where Copsi resided as Earl of Northumbria from the returned information, avoiding other confusing details; **(3)** clearly understand that the question asks for the rivers at the town's location, not just the city itself; **(4)** correctly use a tool to search for the rivers at the town's location and answer the question accurately. We provide a specific case, comparing our agent (D.4) with the baseline agent (D.4).

In RetroAct agent's first attempt, it completes steps **1** and **2** but directly answers with the town. After two reflections, our agent correctly understands step **3** and clarifies the question details. Finally, in a post-reflection attempt, it completes the step **4**. In contrast, the baseline agent correctly completes step **1** but repeatedly fails to extract the correct information in step **2**. Even after nine subsequent reflections and

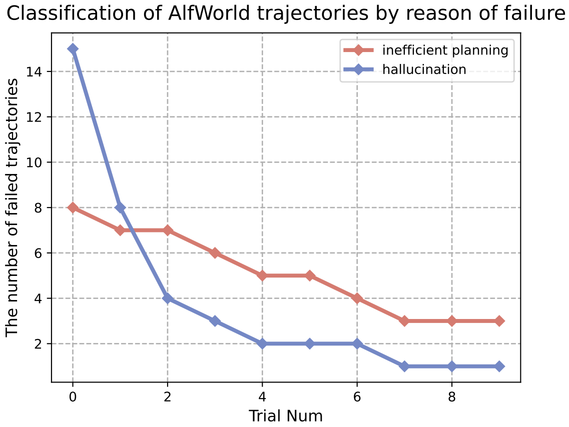

Figure 9: Error Analysis in ALFWorld

attempts, it remains stuck at step **2**, falling into a loop.

This case demonstrates that our planner is superior to the baseline, as it can complete step **2** on the first attempt. Our reflector is also more effective, identifying the critical error during reflection and completing step **3**, and then helping the planner complete the **4**.

AlfWorld In Alfworld, we select a challenging task that necessitates a clear and adaptive planning approach in response to the environment. The agent is in the middle of a room, where a multitude of items is arranged scattered across different spots, to look at the bowl under the desk lamp. To complete this task, the agent needs to follow the correct planning and reasoning process: **(1)** The agent needs to devise a strategy first to take the bowl, then use the desklamp to examine the bowl, as opposed to finding the desklamp and mistakenly taking it away then carrying the desklamp to find the bowl. **(2)** The agent is tasked with locating the bowl in accordance with a pre-established plan. **(3)** The agent, with the bowl in possession, proceeds to find the desklamp, intending to utilize it to look at the bowl. We provide a specific case, comparing our agent (D.4) with the baseline agent (D.4).

In the beginning, the RetroAct agent successfully formulates a plan that involves locating the bowl first and then proceeds to find the desklamp with the bowl in possession in step **1**. However, some hallucination issues still arise in steps **2** and **3**, such as attempting to take non-existent items or failing to act upon seeing the targeted objects. After five reflections, the agent clearly recognized the sequence: first, to locate the bowl, and then, to find the desklamp. It further leverages past trajectory experiences to discern the positions of various objects within the environment. Upon discovering the desklamp, it adopts a strategic approach by noting its location and only afterward proceeds to search for the bowl, intending to utilize the desklamp effectively once the bowl is in hand. In contrast, the baseline agent is struggling with devising an accurate path that initially finds the bowl and then takes it to find a desklamp, despite multiple rounds of reflection, fails to identify and rectify its mistakes in step **1**. It continues to experience hallucinations more severe than RetroAct, leading to persistent inefficiencies in planning and execution even after repeated introspection cycles in step **2** and **3**.

This case demonstrates that our planner is superior to the baseline because it can complete step **1** on the first attempt. It demonstrates an appropriate utilization of learned strategies and prior knowledge integrated to avoid mistakenly taking the desklamp away. Meanwhile, our planner is designed to minimize hallucinations in step **2** to the greatest extent possible, ensuring it executes actions in line with its predefined plan. Given the extensive space to explore within the room, mistakes are inevitable, even with a clear plan in place. Consequently, the agent must also possess the ability to accurately reflect upon and correct errors arising from the current environmental conditions. Our reflector is also more effective, identifying the critical error during reflection and helping the planner complete the target in step **3**. This highlights a need for substantial improvement in its reflector and the robustness of its planner to overcome these challenges.

RetroAct Case D.1: Better Planning and Reflection in HotpotQA

Trial 1:

Question:
When Copsi was made earl of Northumbria, he went back to reside in a town at the confluence of which two rivers?

Thought 1: I need to search Copsi and Northumbria, find the town where Copsi resided as earl of Northumbria.

Action 1: Search[Copsi]

Observation 1: Copsi (or Copsig; Old English: C̈0̆14dpsige)̈ [died 1067] was a Northumbrian magnate in late Anglo-Saxon England. He was a supporter of Tostig and was exiled along with him in 1065. Copsi soon fled to Orkney (then a part of Norway). The next year (1066), he joined Tostig at Sandwich in Kent with 17 ships. Copsi survived Tostig's defeat at Stamford Bridge. When William the Conqueror prevailed at Hastings, he traveled, in March 1067, to pay William homage at Barking (where William was staying while his tower was being constructed in London). In return, William made Copsi earl of Northumbria and sent him back to York. Copsi's rule lasted a mere five weeks. At that time, he was murdered by Osulf, son of Eadulf III of the ancient Bernician family, which had historically governed the area from Bamburghproves at Newburn-upon-Tyne. Osulf, however, only ruled as earl until that autumn, when an outlaw killed him he was tracking.

#Agent successfully extracts the correct town information "York" from a large amount of text.

Thought 2: Copsi was made earl of Northumbria and resided in York.

#Agent correctly extracts the town name on the first trial, but the question asks, "Which two rivers converge in that town?".

Action 2: Finish[York]

Observation 2:Answer is INCORRECT.

...

Trial 3:

#Agent successfully reflects on the previous error, identifying the cause as not correctly responding to the question. In its reflection, it explicitly states, "Then find the rivers it is located on".

Reflections:
I searched Copsi and Northumbria, then found the town where Copsi was made earl of Northumbria. I should have searched the town where Copsi was made earl of Northumbria and found the rivers where it was located.

Question:
When Copsi was made earl of Northumbria, he went back to reside in a town at the confluence of which two rivers?

Thought 1: I need to search Copsi and Northumbria, find the town where Copsi was made earl of

Northumbria, and then find the rivers where it was located.

Action 1: Search[Copsi and Northumbria]

Observation 1: Copsi (or Copsig; Old English: C̈0̆14dpsige)̈ [died 1067] was a Northumbrian magnate in late Anglo-Saxon England. He was a supporter of Tostig, and was exiled along with him in 1065. Copsi soon fled to Orkney (then a part of Norway). The next year (1066), he joined Tostig at Sandwich, in Kent, with 17 ships. Copsi survived Tostig's defeat at Stamford Bridge, and when William the Conqueror prevailed at Hastings he travelled, in March 1067, to pay William homage at Barking (where William was staying while his tower was being constructed in London). In return, William made Copsi earl of Northumbria and sent him back to York. Copsi's rule lasted a mere five weeks, at which time he was murdered by Osulf, son of Eadulf III of the ancient Bernician family which had historically governed the area from Bamburgh, at Newburn-upon-Tyne. Osulf, however, only ruled as earl until that autumn, when he was killed by an outlaw he was tracking.

Thought 2: Copsi was made earl of Northumbria at York. I need to find the rivers where York is located.

#Agent correctly responds to its own reflection by searching for the relevant rivers after identifying the town information, and ultimately answers the question correctly.
Action 2: Search[York rivers]

Observation 2: York ( ) is a historic walled city at the confluence of the rivers Ouse and Foss in North Yorkshire, England. The municipality is the traditional county town of the historic county of Yorkshire to which it gives its name. The city has a rich heritage and has provided the backdrop to major political events in England throughout much of its two millennia of existence. The city offers a wealth of historic attractions, of which York Minster is the most prominent, and a variety of cultural and sporting activities making it a popular tourist destination for millions.

Thought 3: York is located at the confluence of the rivers Ouse and Foss.

Action 3: Finish[Ouse and Foss]

Observation 3: Answer is CORRECT.

Baseline Case D.1: Poor Planning and Reflection in HotpotQA

Trial 1:

Question:
When Copsi was made earl of Northumbria he went back to reside in a town at the confluence of which two rivers?

Thought 1: I need to search Copsi and the two rivers, find the town.

Action 1: Search[Copsi]

Observation 1: Copsi (or Copsig; Old English: C̈0̆14dpsige)̈ [died 1067] was a Northumbrian magnate in late Anglo-Saxon England. He was a supporter of Tostig, and was exiled along with him in 1065. Copsi soon fled to Orkney (then a part of Norway). The next year (1066), he joined Tostig at Sandwich, in Kent, with 17 ships. Copsi survived Tostig's defeat at Stamford Bridge, and when William the Conqueror prevailed at Hastings he travelled, in March 1067, to pay William homage at Barking (where William was staying while his tower was being constructed in London). In return, William made Copsi earl of Northumbria and sent him back to York. Copsi's rule lasted a mere five weeks, at which time he was murdered by Osulf, son of Eadulf III of the ancient Bernician family which had historically governed the area from Bamburgh, at Newburn-upon-Tyne. Osulf, however, only ruled as earl until that autumn, when he was killed by an outlaw he was tracking.

#Despite retrieving the correct information, the agent incorrectly extracts the town information from the search results.
Thought 2: The town at the confluence of the two rivers is Newburn-upon-Tyne.

Action 2: Search[Newburn-upon-Tyne]

Observation 2: Two Rivers High School is a public high school for students in grades 7 through 12 located in Ola, Arkansas, United States. Two Rivers High School is administered by the Two Rivers School District. The Two Rivers High School and school district serves 650 sqmi of rural communities including Ola, Plainview, Rover, and Casa. Two Rivers High School is one of four high schools within Yell County and the sole high school administered by the Two Rivers School District.

Thought 3: Two Rivers High School is not the town at the confluence of the two rivers.

Action 3: Finish[Newburn-upon-Tyne]

Observation 3: Answer is INCORRECT

Trial 2:

#The agent never reflects on the useful information, leading it to remain stuck in a continuous loop.
Reflections:
I searched Copsi and the two rivers, but I did not find the town. I should have searched for the name of the town in the observation.

Thought 1: I need to search for the town where Copsi resided.

Action 1: Search[town at the confluence of two rivers]

Observation 1: The Two Rivers is a 7.0 mi river in Kittson County, northwestern Minnesota, in the United States. Formed by the North Branch of the Two Rivers and the South Branch of the Two Rivers, it is a tributary of the Red River of the North, with its outflow traveling north through Lake Winnipeg and the Nelson River to Hudson Bay.

Thought 2: The town where Copsi resided is not the Two Rivers. I need to search for the correct town.

Action 2: Search[Copsi]

Observation 2: Copsi (or Copsig; Old English: C̈0̆14dpsige)̈ [died 1067] was a Northumbrian magnate in late Anglo-Saxon England. He was a supporter of Tostig, and was exiled along with him in 1065. Copsi soon fled to Orkney (then a part of Norway). The next year (1066), he joined Tostig at Sandwich, in Kent, with 17 ships. Copsi survived Tostig's defeat at Stamford Bridge, and when William the Conqueror prevailed at Hastings he travelled, in March 1067, to pay William homage at Barking (where William was staying while his tower was being constructed in London). In return, William made Copsi earl of Northumbria and sent him back to York. Copsi's rule lasted a mere five weeks. At that time, he was murdered by Osulf, son of Eadulf III of the ancient Bernician family which had historically governed the area from Bamburgh, at Newburn-upon-Tyne. Osulf, however, only ruled as earl until that autumn, when an outlaw killed him he was tracking.

Thought 3: The town where Copsi resided is Newburn-upon-Tyne. I need to search for the correct answer.

Action 3: Search[Newburn-upon-Tyne]

Observation 3: Two Rivers High School is a public high school for students in grades 7 through 12 located in Ola, Arkansas, United States. The Two Rivers School District administers two Rivers High School. The Two Rivers High School and school district serves 650 sqmi of rural communities, including Ola, Plainview, Rover, and Casa. Two Rivers High School is one of four high schools within Yell County and the sole high school administered by the Two Rivers School District.

Thought 4: Two Rivers High School is not the town where Copsi resided. I need to search for the correct answer.

Action 4: Search[Copsi]

Observation 4: Copsi (or Copsig; Old English: C̈0̆14dpsige)̈ [died 1067] was a Northumbrian magnate in late Anglo-Saxon England. He was a supporter of Tostig, and was exiled along with him in 1065. Copsi soon fled to Orkney (then a part of Norway). The next year (1066), he joined Tostig at Sandwich, in Kent, with 17 ships. Copsi survived Tostig's defeat at Stamford Bridge, and when William the Conqueror prevailed at Hastings he travelled, in March 1067, to pay William homage at Barking (where William was staying while his tower was being constructed in London). In return, William made Copsi earl of Northumbria and sent him back to York. Copsi's rule lasted a mere five weeks, at which time he was murdered by Osulf, son of Eadulf III of the ancient Bernician family which had historically governed the area from Bamburgh, at Newburn-upon-Tyne. Osulf, however, only ruled as earl until that autumn, when he was killed by an outlaw he was tracking.

Thought 5: The town where Copsi resided is Newburn-upon-Tyne. I need to search for the correct answer.

Action 5: Search[Newburn-upon-Tyne]

Observation 5: Two Rivers High School is a public high school for students in grades 7 through 12 located in Ola, Arkansas, United States. Two Rivers High School is administered by the Two Rivers School District. The Two Rivers High School and school district serves 650 sqmi of rural communities including Ola, Plainview, Rover, and Casa. Two Rivers High School is one of four high schools within Yell County and the sole high school administered by the Two Rivers School District.

Trial 3-10:

Agent falls into a Trial2-like dead loop until the maximum number of attempts was reached.

RetroAct Case D.2: Better Planning and Reflection in AlfWorld

Here is the task: You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.
Your task is to: look at bowl under the desklamp.

Trial 1:
> think: To solve the task, I need to **find and take a bowl**, **then find and use a desklamp**. First I need to find a bowl. A bowl is more likely to appear in drawer (1-6), shelf (1-6), bed (1), garbagecan (1), laundryhamper (1), safe (1). I can check one by one, starting with drawer 1.
> go to drawer 1
The drawer 1 is closed.
...
> open drawer 6
You open the drawer 6. The drawer 6 is open. In it, you see a book 2, and a pen 2.
> go to shelf 1
On the shelf 1, you see a pencil 2.
...
> go to shelf 6
On the shelf 6, you see nothing.
> think: Now I find a bowl (1). Next, I need to take it.
> take bowl 1 from shelf 6
Nothing happens.
> think: I am stuck in a loop. I will try to execute a different action.
# The agent experiences a hallucination, prompting it to make the motion of picking up a bowl where none actually exists.

> go to bed 1
On the bed 1, you see a book 1, a laptop 1, a pillow 2, and a pillow 1.
> go to desk 2
On the desk 2, you see a bowl 2, a bowl 1, a cd 2, a cd 1, and a creditcard 3.
> think: Now I find a bowl (2). Next, I need to take it.
> take bowl 2 from desk 2
You pick up the bowl 2 from the desk 2.
> think: Now I take a bowl (2). Next, I need to **find a desklamp**. A desklamp is more likely to **appear in desk (1-2)**, bed (1), shelf (1-6), drawer (1-6), garbagecan (1), laundryhamper (1), safe (1). I can check one by one, starting with desk 1.
> go to desk 1
On the desk 1, you see a alarmclock 1, a desklamp 1, a mug 1, and a pencil 1.
> go to desk 2
On the desk 2, you see a bowl 1, a cd 2, a cd 1, and a creditcard 3.
# Despite the presence of a target object, a desklamp, the agent fails to devise a plan incorporating the use of the desklamp to accomplish its task.

```
> use desklamp 1
Nothing happens.
> think: I am stuck in a loop. I will try to execute a different action.
> go to garbagecan 1
On the garbagecan 1, you see nothing.
...
> go to shelf 3
On the shelf 3, you see a desklamp 1, and a pencil 1.
> go to shelf 4
On the shelf 4, you see a alarmclock 2, and a pencil 3.
...
> go to shelf 6
On the shelf 6, you see nothing.
> think: Now I find a bowl (1). Next, I need to take it.
> take bowl 1 from shelf 6
Nothing happens.
> think: I am stuck in a loop. I will try to execute a different action.
> go to bed 1
...
# The agent appears to have disregarded its prior planning, encountering a desklamp yet neglecting to utilize it, proceeding instead to attempt fetching a bowl again, ultimately leading to task failure.
```

STATUS: FAIL

Trial 6:

Reflections:

I was stuck in a loop in which I continually tried to take a bowl from shelf 6. I should have found a bowl first, then taken it. I will try to execute a different action if I am stuck in a loop again.
I was stuck in a loop in which I continually tried to use a desklamp. I should have found a bowl first, then taken it, then used a desklamp to light the bowl. I will try to execute a different action if I am stuck in a loop again.
...

```
> think: To solve the task, I need to find and take a bowl, then find and use a desklamp. First I need to find a bowl. A bowl is more likely to appear in drawer (1-6), shelf (1-6), bed (1), garbagecan (1), laundryhamper (1), safe (1). I can check one by one, starting with desk 1.
> go to desk 1
On the desk 1, you see a alarmclock 1, a desklamp 1, a mug 1, and a pencil 1.
> think: Now I find a desklamp (1).I will try to execute a different action.
> go to desk 2
On the desk 2, you see a bowl 2, a bowl 1, a cd 2, a cd 1, and a creditcard 3.
> think: Now I find a bowl (2). Next, I need to take it.
> take bowl 2 from desk 2
You pick up the bowl 2 from the desk 2.
> think: Now I take a bowl (2). Next, I need to find a desklamp. A desklamp is more likely to appear in desk (1), shelf (1-6), bed (1), garbagecan (1), laundryhamper (1), safe (1). I can check one by one, starting with desk 1.
> go to desk 1
On the desk 1, you see a alarmclock 1, a desklamp 1, a mug 1, and a pencil 1.
> think: Now I find a desklamp (1). Next, I need to use it.
```

> use desklamp 1
You turn on the desklamp 1.

STATUS: OK

Baseline Case D.2: Poor Planning and Reflection in ALfowrld

Here is the task: You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.
Your task is to: look at bowl under the desklamp.

Trial 1:
> think: To solve the task, I need to find and use a desklamp. A desklamp is more likely to appear in desk (2), desk (1), drawer (6), drawer (5), drawer (4), drawer (3), drawer (2), drawer (1), garbagecan (1), laundryhamper (1), safe (1), shelf (6), shelf (5), shelf (4),
> go to desk 1
On the desk 1, you see a desklamp 1, a keychain 3, a mug 2, a pen 2, a pen 1, and a pencil 2.
> go to desk 2
On the desk 2, you see a alarmclock 1, a bowl 1, and a cd 3.
> go to drawer 6
The drawer 6 is closed.
...
> go to drawer 1
The drawer 1 is closed.
...
> go to shelf 6
On the shelf 6, you see nothing.
...
> go to shelf 3
On the shelf 3, you see a desklamp 1, a pen 1, and a pencil 2.
...
> go to bed 1
On the bed 1, you see a book 1, a cellphone 1, a laptop 2, a laptop 1, a pillow 2, and a pillow 1.
> go to desk 1
On the desk 1, you see a desklamp 1, a keychain 3, a mug 2, a pen 2, a pen 1, and a pencil 2.
> think: Now I find a desklamp (1). Next, I need to use it.
> use desklamp 1
You turn on the desklamp 1.
> look at bowl under desklamp
Nothing happens.
> go back
Nothing happens.

STATUS: FAIL

Trial 3:

#The agent navigates the environment under flawed planning strategies.
Reflections:

In this environment, my past attempts were unsuccessful because I tried to execute the task in the wrong order. I will try to execute the task in the correct order this time. I will go to desk 1, find the desklamp, then look for the bowl under the desklamp.

In this environment, my plan was to find a bowl then find and use a desklamp. However, the task says to look at bowl under the desklamp. I should have looked for the desklamp then looked for the bowl. I noticed that the desklamp was found on desk 1. In the next trial, I will go to desk 1, find the lamp, then look for the bowl under the desklamp.

...

```
> think: To solve the task, I need to find and use a desklamp. A desklamp is more likely to appear in desk (2), desk (1), drawer (6), drawer (5), drawer (4), drawer (3), drawer (2), drawer (1), garbagecan (1), laundryhamper (1), safe (1), shelf (6), shelf (5), shelf (4),
> go to desk 2
On the desk 2, you see a alarmclock 1, a bowl 1, and a cd 3.
> go to desk 1
On the desk 1, you see a desklamp 1, a keychain 3, a mug 2, a pen 2, a pen 1, and a pencil 2.
...
> go to drawer 1
The drawer 1 is closed.
> open drawer 1
You open the drawer 1. The drawer 1 is open. In it, you see a creditcard 1.
> go to safe 1
...
The safe 1 is closed.
> open safe 1
You open the safe 1. The safe 1 is open. In it, you see a keychain 2, and a keychain 1.
> go to shelf 3
On the shelf 3, you see a desklamp 1, a pen 1, and a pencil 2.
...
#The agent never reflects on the useful information, leading it to remain stuck in a continuous loop.
> go to shelf 1
On the shelf 1, you see a cellphone 3, and a creditcard 2.
> go to desk 1
On the desk 1, you see a desklamp 1, a keychain 3, a mug 2, a pen 2, a pen 1, and a pencil 2.
> think: Now I find a desklamp (1). Next, I need to use it.
> use desklamp 1
You turn on the desklamp 1.
> look at bowl under desklamp
```

STATUS: FAIL

Trial 3-10:

Agent falls into a Trial3-like dead loop until the maximum number of attempts was reached.