MIRROR: Multi-agent Intra- and Inter-Reflection for Optimized Reasoning in Tool Learning

Zikang Guo[1] [*], **Benfeng Xu**[1], **Xiaorui Wang**[2] and **Zhendong Mao**[1†]
[1]University of Science and Technology of China
[2]Metastone Technology, Beijing, China
{gzk170401, benfeng}@mail.ustc.edu.cn
harrywxr@outlook.com, zdmao@ustc.edu.cn

Abstract

Complex tasks involving tool integration pose significant challenges for Large Language Models (LLMs), leading to the emergence of multi-agent workflows as a promising solution. Reflection has emerged as an effective strategy for correcting erroneous trajectories in agentic workflows. However, existing approaches only exploit such capability in the post-action stage, where the agent observes the execution outcomes. We argue that, like humans, LLMs can also engage in reflection before action execution: the agent can *anticipate* undesirable outcomes from its own decisions, which not only provides a necessarily complementary perspective to evaluate the decision but also prevents the propagation of errors throughout the trajectory. In this paper, we propose **MIRROR**, a framework that consists of both **intra-reflection**, which critically assesses intended actions before execution, and **inter-reflection**, which further adjusts the trajectory based on observations. This design systematically leverages LLM reflection capabilities to eliminate and rectify erroneous actions on a more comprehensive scope. Evaluations on both the StableToolBench and TravelPlanner benchmarks demonstrate MIRROR's superior performance, achieving state-of-the-art results compared to existing approaches.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language processing, showcasing unprecedented performance in planning, reasoning, and complex task completion [Brown, 2020; Chen *et al.*, 2022; Sun *et al.*, 2023; Yang *et al.*, 2023; Touvron *et al.*, 2023]. However, these models face fundamental limitations in accessing real-time information, processing novel data patterns, and executing precise system controls [Petroni *et al.*, 2020; Lewis *et al.*, 2020; Komeili, 2021; Zhang *et al.*, 2022]. These constraints become particularly challenging when tackling real-world tasks that require dynamic adaptation and accurate tool manipulation.

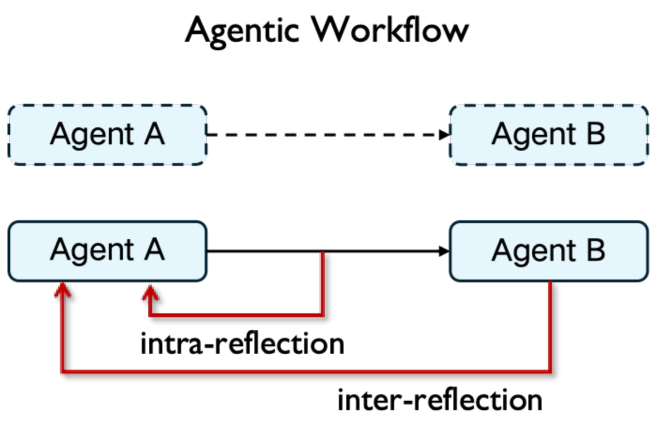


Figure 1: Architectural comparison of intra-reflection and inter-reflection in MIRROR. Intra-reflection operates as a preventive mechanism within each agent, evaluating decisions before execution to prevent errors. Inter-reflection functions as a corrective mechanism between agents, enabling system-wide optimization through post-execution learning. This dual-phase approach creates a comprehensive error prevention and correction system.

The tool learning paradigm emerged as a natural solution to extend LLMs' capabilities beyond their inherent limitations [Qin *et al.*, 2023]. By enabling LLMs to interact with external tools and APIs, this approach significantly expands their practical applications. Recent advances in foundation models have further enhanced their ability to handle basic tool interactions [Achiam *et al.*, 2023; Team *et al.*, 2023; Dubey *et al.*, 2024; Team, 2024]. However, when confronted with sophisticated multi-step tasks requiring intricate tool coordination, even these advanced models struggle to maintain consistent performance.

This challenge has given rise to multi-agent workflows, where complex tasks are distributed across specialized agents [Wu *et al.*, 2024; Hong *et al.*, 2024; Wang *et al.*, 2024b]. In these systems, reflection mechanisms play a crucial role in optimizing agent behavior. Current approaches, exemplified by Reflexion [Shinn *et al.*, 2024], implement reflection as a post-execution analysis tool, where agents learn from completed actions. While this retrospective approach enables

[*]Work done during the internship at Metastone.
[†]Corresponding author: Zhendong Mao.

iterative improvement, it suffers from fundamental limitations: it cannot prevent initial errors, risks irreversible system changes, and incurs significant learning costs through trial and error.

To address these limitations, we propose intra-reflection: a proactive evaluation mechanism that occurs before action execution. This approach mirrors human cognitive processes, where we often mentally simulate outcomes before taking action. By combining this preventive strategy with traditional post-execution reflection, we create a more comprehensive framework for multi-agent learning and adaptation.

Our paper introduces **MIRROR** (Multi-agent Intra- and Inter-Reflection for Optimized Reasoning), a novel framework that seamlessly integrates both reflection phases. This dual-phase approach systematically prevents errors while optimizing learning from necessary failures, creating a more robust and efficient multi-agent system. Unlike existing solutions that rely solely on learning from mistakes, MIRROR enables agents to anticipate and avoid potential errors while maintaining the ability to learn from unavoidable failures.

The primary contributions of this paper are summarized as follows:

- We introduce the concept of intra-reflection in multi-agent tool learning, a pre-action evaluation mechanism that allows agents to critically assess their intended outputs before execution.
- We propose MIRROR, a novel framework for multi-agent systems in tool learning tasks. MIRROR employs both intra-reflection and inter-reflection mechanisms to enhance decision-making, prevent error propagation, and improve collaboration efficiency.
- We provide a comprehensive evaluation of our framework using both StableToolBench and TravelPlanner benchmarks, demonstrating its superior performance across evaluation metrics, significantly outperforming existing state-of-the-art approaches.

2 Related work

LLM-based Task Planning. LLMs have been increasingly leveraged for task planning. Initial approaches like Chain-of-Thought (CoT) [Wei *et al.*, 2022] generate intermediate reasoning steps, while Self-Consistency [Wang *et al.*, 2022] improves robustness by generating multiple reasoning paths and selecting the most consistent solution. ReAct [Yao *et al.*, 2022] integrates reasoning with action-taking capabilities. More advanced methods explore complex reasoning structures: Tree-of-Thought (ToT) [Yao *et al.*, 2024] enables simultaneous exploration and evaluation of multiple reasoning paths, and Graph of Thoughts (GoT) [Besta *et al.*, 2024] provides even greater flexibility with graph-structured, multi-directional thought connections. Our method builds on these by incorporating intra-reflection, allowing LLMs to critically assess and refine reasoning during task execution for enhanced adaptability.

Reflection in LLMs. LLM frameworks increasingly integrate reflection mechanisms to enhance performance and adaptability. For example, Self-refine [Madaan *et al.*, 2024] employs iterative self-critique and refinement, while Reflexion [Shinn *et al.*, 2024] incorporates self-monitoring and feedback loops within the ReAct framework. DFSDT [Qin *et al.*, 2023] advances reflection by enabling models to re-evaluate and re-plan based on their complete history of previous error paths and information, thereby broadening the explored solution space. CRITIC [Gou *et al.*, 2023] leverages external tools for output validation and subsequent improvement. While these approaches predominantly emphasize inter-reflection or post-hoc refinement, MIRROR innovates by stressing continuous intra-reflection during all reasoning phases, facilitating more dynamic and context-aware adaptations in real time.

LLM-Powered Multi-Agent Systems. Single-agent LLM systems face limitations such as hallucinations, restricted context, and lack of collaboration in real-world applications [Huang *et al.*, 2024]. LLM-powered multi-agent systems (MAS) offer a promising solution, where multiple autonomous agents with unique strategies collaborate to manage complex, dynamic tasks beyond single-agent capabilities [Zhuge *et al.*, 2023]. This collaborative approach leverages distinct agent strengths through communication. Notable examples include AutoGen [Wu *et al.*, 2024], MetaGPT [Hong *et al.*, 2024], and OpenHands [Wang *et al.*, 2024b], which utilize Agent-Computer Interfaces for enhanced problem-solving. Our approach extends these MAS frameworks by integrating both intra-reflection and inter-reflection mechanisms, aiming to better address real-world complexities through maximized collaborative potential and adaptive reasoning.

3 Methodology

The MIRROR framework presents a structured approach to enhance LLM capabilities in tool learning. It leverages a specialized multi-agent system synergistically combined with unique intra-reflection and inter-reflection mechanisms. The overarching goal is to optimize the decomposition of complex tasks, the selection and parameterization of tools, and the final generation of answers, while proactively minimizing errors and enabling continuous learning.

3.1 Task Definition

Tool learning tasks involve an LLM interacting with external tools/APIs. Given a task description (Q) and a tool library ($T = \{n, d, p\}$, where n is name, d is description, p is parameters), the system must autonomously select the optimal tool ($t \in T$) and its parameters (p_t). These are passed to an Executor, and the resulting output is synthesized into a comprehensive answer.

3.2 Framework Overview

MIRROR's core innovation is **intra-reflection**, a distinct contribution enabling meaningful self-evaluation *within a single agent during its generation phase and before action execution or handoff to another agent*. This contrasts with methods relying on external agent feedback or post-execution reflection. Intra-reflection functions as a formalized prompting technique, universally applicable across tasks and models,

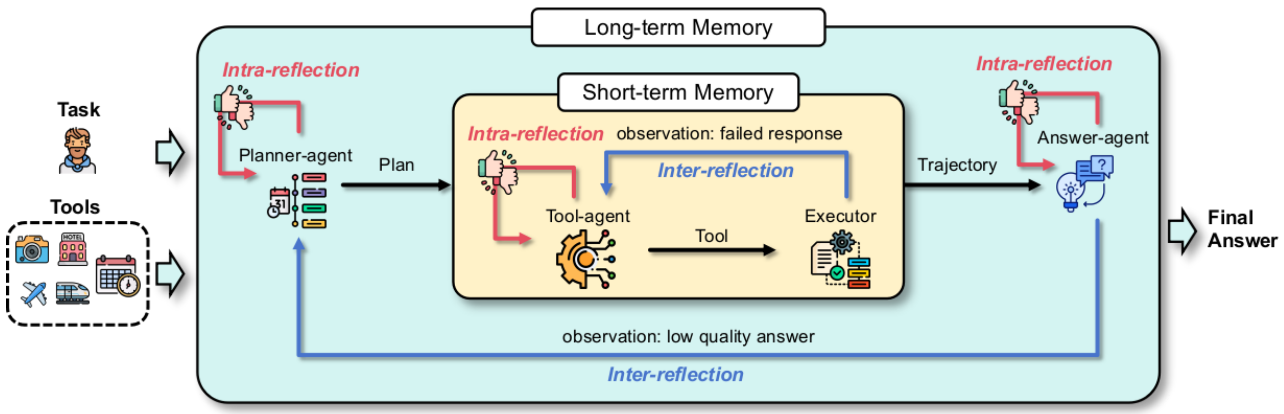


Figure 2: Overview of the MIRROR framework. Given a task and candidate tools, the Planner Agent decomposes the complex task into subtasks. The Tool Agent selects appropriate tools and parameters for each, which are passed to the Executor for execution. The Answer Agent then utilizes the entire execution trajectory to produce the final answer. Critically, each agent undergoes intra-reflection before passing its output, and a dual-memory architecture facilitates inter-reflection.

embedding evaluation criteria directly into the agent's prompt to guide its reasoning and output generation. This systematic approach facilitates proactive error prevention. This overall framework, which deeply embeds intra-reflection within each agent and also integrates inter-reflection for comprehensive learning, is visually outlined in Figure 2.

3.3 Multi-Agent System

MIRROR integrates three distinct, specialized agents:

Planner Agent: Decomposes complex tasks into a topologically structured sequence of manageable subtasks, considering dependencies and execution order. It utilizes Long-Term Memory to refine decomposition strategies based on outcomes from previous attempts within the current task.

Tool Agent: Selects appropriate tools and parameters for each subtask. Its decision-making is guided by Short-Term Memory, which includes execution failure information and the agent's own intra-reflection outputs from prior attempts on the current subtask, helping to avoid repeated errors. Successful subtask results are stored for subsequent use.

Answer Agent: Synthesizes individual results from all completed subtasks, considering the entire execution trajectory, to produce a coherent and comprehensive final answer to the original task.

These agents engage in collaborative interactions, where the validated output of one agent becomes the input for the next. This structured collaboration, augmented by reflection, aims to achieve solutions that surpass the capabilities of any single agent operating in isolation.

3.4 Intra-Reflection

A cornerstone of the MIRROR framework, the intra-reflection process is implemented by each agent *before* its output is passed to the next agent or the Executor. This ensures a thorough self-evaluation, minimizing errors and optimizing the quality of inter-agent information flow. Each agent performs a self-assessment based on its specific role and predefined quality thresholds ($\theta_p, \theta_t, \theta_a$). This mechanism does not depend on additional LLMs or external evaluation agents, functioning as an inherent part of the agent's generation cycle.

Planner Agent Intra-Reflection: After generating an initial task decomposition, the Planner Agent critically evaluates this plan. The assessment focuses on its completeness, efficiency, and coherence. It also considers if subtasks can be further optimized. The agent integrates lessons from prior experiences within the current task to enhance decomposition quality. If the internal evaluation score surpasses the threshold θ_p, the plan is deemed sufficiently robust and proceeds to the Tool Agent. Otherwise, the plan is iteratively revised until it meets the required quality standard. This iterative refinement ensures that only well-optimized and coherent plans advance, significantly improving the framework's overall effectiveness and resource utilization.

Tool Agent Intra-Reflection: Upon receiving a subtask and selecting a tool and parameters, the Tool Agent performs intra-reflection. It evaluates these choices based on how well they meet the specific subtask requirements, whether they incorporate lessons from past attempts (within the current task, leveraging STM), and their alignment with the subtask's objectives. The agent assigns a score reflecting the expected effectiveness of its choices. Selections scoring above threshold θ_t are passed to the Executor for execution. Those scoring below are revised (the agent may choose a different tool or adjust parameters) to better align with subtask goals. This process optimizes tool and strategy selection at a granular level, enhancing the framework's operational efficiency and increasing the probability of successful subtask completion.

Answer Agent Intra-Reflection: After synthesizing the results from all subtasks into a comprehensive solution, the Answer Agent conducts a final intra-reflection. This evaluation focuses on the completeness of the answer, its integration of key actions and observations from the entire trajectory, and the clarity of its presentation. The agent assigns a score to its output. If this score meets or exceeds the threshold θ_a, the task is considered successfully completed. If the score falls below θ_a, indicating a suboptimal answer, the entire task trajectory (including the plan, tool uses, reflections, and the deficient answer) is stored in long-term memory. This stored information is then utilized by the Planner Agent to refine its strategy for the current task, facilitating an improved attempt at successful completion and promoting continuous, task-specific adaptation.

3.5 Inter-Reflection

MIRROR employs a dual-memory architecture to facilitate inter-reflection:

Short-Term Memory (STM): Utilized by the Tool Agent for subtask tool/parameter selection. It records execution failures and the agent's intra-reflection outputs, prompting adjustments. STM is reset upon successful subtask completion to maintain relevance for new selections.

Long-Term Memory (LTM): This memory is **task-specific**. It stores the entire trajectory for the current task (decomposition plan, tool selections, execution results, solution quality). LTM is updated if execution fails or if the Answer Agent's final output is below θ_a. It resets upon overall task completion, enabling the system to learn from successes and failures *within the current task* for progressive strategy refinement.

This dual-memory architecture allows MIRROR to continually adapt and refine its performance through structured experiential learning.

3.6 Agent Collaboration

Agent collaboration critically relies on intra-reflection for high-quality inter-agent handoffs. The Planner Agent passes decompositions, validated for coherence and feasibility by its intra-reflection, to the Tool Agent. The Tool Agent, after its own reflection-refined tool and parameter selection aimed at optimal subtask execution, then directs the process. Results subsequently flow to the Answer Agent for careful synthesis and final quality assurance, also guided by its intra-reflection.

This reflection-gated workflow minimizes error propagation via early self-correction and optimizes inter-agent communication with reliable information exchange. Inter-reflection, through dual memory, further enhances collaboration: STM aids Tool Agent's immediate adaptation to execution challenges, while LTM enables the Planner Agent to refine strategies from current task trajectories.

Integrating both reflection types creates a self-optimizing system. This enhances individual agent efficiency and overall framework performance, robustness, and adaptability for complex tasks. Thus, MIRROR continuously learns and evolves. While dual-reflection increases token processing, this is justified by significant, experimentally validated performance gains, making it a worthwhile trade-off.

4 Experiment

To evaluate our proposed framework, we conducted comprehensive experiments as detailed below.

4.1 Setup

Benchmarks We utilized two benchmarks:

- **StableToolBench** [Guo *et al.*, 2024]: An extension of ToolBench [Qin *et al.*, 2023], this benchmark assesses LLM function invocation capabilities through executable tests.
- **TravelPlanner** [Xie *et al.*, 2024]: This benchmark evaluates agent proficiency in real-world planning and tool interactions. Due to computational constraints, we used its validation set in a two-stage mode to assess tool utilization and planning capabilities.

Metrics

- **StableToolBench**: Metrics include Pass Rate (percentage of tasks successfully completed) and Win Rate (comparison against GPT-3.5+ReAct via LLM evaluation).
- **TravelPlanner**: Metrics include Delivery Rate (plan completion), Commonsense and Hard Constraint Pass Rates (rule adherence), and Final Pass Rate (overall plan feasibility).

Model Our experiments used the following foundational LLMs: GPT-3.5 Turbo [Ouyang *et al.*, 2022] (*gpt-35-turbo-0125*), GPT-4o Mini [Achiam *et al.*, 2023] (*gpt-4o-mini-2024-07-18*), GPT-4o [Achiam *et al.*, 2023] (*gpt-4o-2024-05-13*), Claude 3 Haiku [Anthropic, 2024], and Qwen2.5-72B [Team, 2024]. For evaluation consistency on StableToolBench, we employed GPT-4 Turbo [Achiam *et al.*, 2023] (*gpt-4-turbo-2024-04-09*). GPT-4o served as the parsing model for TravelPlanner.

Baselines We compared our method against six baselines:

- **ReAct** [Yao *et al.*, 2022]: Alternates reasoning and acting within an LLM for interpretable trace generation and task-specific action execution.
- **DFSDT** [Qin *et al.*, 2023]: Constructs and traverses decision trees for systematic re-evaluation and re-planning based on previous error paths.
- **Reflexion** [Shinn *et al.*, 2024]: Employs self-reflection on environmental interactions to improve decision-making.
- **Smurfs** [Chen *et al.*, 2024]: A multi-agent framework assigning distinct roles to agents via prompting for collaborative problem-solving.
- **ToolLlama-2** [Qin *et al.*, 2023]: A model fine-tuned on the ToolBench dataset.
- **ToolGen** [Wang *et al.*, 2024a]: Represents tools as unique vocabulary tokens for direct tool call generation via a three-stage training process, bypassing separate retrieval steps.

4.2 Main Results

Experimental results demonstrate MIRROR's consistent performance advantages across both benchmarks. Across both benchmarks, MIRROR surpasses existing baselines. On StableToolBench, it demonstrates superiority with diverse LLMs (open-source and proprietary), notably outperforming supervised fine-tuning (SFT) methods like

Table 1: Main results on the StableToolBench benchmark. The best-performing method under each LLM core is highlighted in **bold**. Pass Rate (%) and Win Rate (%) are the evaluation metrics, with Win Rate calculated by comparing each method to GPT-3.5+ReAct.

LLM Core	Method	G1-Inst.		G1-Cat.		G1-Tool.		G2-Inst.		G2-Cat.		G3-Inst.		Avg.	
		Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.
GPT-3.5 Turbo	ReAct	$47.2_{\pm0.5}$	/	$42.9_{\pm0.8}$	/	$41.8_{\pm0.5}$	/	$46.1_{\pm0.2}$	/	$43.1_{\pm0.3}$	/	$42.1_{\pm1.5}$	/	$43.9_{\pm0.5}$	/
	DFSDT	$64.5_{\pm0.4}$	66.9	$61.2_{\pm0.8}$	64.7	$64.7_{\pm0.7}$	68.4	$59.3_{\pm0.2}$	60.4	$61.6_{\pm1.2}$	64.5	$57.4_{\pm1.3}$	59.0	$61.5_{\pm0.8}$	64.0
	Reflexion	$75.8_{\pm1.0}$	65.6	$73.0_{\pm1.5}$	76.5	$75.3_{\pm1.2}$	70.3	$72.5_{\pm2.8}$	72.6	$73.0_{\pm0.6}$	72.6	$58.5_{\pm0.8}$	49.2	$71.4_{\pm1.3}$	67.8
	Smurfs	$75.9_{\pm1.1}$	74.2	$82.6_{\pm0.4}$	81.7	$72.2_{\pm0.9}$	71.5	$76.6_{\pm1.4}$	73.6	$79.6_{\pm0.2}$	79.8	$73.0_{\pm2.0}$	68.9	$76.7_{\pm1.0}$	75.0
	MIRROR	$\mathbf{82.5_{\pm0.5}}$	**76.1**	$\mathbf{86.3_{\pm1.3}}$	**87.6**	$\mathbf{84.9_{\pm0.6}}$	**79.7**	$\mathbf{78.8_{\pm1.0}}$	**87.7**	$\mathbf{82.4_{\pm1.0}}$	**83.1**	$\mathbf{87.4_{\pm1.5}}$	**80.3**	$\mathbf{83.7_{\pm1.0}}$	**82.4**
Claude 3 Haiku	ReAct	$54.9_{\pm0.0}$	49.1	$46.1_{\pm0.0}$	47.7	$54.0_{\pm0.6}$	55.1	$55.3_{\pm1.1}$	52.8	$52.2_{\pm1.0}$	50.8	$65.8_{\pm0.8}$	55.7	$54.7_{\pm0.6}$	51.9
	DFSDT	$67.9_{\pm0.8}$	55.2	$68.8_{\pm0.9}$	65.4	$76.8_{\pm0.6}$	68.4	$65.9_{\pm1.6}$	67.9	$69.8_{\pm0.9}$	57.3	$69.4_{\pm0.8}$	57.4	$69.8_{\pm0.9}$	61.9
	Reflexion	$75.8_{\pm0.7}$	68.7	$76.9_{\pm0.4}$	85.0	$80.6_{\pm0.4}$	77.2	$73.4_{\pm1.1}$	77.4	$73.8_{\pm1.3}$	76.6	$73.0_{\pm1.2}$	70.5	$75.6_{\pm0.9}$	75.9
	Smurfs	$68.4_{\pm0.5}$	65.0	$81.2_{\pm1.2}$	81.0	$80.6_{\pm1.3}$	76.6	$72.5_{\pm1.2}$	71.7	$75.4_{\pm0.3}$	69.4	$74.3_{\pm0.8}$	60.7	$75.4_{\pm0.9}$	70.7
	MIRROR	$\mathbf{78.3_{\pm0.8}}$	**74.8**	$\mathbf{81.9_{\pm0.6}}$	**86.9**	$\mathbf{85.2_{\pm0.8}}$	**79.7**	$\mathbf{80.2_{\pm1.7}}$	**81.1**	$\mathbf{78.1_{\pm1.4}}$	**81.5**	$\mathbf{81.4_{\pm0.4}}$	**75.4**	$\mathbf{80.9_{\pm1.0}}$	**79.9**
Qwen2.5-72B	ReAct	$53.0_{\pm1.0}$	52.8	$63.6_{\pm0.4}$	69.3	$60.8_{\pm0.8}$	62.0	$60.8_{\pm0.7}$	67.9	$57.5_{\pm0.7}$	60.5	$54.9_{\pm2.7}$	52.5	$58.4_{\pm1.1}$	60.8
	DFSDT	$62.9_{\pm0.4}$	60.7	$66.3_{\pm0.7}$	72.5	$62.4_{\pm0.1}$	62.7	$64.0_{\pm1.2}$	66.0	$65.3_{\pm0.9}$	64.5	$58.5_{\pm1.4}$	52.5	$63.2_{\pm0.8}$	63.2
	Reflexion	$76.4_{\pm0.3}$	70.6	$75.7_{\pm0.6}$	81.0	$75.9_{\pm0.9}$	77.2	$72.3_{\pm0.2}$	83.0	$77.2_{\pm0.7}$	81.5	$79.0_{\pm0.8}$	68.9	$76.1_{\pm0.6}$	77.0
	Smurfs	$77.8_{\pm0.9}$	73.6	$81.9_{\pm0.2}$	88.9	$78.0_{\pm0.1}$	78.5	$76.7_{\pm0.6}$	84.0	$79.6_{\pm0.7}$	77.4	$88.3_{\pm0.4}$	68.9	$80.4_{\pm0.5}$	78.6
	MIRROR	$\mathbf{83.2_{\pm0.8}}$	**75.5**	$\mathbf{85.3_{\pm0.3}}$	**90.8**	$\mathbf{81.8_{\pm0.3}}$	**80.4**	$\mathbf{77.2_{\pm0.3}}$	**86.8**	$\mathbf{80.9_{\pm1.1}}$	**83.9**	$\mathbf{89.1_{\pm0.8}}$	**85.2**	$\mathbf{82.9_{\pm0.6}}$	**83.8**
GPT-4o	ReAct	$58.6_{\pm1.1}$	63.8	$61.5_{\pm0.7}$	64.1	$62.2_{\pm0.3}$	59.5	$61.5_{\pm1.6}$	67.0	$54.7_{\pm1.9}$	62.9	$52.5_{\pm0.7}$	50.8	$58.5_{\pm1.1}$	61.4
	DFSDT	$68.7_{\pm1.9}$	66.3	$66.1_{\pm0.7}$	67.3	$73.9_{\pm0.5}$	72.8	$67.0_{\pm1.0}$	69.8	$65.9_{\pm0.7}$	74.2	$67.5_{\pm1.0}$	72.1	$68.2_{\pm1.0}$	70.4
	Reflexion	$76.4_{\pm0.3}$	74.8	$76.7_{\pm0.9}$	87.6	$74.4_{\pm0.4}$	78.5	$71.5_{\pm0.8}$	78.3	$73.5_{\pm1.1}$	79.0	$72.7_{\pm0.8}$	70.5	$74.2_{\pm0.7}$	78.1
	Smurfs	$75.1_{\pm1.3}$	73.6	$79.5_{\pm1.1}$	80.4	$81.6_{\pm1.2}$	76.6	$77.0_{\pm0.4}$	83.0	$76.9_{\pm1.0}$	78.2	$84.4_{\pm0.7}$	78.7	$79.1_{\pm1.0}$	78.4
	MIRROR	$\mathbf{82.5_{\pm0.3}}$	**77.3**	$\mathbf{83.3_{\pm0.5}}$	**91.5**	$\mathbf{86.2_{\pm0.7}}$	**86.1**	$\mathbf{77.2_{\pm0.2}}$	**84.9**	$\mathbf{80.8_{\pm1.0}}$	**85.5**	$\mathbf{88.3_{\pm1.5}}$	**86.9**	$\mathbf{83.1_{\pm0.7}}$	**85.4**
ToolLlama-2	SFT	$54.0_{\pm0.9}$	48.5	$54.1_{\pm0.4}$	50.3	$46.1_{\pm2.1}$	49.4	$37.9_{\pm1.5}$	46.2	$43.1_{\pm0.7}$	46.8	$41.3_{\pm0.8}$	37.7	$46.1_{\pm1.1}$	46.5
ToolGen	SFT	$51.2_{\pm0.9}$	47.9	$47.8_{\pm0.6}$	45.1	$52.4_{\pm0.3}$	43.7	$48.7_{\pm2.0}$	52.8	$42.3_{\pm0.3}$	45.2	$35.5_{\pm1.0}$	29.5	$46.3_{\pm0.9}$	44.0

Table 2: Evaluation results on TravelPlanner benchmark. MIRROR consistently outperforms ReAct across different LLM cores.

LLM Core	Method	Delivery Rate	Commonsense		Hard Constraint		Final Pass Rate
			Micro	Macro	Micro	Macro	
GPT-4o Mini	ReAct	75.0	54.7	0	0	0	0
	MIRROR	**96.1**	**69.0**	**3.3**	**5.2**	**2.2**	0
Qwen2.5-72B	ReAct	79.4	56.0	0.6	3.3	2.2	0
	MIRROR	**95.6**	**64.9**	**6.7**	**12.9**	**9.4**	**1.7**
GPT-4o	ReAct	85.6	61.4	5.0	15.2	7.8	2.2
	MIRROR	**100**	**73.2**	**13.9**	**27.6**	**13.3**	2.2

ToolLlama-2 and ToolGen (both with Pass Rates around 46%), underscoring SFT's limitations in complex tool use. This superior performance extends to TravelPlanner, where MIRROR consistently leads ReAct in Delivery Rate, Commonsense Pass Rate, and Hard Constraint Pass Rate. The efficacy of MIRROR is attributed to its dual-reflection architecture, which incorporates intra-reflection for immediate error detection and inter-reflection for strategic optimization. This approach shows clear advantages over Reflexion's single inter-reflection mechanism. Despite these gains, achieving a high Final Pass Rate on TravelPlanner remains challenging due to the inherent complexity of managing information and constraints in multifaceted planning.

Performance across different LLM cores. MIRROR's effectiveness varies with the LLM core and benchmark. Across both StableToolBench (Table 1) and TravelPlanner (Table 2), MIRROR consistently enhanced performance over baselines across all evaluated LLMs. On StableToolBench, this led to average Pass Rate gains (over the next best method) from **2.5%** (Qwen2.5-72B) to **7.0%** (GPT-3.5 Turbo), and Win Rate gains (vs. GPT-3.5+ReAct) from **5.2%** (Qwen2.5-72B) to **9.2%** (Claude 3 Haiku). On TravelPlanner, which tests complex multi-step planning, MIRROR increased Delivery Rates by amounts ranging from **14.4%** (with GPT-4o) to **21.1%** (with GPT-4o Mini). Further, it yielded absolute improvements in macro Pass Rates for Commonsense (from **3.3%** with GPT-4o Mini to **8.9%** with GPT-4o) and Hard Constraints (from **2.2%** with GPT-4o Mini to **7.2%** with Qwen2.5-72B).

4.3 Ablation Study

For the ablation studies, we utilized GPT-4o Mini as the LLM core, with evaluations conducted on StableToolBench using Pass Rate metric.

Effectiveness of Intra-reflection. Intra-reflection mechanisms are critical for MIRROR's performance, as shown in Table 3. The full MIRROR configuration (average Pass Rate: **85.7%**) significantly outperforms variants lacking these components. Removing Planner, Tool, or Answer Agent intra-reflection drops the average Pass Rate to **82.1%**, **81.3%**, and **79.4%**, respectively. Ablating all intra-reflection mechanisms causes the largest decline (to **78.7%**, a **7.0%** drop), particularly impacting complex instruction-heavy tasks like G3-Inst. (a **9.9%** decrease in this category). These findings underscore the meaningful contribution of each agent's intra-reflection and their combined role in enhancing task decomposition, tool selection, and output generation.

Table 3: Ablation study of MIRROR components and Tool Agent strategies. Default MIRROR (top row, bolded) uses a prompt-based Tool Agent; the FC configuration employs function calling for the Tool Agent. All values are Pass Rates (%). Abbreviations: Intra. = Intra-reflection, Inter. = Inter-reflection, FC = Function Calling.

Configuration	G1-Inst.	G1-Cat.	G1-Tool.	G2-Inst.	G2-Cat.	G3-Inst.	Avg.
MIRROR	**84.9**$_{\pm 1.2}$	**87.9**$_{\pm 0.5}$	**89.5**$_{\pm 0.8}$	**82.7**$_{\pm 0.6}$	**83.1**$_{\pm 0.7}$	**86.1**$_{\pm 0.7}$	**85.7**$_{\pm 0.8}$
FC	78.6$_{\pm 0.4}$	83.7$_{\pm 0.5}$	79.3$_{\pm 0.3}$	80.2$_{\pm 0.7}$	85.6$_{\pm 2.2}$	91.8$_{\pm 0.7}$	83.2$_{\pm 0.8}$
w/o Planner Intra.	83.2$_{\pm 0.5}$	83.3$_{\pm 0.3}$	87.0$_{\pm 0.7}$	76.7$_{\pm 1.1}$	81.5$_{\pm 0.9}$	80.9$_{\pm 1.4}$	82.1$_{\pm 0.8}$
w/o Tool Intra.	80.4$_{\pm 0.0}$	84.6$_{\pm 0.5}$	83.3$_{\pm 0.8}$	79.7$_{\pm 0.4}$	80.8$_{\pm 0.5}$	79.0$_{\pm 0.4}$	81.3$_{\pm 0.4}$
w/o Answer Intra.	81.7$_{\pm 0.6}$	80.8$_{\pm 0.6}$	83.1$_{\pm 0.9}$	74.8$_{\pm 0.8}$	78.6$_{\pm 1.0}$	77.6$_{\pm 1.7}$	79.4$_{\pm 0.9}$
w/o Intra.	78.0$_{\pm 1.1}$	79.0$_{\pm 0.3}$	81.3$_{\pm 1.2}$	76.7$_{\pm 0.4}$	80.8$_{\pm 0.8}$	76.2$_{\pm 1.3}$	78.7$_{\pm 0.9}$
w/o Long-term Memory	83.5$_{\pm 0.1}$	86.2$_{\pm 0.9}$	85.7$_{\pm 1.3}$	79.4$_{\pm 1.0}$	81.0$_{\pm 0.3}$	83.9$_{\pm 1.4}$	83.3$_{\pm 0.8}$
w/o Short-term Memory	81.1$_{\pm 0.6}$	81.3$_{\pm 0.6}$	84.7$_{\pm 1.9}$	78.3$_{\pm 2.0}$	81.9$_{\pm 0.0}$	79.0$_{\pm 1.4}$	81.1$_{\pm 1.1}$
w/o Inter.	80.4$_{\pm 0.9}$	80.2$_{\pm 0.4}$	84.5$_{\pm 0.9}$	75.9$_{\pm 1.5}$	79.7$_{\pm 2.2}$	82.5$_{\pm 0.8}$	80.5$_{\pm 1.1}$
w/o Reflection	78.6$_{\pm 0.4}$	80.3$_{\pm 1.1}$	82.4$_{\pm 1.7}$	72.6$_{\pm 0.4}$	77.4$_{\pm 0.3}$	75.4$_{\pm 1.3}$	77.8$_{\pm 0.9}$

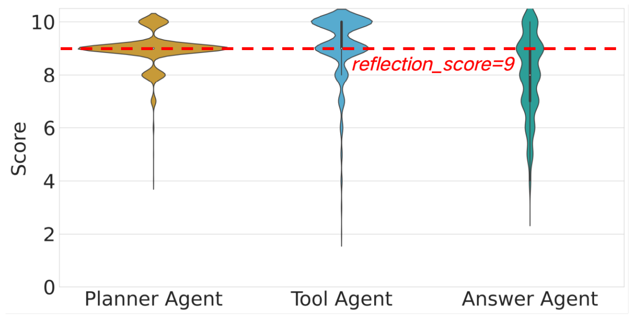


Figure 3: The distribution of intra-reflection scores across three agents.

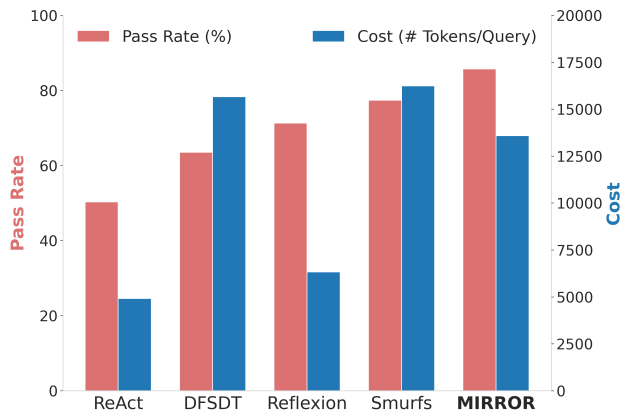


Figure 4: Comparison of MIRROR and other baselines regarding performance and cost.

Intra-reflection score distribution. Figure 3 reveals distinct intra-reflection score distributions for the three agents. Scores for the Answer Agent are more uniformly distributed, with a notable proportion below the threshold, contrasting with the higher-clustering scores of the Planner and Tool Agents. This unique distribution for the Answer Agent aligns with findings from our ablation study (Table 3), where removing its intra-reflection mechanism most significantly impacts system performance. Such a significant impact underscores its critical role in filtering substandard outputs to preserve overall system quality.

Score threshold selection. Preliminary experiments indicated that threshold scores (e.g., Planner Agent at the 90th percentile) yield strong, generalizable results. Thresholds in the 80-90th percentile range effectively balance efficiency and quality. For the Planner Agent, specific threshold values of 7, 8, and 9 yielded Pass Rates of 82.3%, 83.3%, and 85.7%, respectively.

Impact of Inter-reflection. Disabling inter-reflection, as per Table 3, reduces MIRROR's average Pass Rate from **85.7%** to **80.5%**. This mechanism comprises long-term and short-term memory. Ablating long-term memory (**83.3%**) affects consistent planning and strategic decision-making. Removing short-term memory causes a larger drop (**81.1%**), underscoring its critical role in immediate action selection, local decision sequences, and maintaining coherence within task episodes. Thus, inter-reflection is a key architectural component for coordinating tactical and strategic decisions across diverse tasks.

Rounds of Inter-reflection. The number of inter-reflection rounds significantly influences both efficacy and token consumption. Optimal performance is achieved with 5 rounds, yielding an **85.7%** Pass Rate at a cost of 13.6k tokens per query. Fewer rounds, such as 3, result in a lower Pass Rate of 83.4% although with marginally fewer tokens (12.8k per query). Conversely, increasing to 7 rounds not only elevates token usage substantially to 17.2k per query but also leads to a decline in effectiveness, with the Pass Rate dropping to 82.3%. This performance degradation with excessive reflection is likely attributable to induced redundancy and potential reasoning errors.

Prompt vs. Function Calling. In this analysis, we replace the Tool Agent's prompt-based approach with the LLM's function calling (FC) mode to evaluate its impact on performance. Table 3 shows that the prompt-based strategy achieves a higher average Pass Rate of **85.7%** compared to

83.2% for FC. This demonstrates that the function calling capability of existing LLMs still requires further improvement.

Token efficiency analysis. Figure 4 shows MIRROR achieving the highest Pass Rate while using fewer tokens per query than DFSDT and Smurfs. While ReAct and Reflexion are more token-efficient, they yield lower Pass Rates. MIRROR's efficiency is attributed to its dual-reflection architecture: intra-reflection provides early error detection and quality control, and inter-reflection optimizes decision paths. This integrated design enables high performance with minimized computational overhead.

Discussion on LLM core capabilities. We utilized *GPT-4o-2024-05-13* and *GPT-4o-mini-07-18*. Given their differences in both size and training datasets, tool-learning capability is not dictated by model size alone. This is evident on StableToolBench, where MIRROR with *GPT-4o-mini-07-18* achieved an average Pass Rate of **85.7%** (Table 3), outperforming *GPT-4o-2024-05-13* with MIRROR (**83.1%** average Pass Rate, Table 1). Similarly, on the external BFCL-v1 benchmark [Yan *et al.*, 2024] (updated 08/11/2024), *GPT-4o-mini-07-18* (87.35 accuracy) also surpassed *GPT-4o-2024-05-13* (84.12 accuracy).

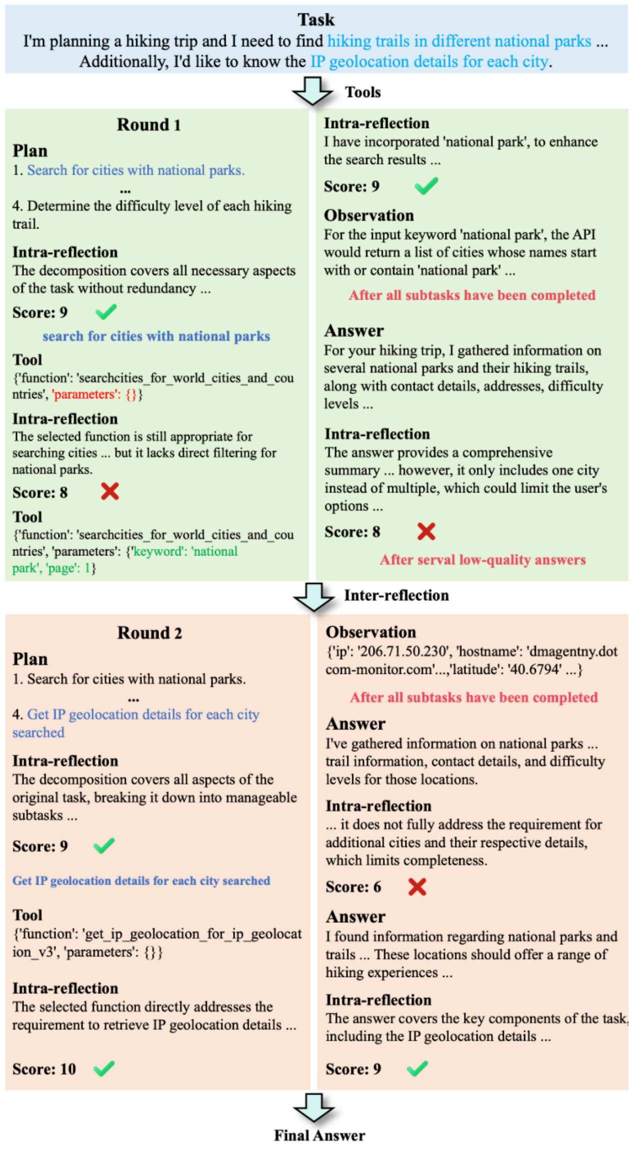
4.4 Case Study

Figure 5 illustrates MIRROR's dual-reflection orchestrating complex task execution. In Round 1, after the Planner Agent decomposes the task, the Tool Agent's initial parameter selection for subtask 1 is flawed. MIRROR's tool-level intra-reflection swiftly intervenes; operating without specific expected results for the subtask, it employs standard, generalizable criteria (e.g., function alignment, experiential learning) for an objective, pre-execution correction of the parameters. Despite this successful local fix and completion of all planned subtasks, the final answer remains suboptimal due to the Planner Agent's inadequate initial task decomposition.

Through inter-reflection on the complete execution trajectory, MIRROR identifies this planning limitation, prompting the Planner Agent to revise its task decomposition strategy. In the second round, the Planner Agent adds the crucial subtask 4, which was overlooked initially. This more comprehensive task decomposition, combined with accurate function selection, leads to a high-quality solution, demonstrating how MIRROR's reflection mechanisms effectively address errors at both planning and execution levels.

4.5 Discussion

While MIRROR demonstrates promising results, limitations in its generalizability and applicability persist. Experimental scope was constrained by budget (few LLMs tested), and MIRROR's efficacy is inherently tied to base LLM capabilities. Moreover, like similar methods, it faces challenges with interdependent constraints in complex scenarios, and its current task-specific learning limits generalization. To mitigate these issues, future work will explore hierarchical reflection for enhanced management of complex tasks, alongside cross-task memory architectures (extending LTM) to improve generalization and adaptability.



Task
I'm planning a hiking trip and I need to find hiking trails in different national parks ...
Additionally, I'd like to know the IP geolocation details for each city.

Tools

Round 1

Plan
1. Search for cities with national parks.
...
4. Determine the difficulty level of each hiking trail.

Intra-reflection
The decomposition covers all necessary aspects of the task without redundancy ...

Score: 9 ✓

search for cities with national parks

Tool
{'function': 'searchcities_for_world_cities_and_countries', 'parameters': {}}

Intra-reflection
The selected function is still appropriate for searching cities ... but it lacks direct filtering for national parks.

Score: 8 ✗

Tool
{'function': 'searchcities_for_world_cities_and_countries', 'parameters': {'keyword': 'national park', 'page': 1}}

Intra-reflection
I have incorporated 'national park', to enhance the search results ...

Score: 9 ✓

Observation
For the input keyword 'national park', the API would return a list of cities whose names start with or contain 'national park' ...

After all subtasks have been completed

Answer
For your hiking trip, I gathered information on several national parks and their hiking trails, along with contact details, addresses, difficulty levels ...

Intra-reflection
The answer provides a comprehensive summary ... however, it only includes one city instead of multiple, which could limit the user's options ...

Score: 8 ✗

After serval low-quality answers

Inter-reflection

Round 2

Plan
1. Search for cities with national parks.
...
4. Get IP geolocation details for each city searched

Intra-reflection
The decomposition covers all aspects of the original task, breaking it down into manageable subtasks ...

Score: 9 ✓

Get IP geolocation details for each city searched

Tool
{'function': 'get_ip_geolocation_for_ip_geolocation_v3', 'parameters': {}}

Intra-reflection
The selected function directly addresses the requirement to retrieve IP geolocation details ...

Score: 10 ✓

Observation
{'ip': '206.71.50.230', 'hostname': 'dmagentny.dotcom-monitor.com'...,'latitude': '40.6794' ...}

After all subtasks have been completed

Answer
I've gathered information on national parks ... trail information, contact details, and difficulty levels for those locations.

Intra-reflection
... it does not fully address the requirement for additional cities and their respective details, which limits completeness.

Score: 6 ✗

Answer
I found information regarding national parks and trails ... These locations should offer a range of hiking experiences ...

Intra-reflection
The answer covers the key components of the task, including the IP geolocation details ...

Score: 9 ✓

Final Answer

Figure 5: A case study demonstrating MIRROR's solution process.

5 Conclusion

We present MIRROR, an innovative framework that advances tool use capabilities through the systematic integration of pre-execution assessment (intra-reflection) and post-execution learning (inter-reflection) in a collaborative multi-agent environment. Rigorous evaluations conducted on both StableToolBench and TravelPlanner benchmarks demonstrate that our dual-reflection approach substantially enhances tool interaction reliability and task execution capabilities. These findings not only validate MIRROR's effectiveness but also underscore the fundamental role of reflection mechanisms in advancing agents' capability to interact with external environments through tools.

Acknowledgements

We thank the reviewers for their helpful feedback and suggestions, which improved this paper. This research is supported by Artificial Intelligence-National Science and Technology Major Project 2023ZD0121200 and National Natural Science Foundation of China under Grant 62222212.

References

[Achiam *et al.*, 2023] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[Anthropic, 2024] Anthropic. Claude 3 family, 2024. Accessed: 2024-10-14.

[Besta *et al.*, 2024] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.

[Brown, 2020] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[Chen *et al.*, 2022] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[Chen *et al.*, 2024] Junzhi Chen, Juhao Liang, and Benyou Wang. Smurfs: Leveraging multiple proficiency agents with context-efficiency for tool planning. *arXiv preprint arXiv:2405.05955*, 2024.

[Dubey *et al.*, 2024] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[Gou *et al.*, 2023] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[Guo *et al.*, 2024] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2024.

[Hong *et al.*, 2024] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.

[Huang *et al.*, 2024] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

[Komeili, 2021] M Komeili. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*, 2021.

[Lewis *et al.*, 2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

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

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

[Petroni *et al.*, 2020] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. Kilt: a benchmark for knowledge intensive language tasks. *arXiv preprint arXiv:2009.02252*, 2020.

[Qin *et al.*, 2023] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

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

[Sun *et al.*, 2023] Jiankai Sun, Chuanyang Zheng, Enze Xie, Zhengying Liu, Ruihang Chu, Jianing Qiu, Jiaqi Xu, Mingyu Ding, Hongyang Li, Mengzhe Geng, et al. A survey of reasoning with foundation models. *arXiv preprint arXiv:2312.11562*, 2023.

[Team *et al.*, 2023] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[Team, 2024] Qwen Team. Qwen2.5: A party of foundation models, September 2024.

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

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

[Wang *et al.*, 2024a] Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. *arXiv preprint arXiv:2410.03439*, 2024.

[Wang *et al.*, 2024b] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenDevin: An Open Platform for AI Software Developers as Generalist Agents, 2024.

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

[Wu *et al.*, 2024] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. In *COLM*, 2024.

[Xie *et al.*, 2024] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.

[Yan *et al.*, 2024] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. 2024.

[Yang *et al.*, 2023] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

[Yao *et al.*, 2022] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

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

[Zhang *et al.*, 2022] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[Zhuge *et al.*, 2023] Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*, 2023.

A Benchmark

To comprehensively evaluate our approach, we employ two complementary benchmarks: StableToolBench and TravelPlanner. StableToolBench, an enhanced version of ToolBench, tackles scalability and stability in tool-use evaluation via an innovative LLM simulator. It leverages 16,464 real-world APIs from RapidAPI and their execution traces to ensure reproducible evaluations. StableToolBench features six evaluation sets (G1-Inst., G1-Tool, G1-Cat., G2-Inst., G2-Cat., and G3-Inst.; with 163, 153, 158, 106, 124, and 61 queries respectively) designed to test generalization across three increasingly complex scenarios: single-tool, intra-category multi-tool, and intra-collection multi-tool instructions.

Complementing StableToolBench's API interaction focus, TravelPlanner evaluates agent capabilities in complex, real-world planning scenarios. It offers a rich sandbox with approximately 4 million data entries and six specialized tools. TravelPlanner distinctively incorporates multiple real-world-mirroring constraint layers, including explicit user requirements (e.g., budget, accommodation) and implicit commonsense rules (e.g., logical routes, non-conflicting transport). Queries are organized into nine groups by travel duration (3, 5, or 7 days) and constraint complexity.

On TravelPlanner, our experiments primarily use its validation set (180 queries; 20 per group) for a balance of evaluation breadth and computational efficiency. Although its test set contains 1,000 diverse queries, the validation set is more practical for our large-scale LLM studies due to significant computational demands, while still ensuring statistical validity.

B Metrics

StableToolBench uses two primary metrics: Pass Rate and Win Rate. The Pass Rate quantifies the percentage of successfully completed tasks, indicating our framework's effectiveness in tool-learning. The Win Rate compares our approach against a baseline (GPT-3.5+ReAct) via LLM-based assessment: a "win" occurs if our solution is judged superior, with ties or losses otherwise.

TravelPlanner employs a multi-dimensional framework for its multi-constraint planning scenarios. Key metrics include:

- **Delivery Rate:** Successful plan completion within step limits.

- **Commonsense Constraint Pass Rate:** Adherence to implicit rules (e.g., logical routes, varied dining).
- **Hard Constraint Pass Rate:** Satisfaction of explicit user requirements (e.g., budget, accommodation).
- **Final Pass Rate:** Overall plan feasibility, combining all constraints.

TravelPlanner utilizes both micro (evaluating individual constraint satisfaction) and macro (assessing holistic compliance) evaluation strategies.

C Prompts

Planner Agent System Prompt

Act like a Planner Agent. You have extensive experience in decomposing complex tasks into manageable subtasks. You are skilled at learning from previous mistakes to avoid repeating errors and ensuring precise task decomposition. You excel in identifying root causes and addressing them effectively to enhance task completion.

Objective:

Decompose a complex task into manageable subtasks. Learn from previous failed trajectories to extract valuable lessons and ensure precise task decomposition. Ensure each subtask corresponds to a function from the provided function list, allowing for multiple calls to the same function if necessary.

Guidelines for Effective Task Decomposition:

1. **Analyze Failed Trajectories:**
   - Review previous failed task trajectories.
   - Identify each failure point and its cause.
   - Document lessons learned to understand root causes.
   - Consider the score indicating completion level and the reason for failure.
2. **Task Decomposition:**
   - Break down the complex task into multiple simple subtasks based on the given task.
   - Ensure each subtask is feasible, clearly defined, and focused on a single aspect of the task.
   - Ensure that the collection of subtasks covers the entirety of the original task scope without redundancy or overlap.
   - Consider dependencies and constraints among subtasks.
   - Assign each subtask to a function from the provided function list, allowing for multiple calls to the same function if necessary.
3. **Intra-Reflection:**
   - After decomposing the task into subtasks, reflect on the decomposition process and the decisions made.
   - Ask yourself:
     - Does the decomposition cover all aspects of the original task without redundancy or overlap?
     - Have all dependencies and constraints between subtasks been accurately accounted for?
     - Are there any subtasks or functions that could be further simplified or combined for better efficiency?
     - Have lessons from previous failures been applied effectively in this decomposition?
   - Based on this reflection, **assign a score** between 1 and 10, with 1 being poor and 10 being excellent. The score should reflect how well the task was decomposed, whether all dependencies were respected, and whether past mistakes were avoided.

Output Format:

- Present each subtask clearly and concisely, detailing the required steps, prerequisites, and expected outcomes.
- Indicate the corresponding function for each subtask from the function list, noting that functions may be reused.
- Include a **intra-reflection** section that documents the agent's assessment of its task decomposition and assigns a **score**.
- Provide the output in a JSON-parsable format:

```
{
  "nodes": [
    {
      "id": "node1",
      "status": 0,
      "subtask": "description of subtask",
      "function": "corresponding function name"
    },
    {
      "id": "node2",
      "status": 0,
      "subtask": "description of another subtask",
      "function": "corresponding function name"
    }
  ],
  "intra_reflection": {
    "evaluation": "Detailed reflection on the decomposition process.",
    "score": <score>
  }
}
```

Planner Agent User Prompt

Given Task:

`{task_description}`

Previous Failed Trajectories:

`{long_memory}`

Available Functions:

`{functions}`

Tool Agent System Prompt

Act like a Tool Agent. You are skilled in automatically selecting the most appropriate functions and parameters to solve a wide range of subtasks. You have advanced expertise in analyzing task trajectories, learning from failures, and dynamically adjusting based on prior results. Your ability to integrate outputs from related subtasks and consider lessons from past trajectories helps ensure precise execution.

Objective:

Your goal is to autonomously select appropriate functions and parameters from a predefined list to solve each subtask in a complex task chain. You must do this efficiently, accurately, and by learning from past mistakes.

Function Selection Guidelines:

1. **Analyze Failed Trajectories:**
   - Review previous task failures to identify where incorrect functions or parameters were chosen.
   - Determine the cause of each failure and document insights to prevent repeating mistakes.
   - Use these insights to inform your current decisions, particularly when selecting functions and parameters for the current subtask.
2. **Function Selection:**
   - Based on your reasoning and insights from failed trajectories, select the most suitable function from the provided function list.
   - Ensure that the selected function aligns with the specific requirements of the current subtask.
   - Take into account the outputs of previous subtasks and their influence on the current subtask to ensure proper function chaining.
   - Confirm that the selected function addresses dependencies or complements other functions in the task chain.
3. **Parameters Selection:**
   - Once the function is selected, choose parameters that best meet the subtask's specific requirements.
   - Align parameters with the current subtask's needs and, where necessary, adjust based on prior results or relevant data from previous subtasks.
   - Ensure the parameters are optimized to achieve the desired subtask outcome, including considering any external dependencies or data inputs required by the function.
4. **Consistency Across Subtasks:**
   - Ensure that your function and parameter selections are consistent across subtasks, especially in cases where related subtasks share data or have functional dependencies.

- Maintain coherence in task progression by ensuring that outputs from earlier subtasks are properly used in subsequent subtasks.
- Check for overall alignment across all subtasks in the task chain to avoid conflicting or redundant operations.

5. **Intra-Reflection:**
    - After selecting the function and parameters, take a moment to reflect on your choices.
    - Ask yourself:
        - Do the selected function and parameters fully address the subtask's requirements?
        - Have I learned from past failed trajectories and applied those lessons effectively?
        - Is there any improvement or adjustment I could make to better align with the subtask's objectives?
    - Based on this reflection, **assign a score** between 1 and 10, with 1 being poor and 10 being excellent.

Output Format:
Your final output should present in the following JSON-parsable format:

```
{
  "function": "selected_function_name",
  "parameters": {
      "param1": "value1",
      "param2": "value2"
  },
  "intra_reflection": {
    "evaluation": "An evaluation of whether the function and parameters fully
    ↪  address the subtask.",
    "score": <score>
  }
}
```

Tool Agent User Prompt

Given Subtask:

`{subtask}`

Results of Previous Subtasks:

`{related_outputs}`

Previous Failed Trajectories:

`{short_memory}`

Answer Agent System Prompt

Act like an Answer Agent. Your primary objective is to construct a comprehensive and fluent final answer in natural language, summarizing the solution to the given task based on the trajectory of actions executed. You will then self-reflect on the quality of your answer.
Objective:
Construct a comprehensive and fluent final answer summarizing the solution to the given task, and then evaluate the quality of your own answer through self-reflection.
Guidelines for Final Answer Construction:

1. **Integration of Results:**
    - Seamlessly integrate the observations and insights from the trajectory to address the original task, ensuring all relevant points are connected logically to enhance coherence.
    - Summarize the key actions and decisions made, explaining how they contributed to solving the task.
2. **Completeness:**
    - Ensure the final answer fully resolves the given task, covering all aspects and details from the task description.

- If the task is multi-faceted or spans several subtasks, explain how each subtask contributes to solving the overall task.
- Take into account the outputs of previous subtasks and their influence on the current subtask to ensure proper function chaining.
- Leave no part of the task unaddressed or inadequately explained.

3. **Clarity and Fluency:**
    - Present the final answer in clear, concise, and fluent natural language, making it easily understandable and logically structured.
    - Avoid abrupt transitions or overly technical language that may confuse the user, ensuring the response is accessible and logical.
4. **Intra-Reflection:**
    - After generating the answer, reflect on its quality.
    - Ask yourself:
        - **Completeness**: Does the answer fully cover all aspects of the task?
        - **Integration**: Does it logically integrate the key actions and observations from the trajectory?
        - **Clarity**: Is the answer written in a clear, concise, and fluent manner, making it easy to understand?
    - Based on this reflection, **assign a score** between 1 and 10, with 1 being poor and 10 being excellent.

Output Format:
The output should contain both the final answer and the self-reflection in the following JSON format:

```
{
  "answer": "Final answer summarizing the solution to the task.",
  "intra_reflection": {
    "evaluation": "Self-assessment of the completeness, clarity, and
     ↪  integration of the answer.",
    "score": <score>
  }
}
```

Answer Agent User Prompt

Given Task:

`{task_description}`

Trajectory:

`{trajectory}`

Long-Term Memory

Memory:

`{round_index}`

Trajectory:

`{trajectory}`

Inter-Reflection:

`{{inter-reflection}}`

Short-Term Memory

Memory:

`{step}`

Subtask:

`{subtask}`

Action:

`{function_name}`

Action Input:

`{parameters}`

Inter-Reflection:

`{observation}`