S-Agent: an Agent Collaborative Framework Inspired by the Scientific Methodology

Anonymous ACL submission

Abstract

An increasing number of advancements have been accomplished in agents empowered by Large Language Models (LLM), particularly in resolving simple dialogue tasks. However, existing agents still face intractable robustness issues for solving complex tasks, encountering the cascading hallucinations induced by multi-step invocations of LLM. Certain recent studies utilize multi-step reasoning, planning strategies, and domain workflows to improve the success rate of complex tasks, yet they neglect the scientific methodology that encompasses the accumulated wisdom derived from centuries of scientific inquiry. Drawing inspiration from the scientific methodology, we propose the S-Agent - an agent collaborative framework meticulously designed to actively experiment and refine theories based on the analysis of experimental results, thereby enhancing the deductive capabilities of LLMs and complementing their inductive and communicative strengths. Additionally, we introduce an innovative parallel planning methodology, wherein agents with identical roles collaborate to simultaneously address the same inquiry. Extensive experiments demonstrate the effectiveness and efficiency of our approach. Notably, we achieve a new state-of-the-art 33.3% pass@1 accuracy on the LeetcodeHardGym coding benchmark and a relatively good 96.3% pass@1 on HumanEval with GPT-4.

1 Introduction

Recently, significant advancements have been achieved in the realm of problem-solving through agents founded upon Large Language Models (LLM). Existing studies incorporated step by step reasoning & planning strategies (Yao et al., 2023b; Shinn et al., 2023; Yao et al., 2023a; Zhou et al., 2023), and used tools to extend agents' capabilities(Wu et al., 2023; Yang et al., 2023; Shen et al., 2023). These studies have demonstrated their capability to tackle uncomplicated dialogue tasks. However, current agents continue to confront insurmountable challenges in terms of resilience when it comes to resolving complex tasks, as they encounter the cascading hallucinations brought about by the multi-step invocations of LLM.

In order to improve the proficiency of agents in resolving complex tasks, some recent studies employ domain expertise to guide agents towards adhering to standard operating procedures (SOP) (Hong et al., 2023; Qian et al., 2023). These studies can only serve specialized applications as they rely on domain-specific procedural knowledge, still lack of principled guidance.

To address the aforementioned issues, we draw inspiration from the scientific method [1] that encompasses the wisdom accumulated through centuries of scientific exploration and has been validated across various disciplines. The magnificent modern science usually adheres to an iterative paradigm: deriving ideas from observations and constructing hypotheses. These hypotheses are then subjected to experimentation, with the outcomes serving as observations that either validate or question the hypotheses. At the heart of this paradigm lies the notion of "falsifiability," introduced by Karl Popper(Popper, 1959), which asserts that there must exist experimental findings capable of disproving the hypotheses. Guided by this paradigm, theory and experiment can mutually inform and enhance scientific understanding.

In this paper, we propose the S-Agent, a multi-agent framework where agents partake in dialogues and collaborations inspired by the scientific method. This framework encompasses crucial processes including idea generation, experiment conduction, and the discussion of results. Our experiments

[1] scientific method: A method of procedure that has characterized natural science since the 17th century, consisting in systematic observation, measurement, and experiment, and the formulation, testing, and modification of hypotheses: criticism is the backbone of the scientific method.

demonstrate that the system performs exceptionally well in tackling challenging tasks, such as coding, multi-hop QA .

To summarize, our key contributions are the following:

- To the best of our knowledge, we are pioneering the integration of formulating, experimenting, and modifying of hypotheses within the LLM agent system. These mechanisms represent the accumulated wisdom of centuries of scientific research and are poised to enhance the credibility and accuracy of LLM agents.
- We present S-Agent, a collaborative framework for agents that integrates an automated workflow planner and a parallel agent task management unit. The framework offers adaptable assistance for developing agents of complex and high-reliability tasks, enabling simultaneous operation at the agent level and significantly enhancing efficiency.
- We conduct extensive experiments to demonstrate the effectiveness and efficiency of our approach. Notably, we achieve a new state-of-the-art 33.3% pass@1accuracy on the leetcode-hard benchmark with GPT-4 and relative good result 96.3% pass@1 accuracy on the HumanEval coding benchmark with GPT-4.

2 Related work

2.1 Think Like Human

In the early phase of the Language Model (LLM) era, researchers began exploring LLMs with the goal of achieving universal question-answering capabilities, formulated as $answer = LLM(question)$(Devlin et al., 2018; Raffel et al., 2020). Subsequently, GPT-3(Brown et al., 2020), a pioneer in this domain, introduced the concept of few-shot learning. This method reflects the human ability to improve problem-solving skills through exposure to a limited number of examples, thereby shifting the paradigm to $answer = LLM(demos, question)$. Furthermore, the Chain of Thought (COT) approach(Wei et al., 2022) demonstrated that incorporating simple instructions such as 'please do it step by step' significantly enhances performance. Consequently, the paradigm evolved to $answer = LLM(instructions, demos, question)$.

By crafting diverse instructions, the Tree of Thoughts (ToT) framework(Yao et al., 2023a) enables LLMs to generate multiple plans and select the most appropriate one. The ReAct framework(Yao et al., 2023b), which integrates reasoning and action, modifies the paradigm to $result_{i+1} = LLM(instruction, demos, result_i, question)$. Reflexion(Shinn et al., 2023) advances this approach by supervising the entire decision-making process and offering feedback on the complete sequence of actions.

This paradigm closely mirrors the human process of problem-solving: step-by-step refinement of solutions based on past experiences and logical analysis. However, we observe that when faced with more complex issues, humans employ another cognitive approach not yet utilized in LLM-based agents: the scientific method, which includes conducting experiments. Scientists design experiments to test their theories, comparing experimental results with expectations to identify discrepancies before actual implementation. Current LLM agents lack this capability; they do not formulate expectations prior to taking actions.

2.2 Use Tool Like Human

While the Marxist philosophy posits that tool utilization and creation are key distinctions between humans and animals, it's the transformative impact of tools on human evolution and dominance on Earth that is truly noteworthy. This concept finds a parallel in the realm of language models like ChatGPT. Despite their impressive performance and global recognition, these models have inherent limitations, including constrained calculation abilities, restricted access to rare knowledge, and limited proficiency in handling other modalities. Mirroring the human approach to overcoming similar constraints, the strategic use of tools has emerged as an effective solution in augmenting these models.

Pioneering efforts such as Visual ChatGPT(Wu et al., 2023) and HuggingGPT(Shen et al., 2023) laid the groundwork by integrating multi-modal models as auxiliary tools, thereby expanding the functionalities of LLMs beyond single-modal capabilities. This trajectory was further propelled by MM-ReAct(Yang et al., 2023), which cleverly incorporated a search engine and Microsoft API services into the mix. This innovative approach has been widely adopted, with OpenAI's ChatGPT introducing plugin functions and Microsoft's New-Bing exemplifying an LLM integrated with Bing search capabilities.

In today's landscape, the ability to configure and customize tool sets has become a fundamental feature in most agent assistant applications. Integrating tools transforms the paradigm into $tool = LLM(instructions, toolset, question), answer = LLM(demos, tool, question)$. However, most previous work has primarily focused on the correctness of tool selection, without considering how to enable LLMs to use complex tools. We aim for our approach to empower LLMs to effectively utilize tools with which they are not yet familiar.

2.3 Collaborate like Human

Researchers have ventured beyond exploring human cognition and tool usage, delving into the intricacies of organizational dynamics. This exploration has been facilitated by employing multi-agent systems to create artificial entities that simulate the workings of a company. The innovative concept of role-playing was pioneered by Camel(Li et al., 2023), utilizing the paradigm $answer = LLM(..., personality, question)$.

This concept was expanded in subsequent studies that modeled their operations explicitly after a corporate structure. Notable examples include MetaGPT(Hong et al., 2023) and ChatDev(Qian et al., 2023), which emulate software companies handling programming tasks. These models assign roles like CEO, CTO, product manager, programmer, and designer, systematically reflecting the organizational structure of a real-world company. The adoption of a Standard Operating Procedure (SOP), crafted by professionals and fed into the system, guides the collaborative process, creating a workflow similar to that of a traditional software company. These works shift the paradigm to $output_i = LLM(personality_i, instruction_i, demos_i, input_i)$, where each index represents a different role. By following a sequence of roles, the answer is generated through this process.

Building on this foundation, AgentVerse(Chen et al., 2023b) and AutoAgents(Chen et al., 2023a) introduced a job market system, simulating the recruitment of expert agents for specific roles. This approach also generate the SOP. By automating everything, the artificial company of agents achieves a high level of task proficiency and autonomy.

While these models typically employ a sequential waterfall workflow, addressing each component of a task linearly, recent research(Zhang et al., 2023; Chen et al., 2023b; Du et al., 2023) has highlighted the benefits of cooperative approaches, such as debates, in enhancing performance. Our work introduces an innovative parallel planning methodology, where agents with identical roles collaborate to simultaneously address the same question. This parallel approach has been instrumental in boosting performance, demonstrating the efficacy of multi-agent cooperation in complex problem-solving.

3 Methodology

To empower the S-Agent system with the scientific method, we have structured the process into three distinct parts: idea generation, experiment conduction, and panel discussion. Each part is facilitated by purpose-built agents, as illustrated in figure 1.

3.1 Idea Generation

The Idea generation part serves as the initial stage for proposing ideas and formulating hypotheses during the execution process of the agent system. Throughout the experiment iterations, ideas undergo refinement and new ones emerge. In this phase, we specifically design LLM-powered agents, termed **idea generation agents**, to emulate the role of scientists in generating structured ideas easily verifiable by subsequent experiments. Meanwhile, these agents can analyze feedback from discussion parts and revise their ideas accordingly. In programming scenarios, these agents initially receive a coding question as input and directly generate Python solutions. In the next few rounds, the agent shall take the discussion feedback as input and generate new solutions if the solutions generated by the previous ideas do not pass the targeted experiments. As shown in Figure 1, the idea generation agents receive the input and produce ideas. A detailed task adaption will be presented in the experiments section.

3.2 Experiments Conduction

The experiment conduction phase consists of designing appropriate experiments and carrying out the designed experiments. These steps are often considered the most crucial in the scientific method. A well-designed experiment can determine not only the validity of a hypothesis but also highlight its advantages over previous theories when the results support the hypothesis. Conversely, when the results reject the hypothesis, the experiment can pinpoint the specific issues that remain unresolved. This focused feedback can make the iteration of

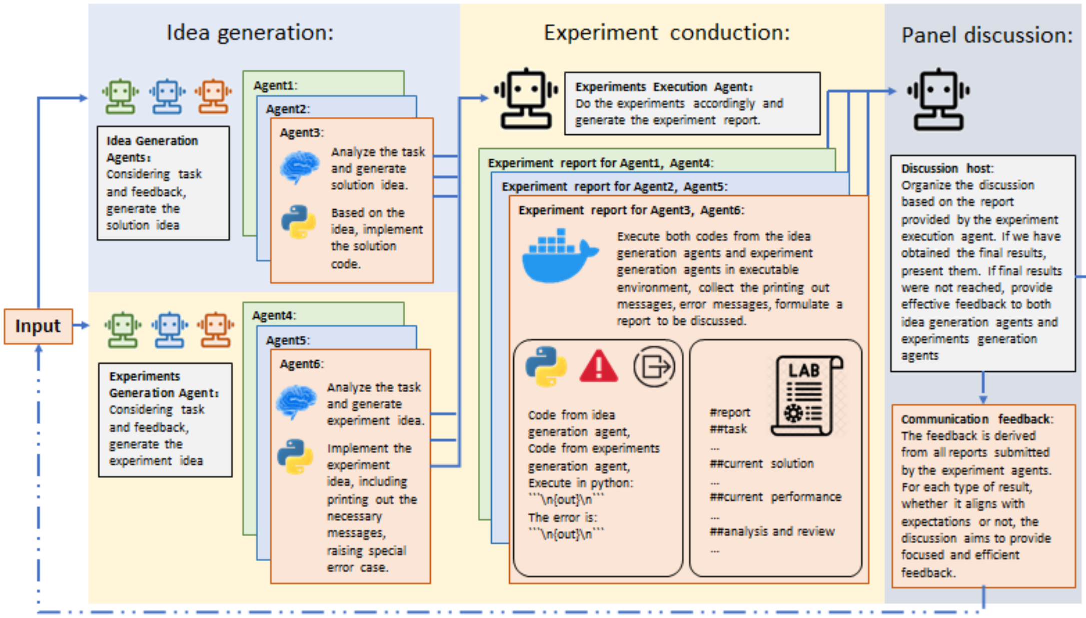

Figure 1: Illustration of S-Agent. The idea generation phase showcases agents responsible for generating ideas, encompassing both solution concepts and corresponding code. The experiment conduction phase begins with the generation of experiment ideas by experiment generation agents, followed by the execution of experiments through experiment execution agents. Experiment reports contained thorough analysis of results and the collection of feedback from the environment are also generated in this phase. In the panel discussion session, a discussion host synthesizes experiment reports from preceding agents, facilitating the generation of valuable feedback. This feedback is systematically organized to serve as input for the subsequent iteration, fostering continuous idea and experiment setting improvement.

hypotheses much faster, also help avoid deduction analysis from wrong start. In our system, applying this hypotheses experiment alignment phase properly can avoid machine hallucination effectively. To equip this mechanism in our system, we designed **experiment generation agents** to do experiments design and **experiment execution agents** to test out hypotheses and produce detailed experiment reports after receiving the results. As illustrated in Figure 1, the experiment generation agents formulated the targeted experiment plan based on proposed idea and execution reports, and the experiment execution agents execute codes with these test cases in execution environment.

3.3 Panel Discussion

Observing the workflow of a scientist reveals that panel discussion, often through paper publications and sharing, is a vital component of the modern academic community. Therefore, when an experiment validates an idea, we facilitate discussion among different agents. This process allows them to review others' work, ensuring internal logic consistency and providing references and inspiration to other agents. The discussion results in either feedback on analyzing experiment results or final answers if the designed experiments successfully verify the proposed ideas. To manage this process effectively, we have designed a specialized agent called a **discussion host**, responsible for aggregating information, assessing previous results, and overseeing the overall status of ongoing discussions.

3.4 Supporting Component

To simplify the deployment of the entire system and reduce execution time, we also include two supporting components named the planner and agent task management.

Automated Workflow Planner The automated workflow planner is a specially designed agent responsible for generating sequences of execution flows for agents and managing their interdependencies. This critical component strategically plans and organizes the order in which agents operate, ensuring a coherent and efficient workflow. To formulate this plan, only the input-output information and the agents' goals are required. As shown in Figure 1, the edges of the directed workflow graph are generated by the planner automatically. This approach draws inspiration from the methodology introduced in LLMCompiler(Kim et al., 2023). The detailed prompt design and sample illustration are given in the appendix.

Agent Task Management Unit The agent task management unit, drawing inspiration from the instruction fetching mechanism in modern computer architecture, plays a crucial role in determining the optimal execution flow of agents based on the intermediate representation generated by the Planner. Employing a greedy policy, this unit swiftly adds agents to the task list as soon as they become ready for parallel calling. The implementation of this agent task management unit involves a straightforward fetching and queue mechanism, foregoing the need for a dedicated agent system.

4 Experiment

We assess the effectiveness of our framework in tackling complex problems using the coding benchmark LeetcodeHardGym(Shinn et al., 2023), alongside evaluations through HumanEval (Chen et al., 2021) and EvalPlus(Liu et al., 2023). Furthermore, we conduct a focused analysis to evaluate the efficacy of our approach in reducing hallucinations on the multi-hop QA benchmark HotpotQA dataset (Yang et al., 2018).

4.1 Coding

HumanEval(Chen et al., 2021)(distributed under the MIT license) is a benchmark for code synthesis, which consists of 164 programming problems with several test cases each. The problems in this dataset are designed to test the ability of LLMs to generate functionally correct codes, which means the generated codes can not only execute successfully but also pass the provided test cases, instead of being linguistically similar to the canonical solution. EvalPlus is a benchmark that aims to improve the quality and quantity of test cases for the existing HumanEval benchmark. EvalPlus contains new test cases that can catch more errors and bugs in the LLM-synthesize code. The LeetCodeHardGym(Shinn et al., 2023) consists of 40 open LeetCode hard problems. It was introduced in Reflexion, where the benchmark utilizes LeetCode's API and the traditional RL package gym (Brockman et al., 2016) to construct this dataset in the humaneval format, requiring no additional configuration modifications.

Our performance on these benchmarks is noteworthy, achieving a Pass@1 (Pass@1 is the probability that a model generates at least one correct solution out of one attempt) of 33.3% on LeetCodeHardGym, establishing a new SOTA. Additionally, we achieve strong performance with 96.3% on HumanEval and 86.6% on EvalPlus,

4.1.1 Implementation Details

As discussed earlier, the scientific method involves three phases: idea generation, experiment conduction, and panel discussion. In this section, we detail our approach to applying this methodology in the adaptation of the HumanEval dataset. We elaborate on our process for prompt design and provide a detailed example in the appendix for clarity.

In the idea generation phase, we task the idea generation agents with the dual roles of analysis and coding. Initially, these agents analyze and formulate a comprehensive strategy to tackle the current task. Subsequently, during the coding phase, they annotate each step of their proposed solution with explanatory comments. This process can be represented as $analysis, modified\_code = LLM(task, previous\_code, feedback)$. This approach mirrors the scientific method where $analysis, modified\_theory = Scientist(phenomenon, previous\_theory, experiment\_result)$ is analogous. The general structure of the prompt format is detailed in Figure 2, which also specifies the output format.

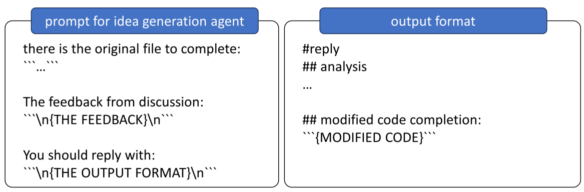

Figure 2: Prompt for idea generation agents and the output template. FEEDBACK is all the previous experiments reports.

During the experiment conduction stage, specialized agents are designated for experiment generation and execution. Experiment generation agents are tasked with creating specific experiments (in the context of coding, experiments refer to test cases), determining the experimental inputs, defining expected outputs, specifying messages for various types of output during analysis, and providing implementation code formatted as 'assert f(input)==output, description of the experiment'. Detailed instructions for this process are illustrated in Figure 4, where $test\_script = LLM(task, current\_code)$. This parallels the scientific method, where $experiment\_plan = experiment\_scientist(phenomenon, current\_theory)$.

The experiment execution agent is tasked with running Python code generated from idea generation within a functional environment and generating the experiment report. The output format of this agent is outlined in Figure 5, with detailed specifications of the report format provided in Figure 10 in the appendix. The agent interprets standard output and error messages to provide precise feedback, facilitating the refinement of theories or experiment designs.

Following the generation of experiment reports during the experiment conduction phase, the discussion host assumes responsibility for aggregating all reports and assessing the resolution of the issue. If the issue persists, each idea generator conducts a thorough analysis of their respective reports to refine their proposed ideas embedded within the generated code. Moreover, each participant involved in experiment generation reviews the reports to make adjustments to their experiments. This process exemplifies the effective panel discussion mechanism established by our framework.

In each test case, we initiate by feeding input to our system. Subsequently, each interaction with received feedback is considered a round. We set a maximum of 10 rounds per test case; exceeding this limit leads to the system being deemed unsuccessful. Conversely, upon achieving success, we derive the final solution from the resulting code. Upon completing all 164 cases, we run the official script to compute a Pass@1 score. We chose the number 10 to balance the system's performance, as increasing this limit might lead to chance improvements without substantial benefit. Comparable frameworks, such as Reflexion(Shinn et al., 2023), use a parameter of 6 for similar reasons. Additionally, our analysis shows that among the 158 successful cases, only 4 required more than 2 rounds to solve. Hence, reducing this number would not notably affect overall performance.

4.1.2 Result and Analysis

We evaluated S-Agent system using both GPT-4 and GPT-3.5. By utilizing GPT-4, our system successfully completed 158 out of 164 tasks (pass@1 = 96.3%), achieving state-of-the-art performance. Furthermore, our system leveraged GPT-3.5 to pass 137 out of 164 tasks (pass@1 = 84.1%), surpassing the performance of all our known works based on the same LLM. The results are summarized in Table 1. We have also attained a SOTA result on the LeetCodeHard benchmark, achieving a Pass@1 accuracy of 33%. Previously, the highest accuracy was 15%, achieved by Reflexion(Shinn et al., 2023).

Recent methodologies in coding tasks aim to enhance performance through several strategies: leveraging the Python execution environment, incorporating reflection mechanisms, and facilitating multi-agent discourse. The Python execution environment allows systems to execute code and verify its validity. Reflection enhances system performance by using verbal reinforcement cues generated by LLMs based on past experiences. Different systems implement reflection in varied ways: MetaGPT(Hong et al., 2023) employs a specialized agent for reflection, language-agent-tree-search (LATS)(Zhou et al., 2023) utilizes reflection in a trajectory format, and Reflexion(Shinn et al., 2023) focuses on enabling the LLM itself to review and learn from feedback. In coding tasks, feedback includes outputs such as standard output (stdout) and error messages generated during code execution.Multi-agent discourse, studied extensively in this context, involves aggregating ideas from different personas to generate more comprehensive and accurate answers(Du et al., 2023; Chen et al., 2023b).

Prior methodologies typically receive feedback passively from the Python Interpreter. In contrast, our approach uniquely empowers the agent to actively influence the feedback process, as detailed in Figure 3. In our framework, outlined in the implementation section, the experiment generation agents configure experiments to produce intermediate results. This approach introduces additional lines such as printed statements that provide detailed insights into any encountered errors. These lines not only utilize feedback but also enrich it through agent-driven efforts, thereby enhancing the feedback with more valuable information. This enrichment guides the system towards effectively addressing complex problems. The enhanced feedback from experimental results and inter-agent panel discussions has propelled us to achieve state-of-the-art performance.

	GPT4	ANPL	MetaGPT	AgentVerse	Reflextion	LATS	S-Agent
code execution	×	✓	✓	✓	✓	✓	✓
reflection	×	×	✓	✓	✓	✓	✓
panel discussion	×	×	×	✓	×	×	✓
experiment design	×	×	×	×	×	×	✓
Pass@1 (GPT3.5-base)	-	76.2%	-	75%	-	83.8%	**84%**
(GPT4-base)	67%	86.6%	87.7%	89%	91%	94.4%	**96.3%**

Table 1: Pass@1 result of related works on programming. We refer code execution as the use of python execution environment, reflection as the use of LLM-generated feedback, panel discussion as agents sharing and discuss each others work, experiment design as actively design experiment according to proposed idea and previous feedback. Our S-Agent system stands out from previous works and achieved best pass@1 score, mainly because of involving actively designing experiments.

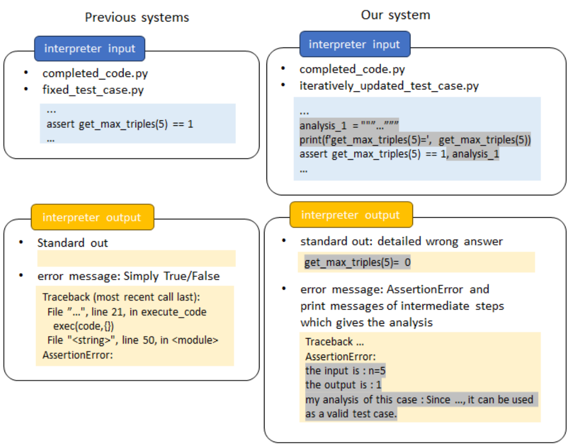

Figure 3: This figure demonstrates the differences in input and output between our system and the previous systems. Because of incorporating specially designed experiments during the python execution step, instead of just providing a binary assessment of code functionality, our system also generate intermediate results and other results based on the experiment requirement, as highlighted in the image.

S-Agent Variants	HumanEval	EvalPlus
Original	96.3%	89.0%
w/o Panel	92.1%	86.6%
w/o Exp&Panel	84.8%	79.9%
w/o Idea&Exp&Panel	68.3%	65.2%

Table 2: Pass@1 on HumanEval and EvalPlus using GPT-4 under different S-Agent Variants. The experiment employs a single agent for idea generation, another for experiment conduction, and a third for panel discussion.

4.1.3 Ablation Study

We conduct experiments on the HumanEval and EvalPlus datasets to investigate the effectiveness of mechanisms within the S-Agent framework.

Table 2 shows the results of the experimental results of three S-Agent variants including the original framework (Original), S-Agent without Panel discussion mechanism (w/o Panel), S-Agent without Panel discussion and Experiment conduction mechanism (w/o Exp&Panel), S-Agent without the Idea generation, Experiment conduction, and Panel discussion mechanisms (w/o Idea&Exp&Panel). The findings indicate that all three fundamental mechanisms play a beneficial role in enhancing the precision of the S-Agent framework.

We also investigate how the number of Idea generation and Experiment conduction agents impact the results of the S-Agent framework. As depicted in Table 3, with an increase in the number of agents, there is a clear improvement in the average pass@1 value, accompanied by a reduction in the standard deviation. This indicates that, solely from a performance perspective, increased discussion can substantially enhance both the accuracy and stability of the entire system. This conclusion also aligned with (Du et al., 2023; Yang et al., 2018). This also aligned with our result in GPT-4 based experiments, the pass@1 increase from 92.1% to 96.3% when increasing the number of agents to 3.

Number of Agents	HumanEval Pass@1 (%)	EvalPlus Pass@1 (%)
1	80.5 ± 1.8	72.0 ± 1.8
3	83.5 ± 1.37	74.4 ± 1.37
5	84.1 ± 0.56	75.6 ± 0.56

Table 3: Pass@1 performance on HumanEval and EvalPlus using GPT-3.5 under different numbers of Idea generation and Experiment conduction agents.

4.2 HotpotQA

The HotpotQA(Yang et al., 2018) dataset, is a crucial benchmark in Natural Language Processing (NLP) and Question Answering (QA). It is specifically designed for models that handle complex, multi-hop question-answering tasks, requiring synthesis of information across multiple text sources from wiki. The HotpotQA dataset is distributed under the CC BY-SA 4.0 license. The code is distributed under the Apache 2.0 license.

We randomly sampled a subset of 50 QA pairs from the original dataset and conducted tests without panel discussion settings. Compared to ReAct(Yao et al., 2023b), our approach improved results by 7.4%. The S-Agent outperforms ReAct(Yao et al., 2023b) by providing more precise and effective feedback.

In our experiment demonstrating how LLM's approach can reduce hallucinations, we examined a query asking "Which show was hosted by Jessica Drake's former spouse?". Both our approach and previous methods use the current result to generate subsequent queries based on Wikipedia. The key difference lies in how queries are generated: previous methods employ $next\_query = LLM(task, current\_knowledge, review\_of\_lastquery)$, whereas our approach utilizes $next\_query = LLM(task, current\_knowledge, expaxted\_of\_last\_query, last\_query\_result)$

In this scenario, both systems first query information about Jessica Drake's ex-husbands, identifying Brad Armstrong and Evan Stone. The React system, lacking explicit expectation management, falls into a loop when querying Brad Armstrong. It assumes the direction is correct solely based on his association with the movie industry, perpetuating the query without finding the answer. In contrast, our framework treats each query as an experiment with predefined expectations. By comparing the actual result with these expectations, LLM identifies incorrect paths and intervenes with feedback such as "The provided text does not mention any show hosted by Brad Armstrong". This feedback redirects attention to Evan Stone, leading to successful resolution of the task.

The precise and efficient feedback mechanism in our framework plays a crucial role in preventing LLM from persisting in incorrect directions, thereby mitigating the risk of endlessly generating inaccurate information. For detailed prompts, please refer to the appendix.

5 Conclusion

In this paper, we introduce the S-Agent, an innovative multi-agent framework in which agents engage in dialogues and collaborations inspired by the scientific methodology. The framework incorporates the essential processes of hypothesis development, experimentation, and refinement. These processes embody the collective knowledge accumulated over centuries of scientific inquiry and are poised to enhance the credibility and precision of LLM agents. The S-Agent integrates an automated workflow planner and a parallel agent task management unit, providing flexible support for developing agents for complex and high-stakes tasks, facilitating concurrent operation at the agent level. Extensive experiments confirm the efficacy and efficiency of our methodology. Notably, the S-Agent achieves a new state-of-the-art 96.3% pass@1 accuracy on the HumanEval coding benchmark with GPT-4.

6 Limitations and Future Work

The primary objective of this paper is to elucidate the concept of scientific methodology. While our general framework may not exhibit the same level of sophistication as pioneering works like AutoAgents(Chen et al., 2023a) which autonomously generate requisite agents, our framework still requires the manual implementation of specific agents tailored for specialized tasks. In future developments, a key direction of research involves exploring methods to automatically generate agents with finely crafted prompts, presenting an important avenue for further exploration.

Currently, following the generation of the Directed Acyclic Graph (DAG) plan, the plan remains static. However, it is imperative to establish dynamic refinement for this plan. Recent advancements, exemplified by works such as ReAct(Yao et al., 2023b), BabyAGI, and XAgent, have endeavored to enhance plans based on feedback received at each step. While these approaches typically involve linearly designed steps, there is a research gap in developing methods to dynamically refine a DAG-formatted plan with the capability for parallel execution.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Sesay Jaward, Karlsson Börje, Jie Fu, and Yemin Shi. 2023a. Autoagents: The automatic agents generation framework. *arXiv preprint*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. Metagpt: Meta programming for a multi-agent collaborative framework.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An llm compiler for parallel function calling. *arXiv*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Karl Popper. 1959. *The Logic of Scientific Discovery*, 2002 pbk; 2005 ebook edition. Routledge.

Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. In *Advances in Neural Information Processing Systems*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action.

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. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of Thoughts: Deliberate problem solving with large language models.

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

Jintian Zhang, Xin Xu, and Shumin Deng. 2023. Exploring collaboration mechanisms for llm agents: A social psychology view.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models.

A Additional Details

A.1 Automated Workflow Planner

Given a user query, the planner will create a plan to solve the user query with the most parallelizability. The prompt of the planner agent contains the profiles of every agent and the tools it equips. Each task in the plan must have a unique ID, which is strictly increasing. Inputs for each task can either be constants or outputs from preceding actions. If the outputs from preceding actions are needed, we use the format $id to denote the ID of the previous agent task whose output will be replaced with the input. Upon the completion of all agent tasks, we always call join as the last task in the plan to collect all the previous task outputs and formulate a final output. We use the HumanEval coding question $get\_max\_triplets$ to demonstrate the functionality of our LLM planner agents. Figure 6 shows the plan that the planner creates after receiving the coding question as the user query. In this case, user queries are fed into three Coders and the Tester. Afterward, the output from the Coders should be considered as the input to the Expereimnters with the output from the Tester respectively. Then the gathered outputs from these Experiments should be the input of the Discussion Hoster. In the end, we collect all the results and finish this plan.

A.2 Agent Task Management Unit

Figure 7 shows how the management unit handles the agent tasks. Agents are equipped with the tools that the user provides and tasks are delegated to the associate tool. The management unit synchronously listens to the task queue and schedules tasks as they arrive in the queue based on their dependencies. More specially, in this case, the three coders and the Tester agents execute in parallel at the same time due to empty dependencies. While the Experiment agents cannot execute in parallel until the completion of all Coder agents and Tester Agents. Meanwhile, we shall replace dependency placeholders, i.e. $i, in the args of the agent task with the actual output.

B Additional Results

B.1 Case study for HumanEval

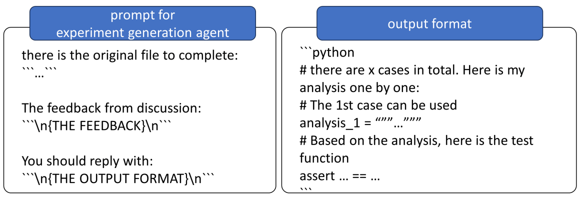

Figure 4: Prompt for experiments generation agent and the output template. FEEDBACK is all the previous experiments reports.

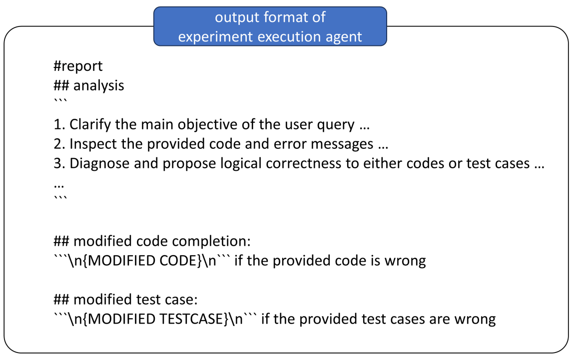

Figure 5: Output format of the experiment execution agent, where detailed analysis and modification of either codes or test cases are produced.

To get a closer look at a specific task, we chose No.147 as a demo because this task is only solved by our system, and it cost 5 rounds of modification to get the final answer. For the task and idea generation agent, refer figure 8, for the experiment generation agent refer figure 9, for the experiment execution agent and the exact augment feedback from python, refer figure 10

Planner

1. Coder_1(def get_max_triples(n): """ You are given a positive integer n. You have to create an integer array a of length n. For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1. Return the number of triples (a[i], a[j], a[k]) of a where i < j < k, and a[i] + a[j] + a[k] is a multiple of 3. Example : Input: n = 5 Output: 1 Explanation: a = [1, 3, 7, 13, 21] The only valid triple is (1, 7, 13). """)
2. Coder_2(def get_max_triples(n): """ You are given a positive integer n. You have to create an integer array a of length n. For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1. Return the number of triples (a[i], a[j], a[k]) of a where i < j < k, and a[i] + a[j] + a[k] is a multiple of 3. Example : Input: n = 5 Output: 1 Explanation: a = [1, 3, 7, 13, 21] The only valid triple is (1, 7, 13). """)
3. Coder_3(def get_max_triples(n): """ You are given a positive integer n. You have to create an integer array a of length n. For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1. Return the number of triples (a[i], a[j], a[k]) of a where i < j < k, and a[i] + a[j] + a[k] is a multiple of 3. Example : Input: n = 5 Output: 1 Explanation: a = [1, 3, 7, 13, 21] The only valid triple is (1, 7, 13). """)
4. Tester(def get_max_triples(n): """ You are given a positive integer n. You have to create an integer array a of length n. For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1. Return the number of triples (a[i], a[j], a[k]) of a where i < j < k, and a[i] + a[j] + a[k] is a multiple of 3. Example : Input: n = 5 Output: 1 Explanation: a = [1, 3, 7, 13, 21] The only valid triple is (1, 7, 13). """)
5. Experimenter($1+++$4)
6. Experimenter($2+++$4)
7. Experimenter($3+++$4)
8. DiscussionHoster($5+++$6+++$7)
9. join()

Figure 6: Given a coding task: "def $get\_max\_triplets$: You are given a positive integer n. You have to create an integer array a of length n. For each i ($1 \leq i \leq n$), the value of $a[i] = i \times i - i + 1$. Return the number of triples $(a[i], a[j], a[k])$ of a where $i < j < k$, and $a[i] + a[j] + a[k]$ is a multiple of 3. Example: Input: $n = 5$, Output: 1, Explanation: $a = [1, 3, 7, 13, 21]$, The only valid triple is $(1, 7, 13)$", the planner agent shall automatically make the detailed plan of the agents' workflow and their dependencies.

Based on such augmented feedback, these agents can get more specific reviews about current code , just like scientists can draw more specific conclusions based on specific experiment results. Here they find out that the problem didn't handle reminder 1 and 2 properly .

Based on this feedback, they will repeat the work procedure. By observing the correct final answer, we would find that the issue at the first draft is that it did not consider three 1s or three 2s can also lead to reminding 0, just like three 0s.

B.2 HotpotQA

B.2.1 dataset

The HotpotQA(Yang et al., 2018) dataset stands as a pivotal benchmark within the realm of Natural Language Processing (NLP) and Question Answering (QA). This dataset serves a distinctive purpose, tailored for assessing models tasked with intricate, multi-hop question-answering assignments that necessitate the synthesis of information from various textual sources. Consequently, the inclusion of HotpotQA in evaluations requires models to showcase sophisticated reasoning and comprehension abilities.

HotpotQA offers two evaluation settings, Fullwiki and Distractor. In the Fullwiki Setting, the dataset provides only questions, and users must retrieve related information from the entire Wikipedia dataset using their Information Retrieval (IR) system. The effectiveness of the search strategy in this setting is crucial, as the content found can significantly influence the results. Users then use their models to answer the question based on this information. In the Distractor Setting, the dataset provides questions along with context from the Wikipedia dataset, which includes both related and unrelated paragraphs. In this case, the user's model must be able to sift through the shuffled context to find the relevant information and answer the question correctly. In both settings, models are tasked with predicting the answer and identifying the supporting paragraphs in the context. When evaluating performance, we use Exact Match (EM), which measures whether the model's answer precisely matches the ground truth answer.

B.2.2 implementation

In this particular dataset, we empower idea generation agents to create the entire search plan. This includes formulating a set of queries for interaction with Wikipedia and specifying the desired

```
Agent Task Management Unit

1. Task(idx=1, name='Coder_1', tool=<function writecode_1 at 0x11c11b370>, args=('\ndef get_max_triples(n):\n    """\n    You are given a positive integer n. You have to ..."""\n',), dependencies=[], stringify_rule=<function <lambda> at 0x11c11b2e0>, thought='', observation=None, is_join=False)
2. Task(idx=2, name='Coder_2', tool=<function writecode_2 at 0x11c11b520>, args=('\ndef get_max_triples(n):\n    """\n    You are given a positive integer n. You have to ..."""\n',), dependencies=[], stringify_rule=<function <lambda> at 0x11c11b6d0>, thought='', observation=None, is_join=False)
3. Task(idx=3, name='Coder_3', tool=<function writecode_3 at 0x11c11b760>, args=('\ndef get_max_triples(n):\n    """\n    You are given a positive integer n. You have to ..."""\n',), dependencies=[], stringify_rule=<function <lambda> at 0x11c11b9a0>, thought='', observation=None, is_join=False)
4. Task(idx=4, name='Tester', tool=<function create_unit_tests at 0x11c11bd00>, args=('\ndef get_max_triples(n):\n    """\n    You are given a positive integer n. You have to ..."""\n',), dependencies=[], stringify_rule=<function <lambda> at 0x11c13c0d0>, thought='', observation=None, is_join=False)
5. Task(idx=5, name='Experimenter', tool=<function experiment at 0x11c13fe20>, args=('$1+++$4',), dependencies=[1, 4], stringify_rule=<function <lambda> at 0x11c13ff40>, thought='', observation=None, is_join=False)
6. Task(idx=6, name='Experimenter', tool=<function experiment at 0x11c13fe20>, args=('$2+++$4',), dependencies=[2, 4], stringify_rule=<function <lambda> at 0x11c13ff40>, thought='', observation=None, is_join=False)
7. Task(idx=7, name='Experimenter', tool=<function experiment at 0x11c13fe20>, args=('$3+++$4',), dependencies=[3, 4], stringify_rule=<function <lambda> at 0x11c13ff40>, thought='', observation=None, is_join=False)
8. Task(idx=8, name='DiscussionHoster', tool=<function discuss at 0x11c13c1f0>, args=('$5+++$6+++$7',), dependencies=[5, 6, 7], stringify_rule=<function <lambda> at 0x11c13c4c0>, thought='', observation=None, is_join=False)
9. Task(idx=9, name='join', tool=<function instantiate_task.<locals>.<lambda> at 0x11c35add0>, args=('<END_OF_PLAN>',), dependencies=[1, 2, 3, 4, 5, 6, 7, 8], stringify_rule=None, thought=('',), observation=None, is_join=True)
```

Figure 7: Here is an example of how the agent task management unit works for the coding task: def $get\_max\_triplets$, where tasks with no dependencies execute parallelly.

knowledge to be retrieved through these queries. Simultaneously generating pairs of queries and targeted knowledge streamlines the experimental process, especially when assessing the efficiency of the queries. This approach aligns with the scientific method, emphasizing the importance of providing falsifiable ideas.

In this context, the experiment is to evaluate whether the query can retrieve the targeted knowledge, therefore, a rule-based experiment generation agent is employed.

The experiment execution agent in this scenario is more intricate due to the complexity of communicating with the Wikipedia API. During the implementation of this specific experiment execution agent,we let llm automatically choose from several actions. If a Wikipedia page is successfully retrieved by a query, the agent systematically analyzes the information, starting from the first paragraph and summary information box, followed by the table of contents (TOC) of the Wikipedia page.If the answer can be directly generated from this initial information, it is returned promptly. If no answer is found, but certain paragraphs potentially contain details relevant to answering the question, the agent continues to read these paragraphs. The answer is returned if discovered in this process; otherwise, a summary and analysis detailing the reasons for the failure to find an answer are provided.

In situations where a Wikipedia page cannot be located with the given query, the agent generates a summary of similar pages returned by the API.

input to idea generation agent

there is the original file to complete:
```python
def get_max_triples(n):
  """You are given a positive integer n. You have to create an integer array a of length n. For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1. Return the number of triples (a[i], a[j], a[k]) of a where i < j < k, and a[i] + a[j] + a[k] is a multiple of 3.
Example :
    Input: n = 5
    Output: 1
    Explanation:a = [1, 3, 7, 13, 21]The only valid triple is (1, 7, 13).   """
```
The feedback from discussion:
```
{THE FEEDBACK}
```
You should reply with:
```
{THE OUTPUT FORMAT}
```

output from idea generation agent

reply
analysis
The implementation of the `get_max_triples` function appears to be conceptually incorrect, as it only checks for a triple made up of one element from each of the three possible residue classes modulo 3. This strategy does not cover all possible valid triples. The example requires finding all triples `(a[i], a[j], a[k])` where `i < j < k` and `a[i] + a[j] + a[k]` is a multiple of 3. Thus, the correct logic needs to account not only for one from each category but also for other combinations such as three zeros modulo 3. The current function fails to do this, which is likely why the test assertion fails.
modified code completion
```python
def get_max_triples(n):
  # Generate the array where a[i] = i * i - i + 1
  a = [i * i - i + 1 for i in range(1, n + 1)]
  # Count the occurrences of each number mod 3 in the array
  counts = [0, 0, 0]
  for number in a:
    counts[number % 3] += 1
  # Calculate the number of triples
  # We can form triples in the following ways:
  # 1. Three numbers whose remainders mod 3 are all zero (counts[0] choose 3)
  # 2. One number from each remainder category (counts[0] * counts[1] * counts[2])
  triples = (counts[0] * (counts[0] - 1) * (counts[0] - 2) // 6) + (counts[1] * counts[2] *
counts[0])
   return triples
```

Figure 8: The original task is "You are given a positive integer n. You have to create an integer array a of length n. For each i $(1 \leq i \leq n)$, the value of $a[i] = i \times i - i + 1$. Return the number of triples $(a[i], a[j], a[k])$ of a where $i < j < k$, and $a[i] + a[j] + a[k]$ is a multiple of 3. Example: Input: $n = 5$, Output: 1, Explanation: $a = [1, 3, 7, 13, 21]$, The only valid triple is $(1, 7, 13)$."
At first, one of the idea agents gives a draft saying that for all numbers in a_i , there is only 3 possible reminders when divided by 3 : 0,1,and 2. so it suggest first counts how many numbers reminds 0 ,1,and 2 . than there should be 2 senarieo that the 3 selected 3 numbers from the list satisfy the requirement : all the reminder is 0 or reminder is 0,1,2 .these two can be calculated by the 3 counts .

input to experiment generation agent

there is the original file to complete:
```python
def get_max_triples(n):
  """You are given a positive integer n. You have to create an integer array a of length n. For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1. Return the number of triples (a[i], a[j], a[k]) of a where i < j < k,   and a[i] + a[j] + a[k] is a multiple of 3.   Example :
    Input: n = 5
    Output: 1
    Explanation:a = [1, 3, 7, 13, 21]The only valid triple is (1, 7, 13).   """
```
The feedback from discussion:
```\n{THE FEEDBACK}\n```
You should reply with:
```\n{THE OUTPUT FORMAT}\n```

output from experiment generation agent

```python
# there are 1 cases in total Here is my analysis one by one:
# The 1st case can be used
analysis_1 = """the input is : n=5the output is : 1my analysis of this case : Since the example provided gives a direct output after calculation for n=5,and follows the definition of the function properly as per the problem statement, it can be used as a valid test case."""
# Based on the analysis, here is the test function (only contain the testcase that can be used):
print(f'get_max_triples(5)=', get_max_triples(5))
assert get_max_triples(5) == 1, analysis_1
print('all passed')
```

Figure 9: then , one of the experiments agents , design its experiment about the task , first print the result if n=5 , then if the code draft showed that the result is not 5 , raise an error message contains the analysis about why when n =5 it should be the result .

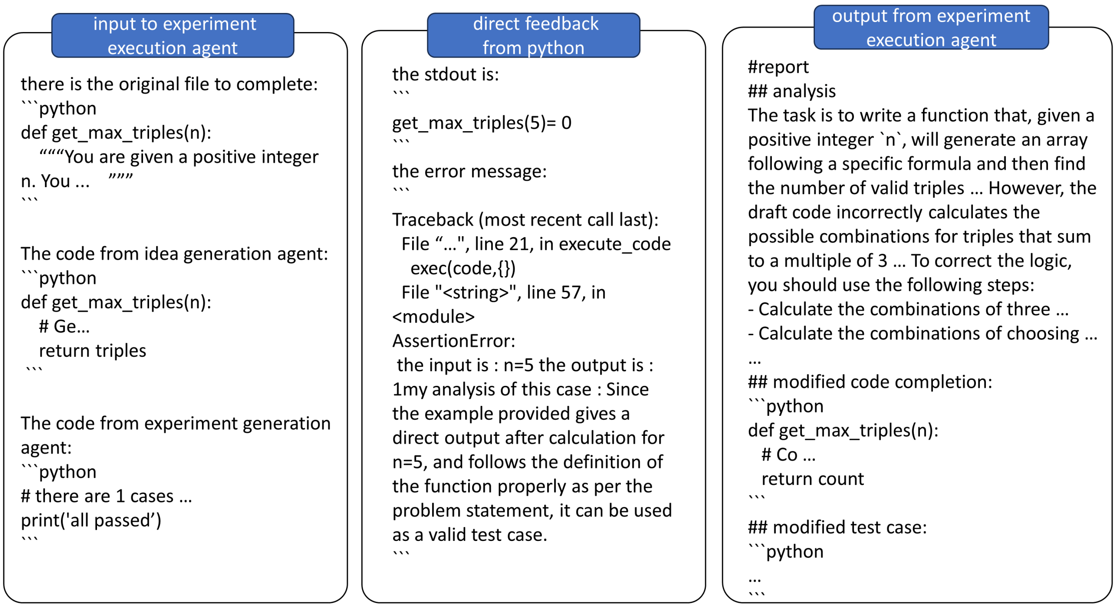

Figure 10: then the experiment excuted in a python interpreter , and give stdout , error message , the high light part of the error message is added by the experiment agent . this part is augmented feedback of current code draft .

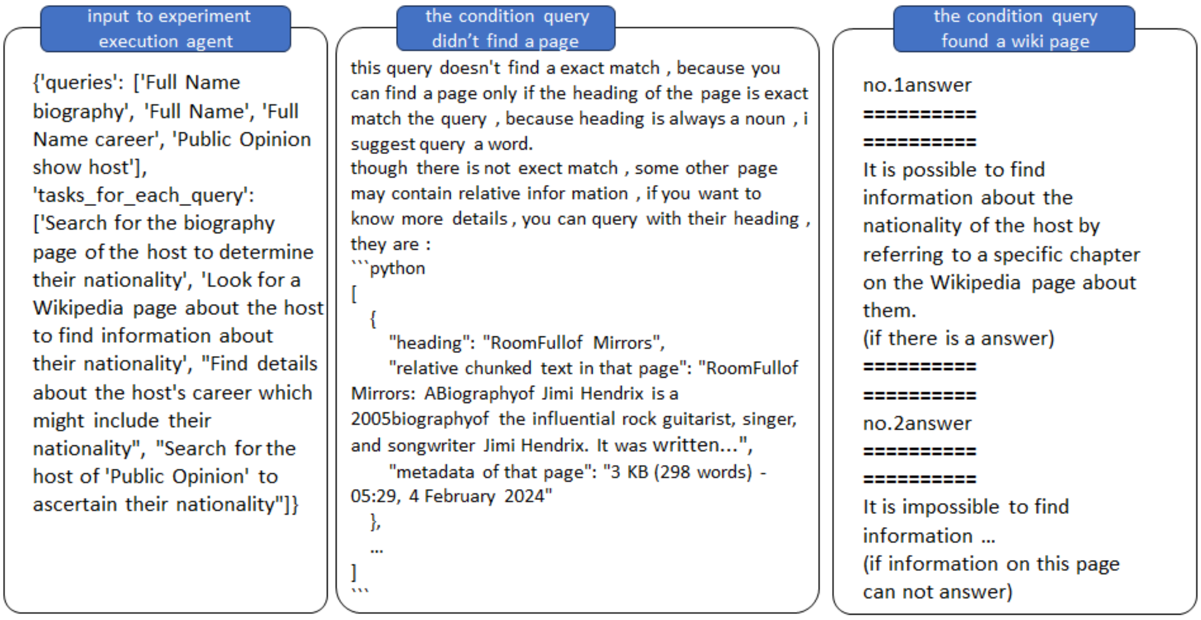

Figure 11: Input output example for experiment execution agents in HotpotQA

input to idea generation agent

you need to help me check wikipedia, to collect information to answer:
"{question}"
I already checked wikipedia with {n} queries, which you can refer:
{experiences}
is the current information enough to get answer? I will check wikipidia if you still have something not sure, please give me a list questiones you need me check, and the list of queries which could help checking these from wikipidia, attention, wiki page's title is often a noun, only exact match can successfully get the page.
if you can already answering the question based on current information, please tell me your final answer.
Reply with the format:{format}

output from idea generation agent

```
{
  'queries': [
    'Forestville Commonwealth',
    'History of Forestville Commonwealth',
     ...
    ],
  'tasks_for_each_query': [
    'Determine what Forestville Commonwealth is and its connection to socialist philosophy.',
    'Explore the historical background and establishment timeline of Forestville Commonwealth.', 'Investigate possible connections between the founder(s) of Forestville Commonwealth and socialist theories or movements.'
    ...
    ]
}
```

Figure 12: Input output example for idea generation agents in HotpotQA