AgentGit: A Version Control Framework for Reliable and Scalable LLM-Powered Multi-Agent Systems

Anonymous submission

Abstract

With the rapid progress of large language models (LLMs), LLM-powered multi-agent systems (MAS) are drawing increasing interest across academia and industry. However, many current MAS frameworks struggle with reliability and scalability, especially on complex tasks. We present AgentGit, a framework that brings Git-like rollback and branching to MAS workflows. Built as an infrastructure layer on top of LangGraph, AgentGit supports state commit, revert, and branching, allowing agents to traverse, compare, and explore multiple trajectories efficiently. To evaluate AgentGit, we designed an experiment that optimizes target agents by selecting better prompts. We ran a multi-step A/B test against three baselines—LangGraph, AutoGen, and Agno—on a real-world task: retrieving and analyzing paper abstracts. Results show that AgentGit significantly reduces redundant computation, lowers runtime and token usage, and supports parallel exploration across multiple branches, enhancing both reliability and scalability in MAS development. This work offers a practical path to more robust MAS design and enables error recovery, safe exploration, iterative debugging, and A/B testing in collaborative AI systems.

Introduction

As large language models (LLMs) advance rapidly (Sindhu et al. 2024; Hadi et al. 2023), their use in multi-agent systems (MAS) is drawing growing interest from both academia and industry (Li et al. 2024; Yang et al. 2024). An MAS comprises multiple autonomous agents—often powered by LLMs—that interact within a shared environment to achieve complex goals through coordination and communication. MAS applications span many domains, including automated software development (Qian et al. 2023; Wang et al. 2024; He, Treude, and Lo 2025), scientific simulation (Uhrmacher and Weyns 2018), drug discovery and design (Fu et al. 2017; Kodela 2025), intelligent transportation (Zhou et al. 2022; Troullinos et al. 2021), financial analysis (Hafezi, Shahrabi, and Hadavandi 2015; Raudys and Zliobaite 2006), and intelligent tutoring in education (Vicari and Giraffa 2002).

However, current LLM-powered MAS fall short of industrial-grade needs, which require high reliability and treat scalability as a core property for future growth (Rana and Stout 2000; Lee et al. 1998). Empirical studies report that most proposed MAS achieve low accuracy (often below 50%) even within their target domains (Pan et al. 2025), and taxonomies of these failures point to systemic reliability gaps. In addition to reliability, scalability remains a major challenge.

Regarding the poor performance of LLM-powered MAS, we argue that issues like failed tool calls, unexecutable instructions, and endless reasoning loops are symptoms rather than root causes. The core limitation lies in the architecture: most agent frameworks lack a rollback mechanism. When execution fails, the system cannot be reverted to a stable state to explore alternative paths. As a result, a single erroneous action can cascade into full task failure, causing the entire agentic workflow to collapse irreversibly and wasting accumulated context.

Mainstream LLM agent frameworks such as LangChain (https://docs.langchain.com/) and LangGraph (Pelluru 2025) provide modular state control and composable workflows. However, their execution remains largely linear and irreversible: each agent action mutates state without built-in, lossless recovery. While LangGraph supports rollback, its mechanism discards intermediate results during restoration, limiting its capability to preserve the full execution context. When an error occurs, intermediate states are lost. Without commit/rollback, MAS cannot perform localized recovery, branching exploration, or incremental optimization. As a result, reliability and scalability are constrained in real-world, dynamic environments.

To address this limitation, we propose AgentGit, a framework that introduces Git-like version control semantics into agentic workflows. Built as an infrastructure layer on top of LangGraph, AgentGit provides state commit, state revert, and branching operations that allow agents to traverse, compare, and explore multiple trajectories. With this mechanism, an agent can automatically roll back to its last stable checkpoint upon failure and attempt a different strategy, transforming fragile, linear pipelines into robust, explorable, and self-correcting systems.

In terms of reliability, AgentGit enables fine-grained error recovery, deterministic replay, and safe experimentation. The rollback and branching capabilities allow agents to restore consistent states, isolate faulty trajectories, and re-execute alternative actions without compromising prior results. Furthermore, reproducible checkpoints facilitate systematic debugging, unit testing, and policy validation.

In terms of scalability, AgentGit supports parallel explo-

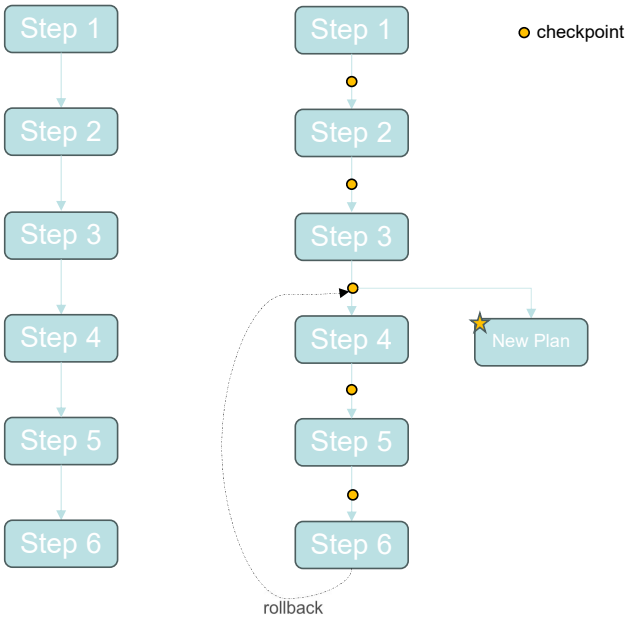

Figure 1: Comparison of task execution workflows: standard model vs. AgentGit with rollback functionality

ration across multiple branches, allowing diverse strategies to evolve independently without redundant computation. Its persistent checkpoint architecture allows large-scale agents to share and reuse prior states across sessions and tasks, thus providing a scalable foundation for building complex, multi-agent ecosystems.

To demonstrate the effectiveness of AgentGit, we designed a set of experiments to compare the efficiency of different frameworks in completing complex tasks. The experiment simulated a real-world scenario: retrieving abstracts of papers from arXiv on a specific topic and performing subsequent analysis and optimization. The workflow consisted of four steps: Search and Extract, Introduction, Analysis, and Discussion. In the experiment, we conducted an A/B test to compare the performance of four frameworks—LangGraph, AutoGen, Agno, and LangGraph+AgentGit—in terms of tool invocation and prompt generation. While all frameworks ultimately arrived at the global optimal solution, the focus of the experiment was to compare their efficiency, particularly in testing different tools and prompts.

The experimental results demonstrated that AgentGit significantly outperformed other frameworks in execution efficiency. By leveraging its unique rollback mechanism, AgentGit allowed skipping previously completed steps and directly testing new tools or prompts from specific nodes, thereby avoiding redundant execution of earlier workflows. Compared to other frameworks, AgentGit substantially reduced the overall runtime and resource consumption, confirming its reliability and scalability in complex task scenarios.

The contributions of this work are as follows:

- We propose AgentGit, the first multi-agent framework toolkit that introduces Git-like rollback and branching mechanisms into LLM-powered agent systems, enabling efficient and reversible execution in complex workflows.
- We analyze the theoretical complexity of the rollback mechanism in AgentGit, demonstrating its scalability and efficiency in reducing redundant computations during iterative tasks.
- We design and conduct an A/B test task to evaluate the performance of AgentGit in comparison to other frameworks (LangGraph, AutoGen, and Agno). Experimental results show that the rollback functionality of AgentGit significantly improves testing efficiency by reducing token consumption and optimizing runtime.
- We introduce potential application cases for AgentGit, such as error recovery, safe exploration, iterative debugging, and A/B testing, highlighting its versatility in accelerating MAS development and enhancing system robustness.
- We fully open-source our dataset, codebase, and the AgentGit framework to facilitate further research and development in the field of MAS.

Related Works

LLM-powered multi-agent frameworks provide standardized infrastructures for designing, orchestrating, and evaluating interactions among multiple agents powered by LLMs, thus simplify the development of LLM-powered MAS (Li et al. 2024; He, Treude, and Lo 2025). Such frameworks abstract complex coordination processes—such as message passing, role assignment, and tool invocation—into reusable components, enabling researchers and developers to efficiently prototype and deploy collaborative AI systems. In recent years, with the rapid development of LLM technology, the application scenarios of MAS have expanded significantly, including automated software development, scientific simulations, knowledge graph construction, and intelligent customer service systems (Qian et al. 2023; Pan et al. 2025). These scenarios demand higher reliability and scalability from MAS (Rana and Stout 2000; Lee et al. 1998).

The current mainstream multi-agent frameworks include LangGraph (LangChain 2025), Agno (Agno 2025), Autogen (Wu et al. 2024), CrewAI (CrewAI 2025) and Dify (Dify 2025). LangGraph adopts a graph-based orchestration paradigm, representing agent workflows as directed acyclic graphs. This design ensures deterministic control and reproducibility, making it well-suited for structured pipelines. In contrast, AutoGen models agents as conversational entities that exchange natural language messages, allowing flexible, dialogue-driven coordination among agents and humans. CrewAI organizes agents into role-based teams, where each agent is assigned a specific responsibility under a shared objective—an approach that closely resembles human organizational structures and facilitates multi-role collaboration. Agno focuses on flexible orchestration and adaptive agent coordination, allowing the system to evolve its behavior over time. Dify offers a low-code environment where developers can visually compose

and deploy agent workflows, aiming for ease of deployment rather than experimental control.

Although these frameworks exhibit certain advantages in specific scenarios, they share a critical limitation: the lack of rollback mechanisms. Specifically, these frameworks cannot restore to previously stable states during task execution, nor can they support multi-branch exploration or error recovery. Once an agent executes an incorrect action, the overall workflow often fails irreversibly, requiring human intervention. What's more, while LangGraph supports rollback, its mechanism deletes subsequent results upon reverting to a previous state, limiting its ability to retain and reuse intermediate data. This limitation not only increases the cost of task failures but also restricts the applicability of MAS in complex and dynamic environments.

Our work addresses this limitation by introducing a rollback-capable multi-agent framework that enhances robustness through reversible execution and controlled state restoration, enabling more reliable multi-agent collaboration. Furthermore, our framework supports multi-branch exploration and persistent state storage, allowing MAS to perform efficient testing and optimization in complex task scenarios.

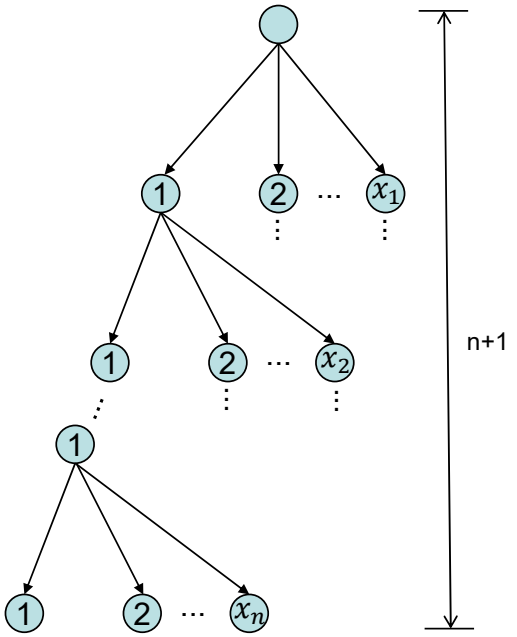

Figure 2: Tree diagram illustrating the branching structure of the task execution process

AgentGit

AgentGit significantly improves the execution efficiency and flexibility of MAS by introducing rollback and branching mechanisms.

Rollback

The rollback mechanism is one of the core features of AgentGit. It creates permanent checkpoints to save the complete system state, including session history, tool invocation records, environment variables, and intermediate reasoning processes. When rollback is required, the system can restore its state from a specified checkpoint, avoiding the need to re-execute previously completed steps. Checkpoints can be created manually or automatically triggered after critical operations. The need for rollback is particularly evident in complex task scenarios, such as testing the effectiveness of different tools or prompts. The rollback mechanism allows the system to directly test new approaches from a specific checkpoint without re-executing earlier steps, significantly reducing overall runtime and resource consumption. The rollback process involves the following steps: first, the system loads the state corresponding to the checkpoint ID specified by the user; next, it restores session history and tool invocation records; finally, it resumes subsequent tasks from the restored state. Through this mechanism, AgentGit effectively optimizes task execution workflows and enhances overall system efficiency.

Figure 1 shows the differences between the standard model and AgentGit with rollback mechanism in the task execution process. The standard model in the left executes tasks linearly, with each step permanently altering the system state. If an error occurs or adjustments are needed at Step 4, the system must restart from Step 1, leading to redundant computation and resource waste. AgentGit creates checkpoints after each critical step (e.g., Step 3). If adjustments are needed at Step 4, the system can roll back to the checkpoint at Step 3 and directly test new approaches from that state, without re-executing Steps 1, 2 and 3. This mechanism significantly reduces redundant computation and improves efficiency.

Branching

Branching enables the creation of new branch paths from specific checkpoints. This mechanism allows the system to independently explore different strategies or approaches across multiple paths. Each branch inherits the complete state information of the original path, including session history, tool invocation records, environment variables, and intermediate reasoning processes, ensuring the integrity and independence of each branch. The process of creating a branch involves the following steps: first, the system loads the state from the specified checkpoint; second, a new branch ID is generated, and the branch environment is initialized; finally, users can test different tools or prompt strategies on the new branch without affecting the execution of the original path.

The significant advantage of the branching mechanism is its support for parallel computation. Multiple branches can run simultaneously, testing different strategies or approaches, thereby significantly improving task execution efficiency. For instance, in a complex task, users can test the effectiveness of Tool A on one branch while testing Tool B on another branch, without waiting for one test to complete before starting the next. Additionally, branching supports navigation and merging operations, allowing users to switch between branches, review execution results, and integrate the outcomes of multiple branches. The merging pro-

cess is similar to Git's branch merging operation, supporting conflict detection and resolution to ensure the completeness and consistency of the final result.

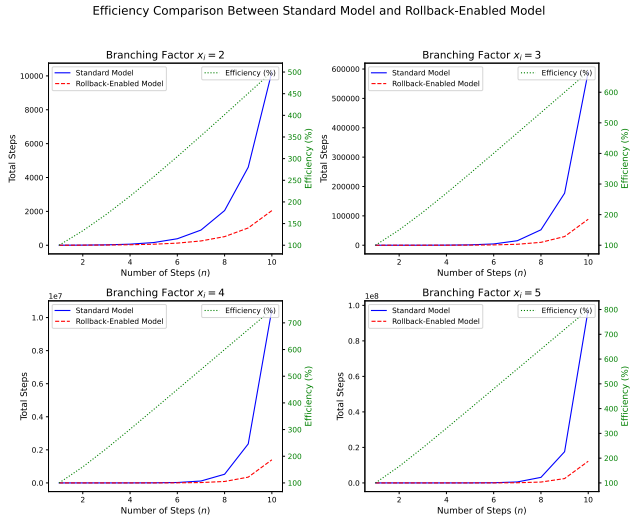

Figure 3: Visualization of the total steps required and efficiency trends for the standard model and rollback-enabled model under varying x_i and n

Complexity Analysis

Complexity analysis demonstrates that Agent Git's mechanism effectively reduces redundant computations and optimizes resource utilization in high-complexity tasks, providing a reliable and efficient solution for the development of MAS.

Lemma 1. *In an MAS, a workflow consists of n steps, where each step allows the selection of different tools or prompt options. Suppose the i-th step has x_i available tools or prompt options. Then, the total number of possible outcomes L after executing the workflow can be expressed as:*

$$L = \prod_{i=1}^{n} x_i,$$

where x_i represents the number of tools or prompt options available at the i-th step.

To facilitate the understanding of the task execution process and rollback mechanism, we use a tree diagram to represent the branching structure of the entire task, as illustrated in Figure 2. In the tree diagram, the *root node* represents the initial input of the task, such as the original task or question provided by the user. The *intermediate nodes* represent the intermediate results after each step of execution, which can be stored as checkpoints for rollback or branching operations. The *leaf nodes* represent the final results of the task, such as the optimized content or results generated after completing all steps.

The *edges* of the tree represent the specific execution paths from one node to the next, reflecting the choice of tools or LLMs. The *height of the tree* is $n+1$, where n is the number of steps in the task. Each step corresponds to one layer of the tree, and the root node occupies the first layer as the initial input, making the total height of the tree $n+1$. For the tree structure, each node at the $(i-1)$th layer has x_i possible execution paths leading to nodes in the ith layer. Therefore, the number of leaf nodes in the tree L can be expressed as:

$$L = x_1 \cdot x_2 \cdot x_3 \cdot \cdots \cdot x_n = \prod_{i=1}^{n} x_i.$$

Lemma 2. *In an MAS without rollback mechanism, if the workflow consists of n steps, and the ith step has x_i possible tools or prompt options, then the total number of steps required to generate all possible outputs is given by:*

$$\mathcal{S}_{std} = n \prod_{i=1}^{n} x_i.$$

Let $\mathcal{S}_{\text{std}}$ denote the total steps for the standard model. Since each path contains n steps and the number of leaf nodes in the tree $L = \prod_{i=1}^{n} x_i$. Without a rollback mechanism, the standard model needs to execute the full path for each leaf node, so

$$\mathcal{S}_{\text{std}} = n \prod_{i=1}^{n} x_i.$$

Lemma 3. *In an MAS with rollback mechanism, if the workflow consists of n steps, and the ith step has x_i possible tools or prompt options, then the total number of steps required to generate all possible outputs is given by:*

$$\mathcal{S}_{rollback} = \sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} x_j \cdot x_i \right).$$

Let $\mathcal{S}_{\text{rollback}}$ denote the total steps for the rollback-enabled model. With a rollback mechanism, the model can avoid re-executing previous steps by rolling back to a checkpoint, thus the total number of steps required to generate all leaf nodes equals the total number of edges in the tree, so

$$\begin{aligned}\mathcal{S}_{\text{rollback}} &= x_1 + x_1 \cdot x_2 + x_1 \cdot x_2 \cdot x_3 + \cdots + \prod_{i=1}^{n} x_i \\ &= \sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} x_j \cdot x_i \right),\end{aligned}$$

where $\prod_{j=1}^{i-1} x_j$ represents the number of nodes at the $i-1$th layer, and x_i represents the number of branches for each node.

It indicates significantly from above that the total number of steps required differs largely between a standard MAS and a rollback-enabled one.

Figure 3 visualizes the total steps required for both the standard MAS and the rollback-enabled one. In this figure, we assume that each intermediate node has 2, 3, 4, or 5 child nodes or branches (i.e., $x_i = 2, 3, 4, 5$), and vary the number

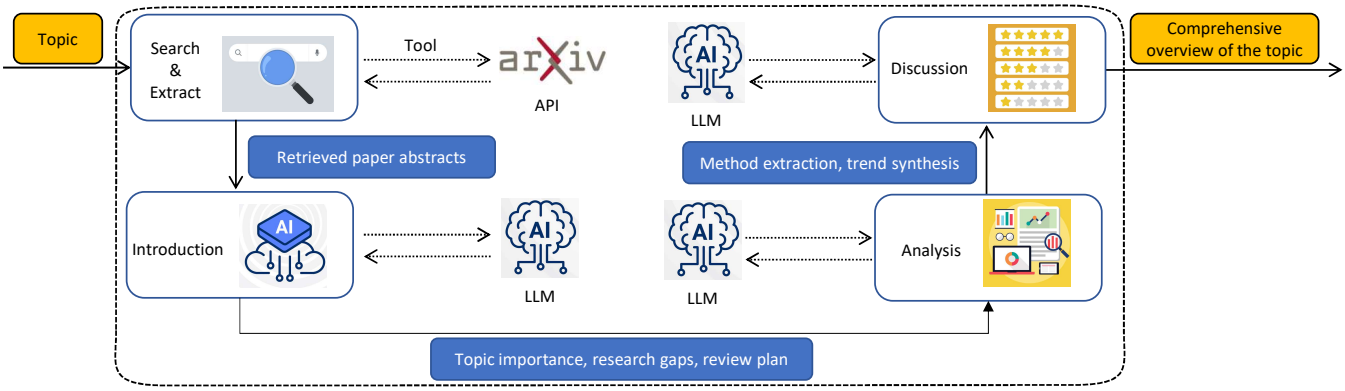

Figure 4: Workflow of the MAS task scenario for retrieving abstracts of papers related to a specific topic

of steps n. From Figure 3, we can observe that the rollback mechanism significantly reduces the total number of steps required to generate all leaf nodes, especially as the number of steps n increases. This demonstrates the substantial efficiency improvement achieved by Agent Git in complex task scenarios.

Proposition 1. *In an MAS with n steps, if each step has α possible tools or prompt options (where α is a constant), define efficiency η as the ratio of the total steps required by a system without rollback mechanism to the total steps required by a system with rollback mechanism to generate all possible final results. When $N \to \infty$, the growth trend of efficiency η becomes infinite.*

From above, let η denote the efficiency To quantify the efficiency difference between the two mechanisms, defined as the ratio of the total steps required by the standard model to the total steps required by the rollback-enabled model:

$$\eta = \frac{\mathcal{S}_{\text{std}}}{\mathcal{S}_{\text{rollback}}} = \frac{n \prod_{i=1}^{n} x_i}{\sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} x_j \cdot x_i \right)}.$$

Consider a special case where each step has α possible tools or prompt options. This is equivalent to saying that the number of branches for each intermediate node x_i is a constant α. In this scenario, the efficiency η can be rewritten as:

$$\eta = \frac{n\alpha^n}{\sum_{i=1}^{n} \alpha^i}.$$

When n approaches infinity, we can analyze the growth trend of efficiency using the limit:

$$\lim_{n\to\infty} \eta = \lim_{n\to\infty} \frac{n\alpha^n}{\alpha \cdot \frac{\alpha^n - 1}{\alpha - 1}} = \lim_{n\to\infty} n \cdot \frac{\alpha - 1}{\alpha} \frac{1}{1 - \alpha^{-n}}$$
$$= \lim_{n\to\infty} n \cdot (1 - \alpha^{-1}) = \infty.$$

This indicates that the growth trend of efficiency η is infinite as n increases, and Figure 3 intuitively demonstrates this point.

Proposition 2. *In an MAS with n steps, if each step has α possible tools or prompt options (where α is a constant), define efficiency η as the ratio of the total steps required by a system without rollback mechanism to the total steps required by a system with rollback mechanism to generate all possible final results. When $N \to \infty$, the limit of η/n approaches $\frac{\alpha-1}{\alpha}$.*

η/n represents the average efficiency improvement brought by each step of task execution. By calculating $\lim_{n\to\infty} \eta/n$, we can quantify the performance of the rollback mechanism in high-complexity tasks:

$$\lim_{n\to\infty} \frac{\eta}{n} = \lim_{n\to\infty} \frac{\alpha^n}{\sum_{i=1}^{n} \alpha^i} = \lim_{n\to\infty} \frac{\alpha^n}{\alpha \cdot \frac{\alpha^n - 1}{\alpha - 1}}$$
$$= \lim_{n\to\infty} \frac{\alpha - 1}{\alpha} \frac{1}{1 - \alpha^{-n}} = \frac{\alpha - 1}{\alpha}.$$

As the task complexity increases, the average efficiency improvement per unit task complexity stabilizes, with its limit given by:

$$\lim_{n\to\infty} \frac{\eta}{n} = \frac{\alpha - 1}{\alpha}.$$

This result indicates that the impact of the number of branches α on efficiency improvement diminishes as α increases, and the average efficiency improvement approaches 1 when α tends to infinity. Figure 3 visually illustrates this observation. From the curves in the figure, it can be seen that their slopes gradually stabilize as $N \to \infty$.

Experiment

While AgentGit has a wide range of potential applications, such as error recovery, safe exploration, iterative debugging, and A/B testing, we chose a representative experiment to demonstrate its efficiency in complex task scenarios—retrieving abstracts of papers from arXiv on a specific topic and performing subsequent analysis and optimization. In this experiment, we conducted an A/B test to compare the effectiveness of different prompt generation methods, aiming to identify the optimal combination for completing the task.

Experimental Setup

Task Scenario This experiment simulates an MAS task scenario, where the task is to retrieve abstracts of papers related to a specific topic from arXiv, analyze and optimize

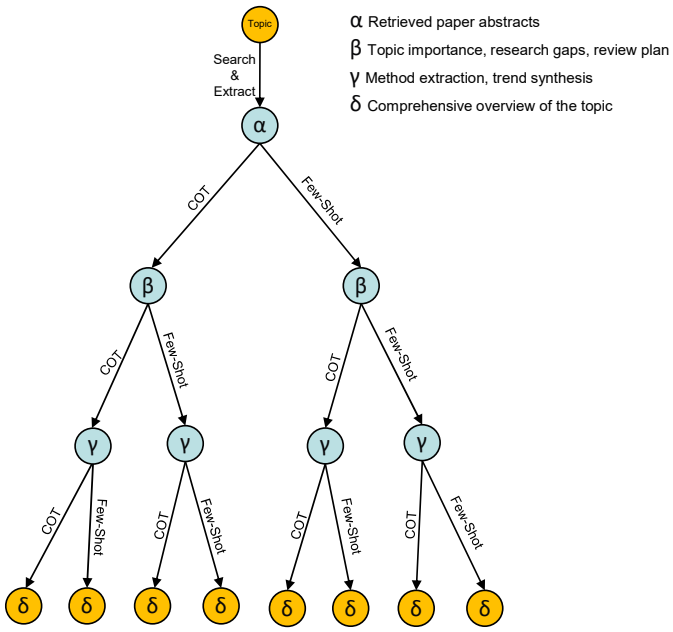

Figure 5: Tree structure representing the experimental workflow

these abstracts, and generate a final comprehensive report. The workflow of the task scenario is illustrated in Figure 4. This task aims to identify the optimal prompt generation methods through A/B test and consists of four steps: Search and Extract, Introduction, Analysis, and Discussion. First, the system retrieves paper titles and abstracts related to the specified topic by invoking arXiv API. Then, it filters and extracts abstracts from the retrieved papers to ensure relevance and uniqueness. Next, the system generates an Introduction based on the extracted abstracts using different prompt generation methods. Following this, the system performs an Analysis of the abstracts, extracting key insights and evaluating their contributions. Finally, the system generates a Discussion section to summarize the findings and provide a comprehensive overview of the topic.

Baselines The experiment compares four frameworks: LangGraph, AutoGen, LangGraph+AgentGit, and Agno. These frameworks represent different approaches to MAS design, with LangGraph and AutoGen serving as baseline frameworks, Agno focusing on task decomposition and collaboration, and LangGraph+AgentGit introducing rollback mechanisms to optimize task execution. These frameworks were selected to provide a comprehensive comparison of tool invocation and prompt generation efficiency.

Experimental Design The experiment conducts A/B test to compare the effectiveness of different prompt generation methods to identify the optimal approach for completing the task. Specifically, in the Search and Extract step, the system utilized the arXiv API to retrieve paper abstracts related to the specified topic. In the subsequent steps—Introduction, Analysis, and Discussion—prompts generated by different methods, including COT Prompt (Wei et al. 2022b,a) and

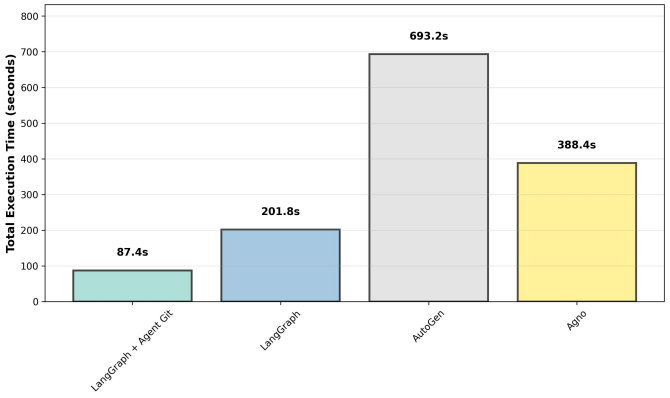

Figure 6: Execution time comparison of different frameworks for completing the task

Few-Shot Prompt (Mann et al. 2020), were compared to evaluate their impact on task performance.

For this experiment, we used LangGraph version 0.6.6 (released on August 20, 2025) as the baseline framework, and AgentGit version 0.0.1 was integrated into LangGraph to enable rollback mechanisms. Additionally, all frameworks utilized the GPT-4o-mini model with a temperature setting of 0 to ensure consistent outputs for identical inputs.

To evaluate the quality of the final outputs generated by each framework, we employed G-eval (Liu et al. 2023), a widely used evaluation metric for assessing the coherence, relevance, and overall quality of generated text. G-eval scores were assigned to the comprehensive reports produced by each framework under specific prompt combinations, such as COT-COT-COT and Few-Shot-Few-Shot-Few-Shot. This scoring method ensured an objective comparison of the frameworks' ability to generate high-quality outputs.

The experimental design is illustrated in Figure 5, which represents the workflow as a tree structure. The root node corresponds to the initial input provided to the MAS, such as the specified topic for retrieving paper abstracts. Each edge in the tree represents a specific operation, such as

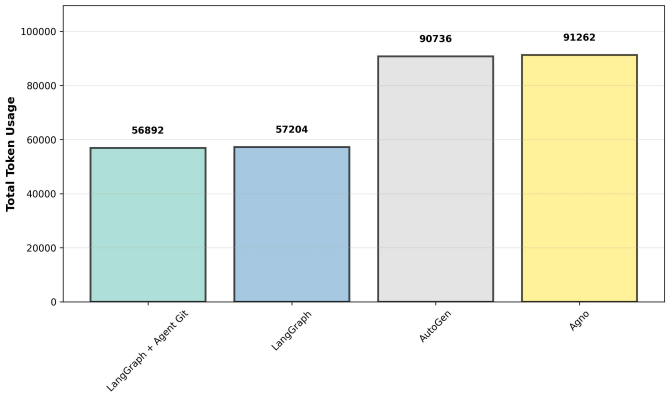

Figure 7: Token usage comparison of different frameworks for completing the task

”Search and Extract with arXiv API” or ”Analysis with prompt generated by COT method.” Intermediate nodes represent checkpoints, where the system stores the state of the task after completing a specific step. The leaf nodes correspond to the final polished outputs, which are comprehensive reports containing the refined abstracts of papers. This tree structure visually captures the branching possibilities at each step, allowing for systematic exploration of different tools and prompt generation methods.

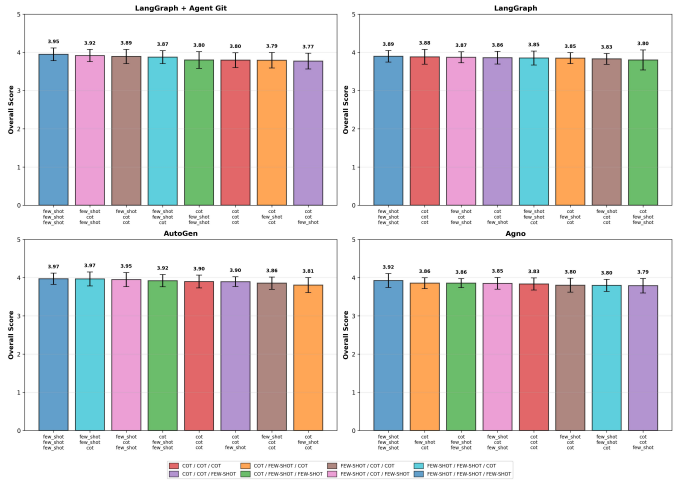

Figure 8: G-eval scores of final outputs generated by specific prompt combinations in different frameworks

Results

Execution Time Analysis Figure 6 illustrates the execution time of the four frameworks for completing the task. LangGraph+AgentGit significantly outperformed the other frameworks in terms of runtime, achieving the shortest execution time. This improvement can be attributed to the rollback and branching mechanism introduced by AgentGit, which allows the system to avoid re-executing previously completed steps and process in parallel. In contrast, LangGraph supports rollback but lacks branching capabilities, requiring sequential execution for each branch tested. AutoGen and Agno, on the other hand, lack both rollback and branching mechanisms, resulting in repeated execution of all four steps for every new test. This result highlights the efficiency of AgentGit in optimizing task execution in complex scenarios.

Token Consumption Analysis Figure 7 compares the token consumption of the four frameworks during task execution. The experimental results show that LangGraph+AgentGit consumed significantly fewer tokens than AutoGen and Agno, and slightly fewer than LangGraph, remaining almost consistent with it. This result aligns with intuition, as LangGraph also utilizes its rollback mechanism (although it does not permanently store intermediate processes), thereby avoiding the repeated generation of large amounts of content. However, the token consumption of LangGraph+AgentGit is not entirely identical to LangGraph, which may be due to the variance in output generation by large models. Specifically, for the same input, the output may exhibit slight variations, leading to differences in token usage.

Prompt Combination Performance Figure 8 presents the performance scores of different frameworks under specific prompt combinations, such as COT-COT-COT. The experimental results show that for specific prompt combinations, the scores of the four frameworks vary slightly but remain generally consistent. Additionally, for different prompt combinations, the scores within the same framework are also largely consistent. This may indicate that the effects of COT Prompt and Few-Shot Prompt do not differ significantly. The slight variations in scores across frameworks may be attributed to the inherent variance in large language models, which can introduce subtle differences in output generation for identical inputs. While further investigation is needed to fully confirm the consistency of the experimental results, it can be observed that the performance of all four frameworks under different prompt combinations is generally consistent. This consistency further validates the robustness of the frameworks under different prompt combinations, while also demonstrating that LangGraph+AgentGit can maintain output quality comparable to other frameworks while improving execution efficiency.

Conclusion

In this paper, we introduced AgentGit, a novel framework that integrates Git-like rollback and branching mechanisms into LLM-powered MAS. By enabling state commit, state revert, and branching operations, AgentGit addresses critical limitations in reliability and scalability faced by existing MAS frameworks. Our experimental results demonstrate that AgentGit significantly improves execution efficiency by reducing redundant computations, optimizing runtime, and minimizing token consumption. These findings validate the effectiveness of AgentGit in complex task scenarios, such as retrieving and analyzing paper abstracts.

Beyond the scope of the experiments presented, AgentGit opens up new possibilities for MAS development and application. Its rollback and branching capabilities provide robust solutions for error recovery, safe exploration, iterative debugging, and A/B testing, enabling developers to systematically optimize workflows and enhance system reliability. By supporting parallel exploration and persistent state storage, AgentGit lays the groundwork for scalable and adaptive multi-agent ecosystems, paving the way for future advancements in collaborative AI systems.

References

Agno. 2025. Agno: Multi-Agent System Framework. https://docs.agno.com/introduction. Accessed: 2025-10-31.

CrewAI. 2025. CrewAI: Multi-Agent System Framework. https://github.com/crewAIInc/crewAI. Accessed: 2025-10-31.

Dify. 2025. Dify: Multi-Agent System Framework. https://github.com/langgenius/dify. Accessed: 2025-10-31.

Fu, R.-g.; Sun, Y.; Sheng, W.-b.; and Liao, D.-f. 2017. Designing multi-targeted agents: An emerging anticancer drug discovery paradigm. *European journal of medicinal chemistry*, 136: 195–211.

Hadi, M. U.; Qureshi, R.; Shah, A.; Irfan, M.; Zafar, A.; Shaikh, M. B.; Akhtar, N.; Wu, J.; Mirjalili, S.; et al. 2023. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea preprints*, 1(3): 1–26.

Hafezi, R.; Shahrabi, J.; and Hadavandi, E. 2015. A bat-neural network multi-agent system (BNNMAS) for stock price prediction: Case study of DAX stock price. *Applied Soft Computing*, 29: 196–210.

He, J.; Treude, C.; and Lo, D. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5): 1–30.

Kodela, K. C. 2025. Autonomous Agentic AI Systems for Pharmaceutical Drug Discovery: A Multi-Agent Framework for Molecular Design and Optimization. *Available at SSRN 5382801*.

LangChain. 2025. LangGraph: Multi-Agent System Framework. https://github.com/langchain-ai/langgraph. Accessed: 2025-10-31.

Lee, L. C.; Nwana, H. S.; Ndumu, D. T.; and De Wilde, P. 1998. The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, 16(3): 94–103.

Li, X.; Wang, S.; Zeng, S.; Wu, Y.; and Yang, Y. 2024. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1): 9.

Liu, Y.; Iter, D.; Xu, Y.; Wang, S.; Xu, R.; and Zhu, C. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*.

Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1(3): 3.

Pan, M. Z.; Cemri, M.; Agrawal, L. A.; Yang, S.; Chopra, B.; Tiwari, R.; Keutzer, K.; Parameswaran, A.; Ramchandran, K.; Klein, D.; et al. 2025. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

Pelluru, K. 2025. LangChain & LangGraph in Production: Architectures for Multi-Agent LLM Systems. *Journal of Data and Digital Innovation*, 2(3): 1–9.

Qian, C.; Liu, W.; Liu, H.; Chen, N.; Dang, Y.; Li, J.; Yang, C.; Chen, W.; Su, Y.; Cong, X.; et al. 2023. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

Rana, O. F.; and Stout, K. 2000. What is scalability in multi-agent systems? In *Proceedings of the fourth international conference on Autonomous agents*, 56–63.

Raudys, Š.; and Zliobaite, I. 2006. The multi-agent system for prediction of financial time series. In *International Conference on Artificial Intelligence and Soft Computing*, 653–662. Springer.

Sindhu, B.; Prathamesh, R.; Sameera, M.; and KumaraSwamy, S. 2024. The evolution of large language model: Models, applications and challenges. In *2024 international conference on current trends in advanced computing (ICCTAC)*, 1–8. IEEE.

Troullinos, D.; Chalkiadakis, G.; Papamichail, I.; and Papageorgiou, M. 2021. Collaborative multiagent decision making for lane-free autonomous driving. In *Proceedings of the 20th international conference on autonomous agents and multiagent systems*, 1335–1343.

Uhrmacher, A. M.; and Weyns, D. 2018. *Multi-Agent systems: Simulation and Applications*. CRC press.

Vicari, R. M.; and Giraffa, L. M. M. 2002. The Use of Multi-Agent Systems to Build Intelligent Tutoring Systems. *AIP Conference Proceedings*, 627(1): 340–348.

Wang, X.; Li, B.; Song, Y.; Xu, F. F.; Tang, X.; Zhuge, M.; Pan, J.; Song, Y.; Li, B.; Singh, J.; et al. 2024. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Yang, Y.; Peng, Q.; Wang, J.; Wen, Y.; and Zhang, W. 2024. LLM-based Multi-Agent Systems: Techniques and Business Perspectives. *arXiv preprint arXiv:2411.14033*.

Zhou, W.; Chen, D.; Yan, J.; Li, Z.; Yin, H.; and Ge, W. 2022. Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic. *Autonomous Intelligent Systems*, 2(1): 5.