SAGE: Self-evolving Agents with Reflective and Memory-augmented Abilities

Anonymous ACL submission

Abstract

Large language models (LLMs) have made significant advances in the field of natural language processing, but they still face challenges such as continuous decision-making, lack of long-term memory, and limited context windows in dynamic environments. To address these issues, this paper proposes an innovative framework—Self-evolving Agents with Reflective and Memory-augmented Abilities (SAGE). The SAGE framework comprises three agents: the User, the Assistant, and the Checker. By integrating iterative feedback, reflective mechanisms, and a memory optimization mechanism based on the Ebbinghaus forgetting curve, it significantly enhances the agents' capabilities in handling multi-tasking and long-span information. The agents, through self-evolution, can adaptively adjust strategies, optimize information storage and transmission, and effectively reduce cognitive load. We evaluate the performance of the SAGE framework on AgentBench(Liu et al., 2023b) and long text tasks. Experimental results show that SAGE significantly improves model performance, achieving a 2.26X improvement on closed-source models and an improvement ranging from 57.7% to 100% on open-source models, with particularly notable effects on smaller models (see Table 1).

1 Introduction

In recent years, large language models (LLMs) have made significant progress in the field of natural language processing, demonstrating powerful performance in tasks such as dialogue and text generation(Brown et al., 2020). However, these models still face several challenges: (1) Agents need to continuously make decisions in changing environments and adapt to new situations and tasks. (2) Agents lack long-term memory mechanisms, which is increasingly evident in situations requiring sustained interaction with the environment(Graves et al., 2016). The limited context window also hinders the model's ability to handle information over long time spans(Rae et al., 2019).

To tackle these challenges, researchers have proposed meta-learning and multi-task learning to enhance the transferability and adaptability of LLM agents. For memory limitations, prior works like MemGPT(Packer et al., 2024) use a FIFO queue to manage forgetting, while MemoryBank employs a forgetting curve based on insertion time. However, these approaches are often task-specific, lacking a general framework to systematically improve LLM agents in complex environments. Recent innovations, such as AutoGPT[1] and BabyAGI[2], leverage LLMs as core controllers, aiming to solve real-world challenges. Yet, multi-agent frameworks still face issues like communication overload, heavily relying on memory to maintain context. As interaction history grows, resource demands and latency increase, limiting efficient deployment in practical scenarios.

In this paper, we propose an innovative framework, Self-evolving Agents with reflective and memory-augmented abilities (SAGE). By enhancing agents' self-adjustment capabilities through reflection, they can more effectively utilize historical information and make efficient decisions when faced with complex and dynamic tasks. From the perspective of self-evolution, we introduce a memory optimization mechanism based on the Ebbinghaus forgetting curv(Ebbinghaus, 1885). This mechanism helps agents selectively retain key information, optimize information storage and transmission, reduce unnec-

[1] https://github.com/Significant-Gravitas/AutoGPT
[2] https://github.com/yoheinakajima/babyagi

essary cognitive load, and enhance agents' capabilities in interaction tasks with the environment. Experimental results show that our approach achieves significant improvements across various benchmarks, particularly excelling in smaller models. Specifically, on AgentBench, the performance of powerful LLMs like GPT-3.5 and GPT-4 is enhanced by up to 2.26X. For open-source models, performance improvements range from 57.7% to 100%. In tasks such as multi-source question answering and code generation, our approach achieves state-of-the-art results(Etezadi and Shamsfard, 2023).

The main contributions of our work are as follows:

- We propose a novel framework, SAGE, which introduces a reflection mechanism to enhance the self-adjustment capabilities of agents. Without any additional training, this enables agents to more effectively utilize historical information and make efficient decisions when faced with complex and dynamic tasks.

- We introduce a memory optimization mechanism based on the Ebbinghaus forgetting curve. This helps agents selectively retain key information, reducing the issue of information overload in multi-agent systems.

- SAGE achieves improvements over strong baselines in multiple challenging real-world tasks and achieves state-of-the-art results on benchmarks. This framework can be extended to other LLMs, with particularly notable improvements in smaller models.

2 Related work

2.1 Self-Improvement of Reasoning and Decision-Making

A lot of research is focused on making large language models (LLMs) better at improving themselves. Some researchers are working on using carefully crafted prompts to help models learn how to get better, although this usually only works for one-off tasks. Others are tweaking how models get feedback during tasks, which helps them get better at thinking things through(Huang et al., 2022). There's also work on using strategies like random beam searches to help models make smarter decisions and assess their own work. Most current methods rely on quick, one-off tweaks and learning strategies that need lots of resources and hands-on tech help(Tian et al., 2024). This paper introduces a self-reflection mechanism, showing that LLMs can keep getting better and produce higher quality work across different tasks, all without needing extra training.

2.2 Memory Mechanism for LLM-based Agents

In LLM-based agents, the memory module is a critical component responsible for storing, processing, and retrieving task-related information. This memory plays a pivotal role in how the agent accumulates knowledge, handles historical experiences, and supports its decision-making processes. To enhance the self-evolution capabilities of these agents, researchers are focused on designing and optimizing these memory modules(Raffel et al., 2020). Past research has covered various designs and implementations of memory modules. This includes integrating information from different trials to boost reasoning abilities or storing information in natural language to enhance the module's interpretability and user-friendliness(Wada et al., 2019). While there has been progress, further improvements in self-adjustment capabilities and memory management efficiency are still needed to better address complex problems in real-world applications. Our proposed memory optimization mechanism is designed to help agents better manage and adapt to dynamic and complex task environments.

3 Method

In this section, we present the SAGE framework, designed to improve agent performance by leveraging three core mechanisms: iterative feedback, reflection, and MemorySyntax (as shown in Figure 1). The assistant agent A iteratively updates its policy π_θ based on feedback f_t provided by the checker agent C, optimizing over successive iterations to maximize the expected reward R. The reflection mechanism allows A to incorporate historical observations $\mathcal{O}_t$ and actions $\mathbf{a}_t$, forming a self-reflection r_t, which is stored in the memory $\mathcal{M}_L$ for future decision-making. Finally, MemorySyntax combines the Ebbinghaus forgetting curve with linguistic principles to manage memory decay, dynamically updating the agent's short-term

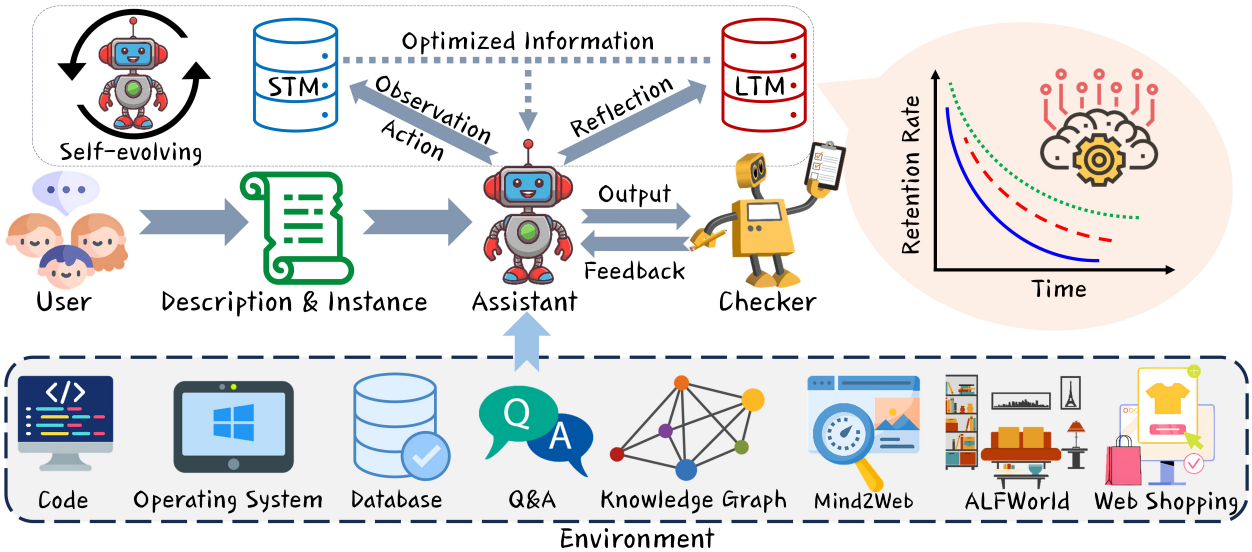

Figure 1: An illustration of the SAGE.

memory $\mathcal{M}_S$ and long-term memory $\mathcal{M}_L$ by prioritizing information based on its retention strength $S(I_t)$, thus improving the agent's ability to retain crucial information while discarding less relevant data. In the following subsections, we describe the detailed implementation of these components.

3.1 Iterative Feedback

The iterative feedback mechanism in the SAGE framework enables the assistant agent A to refine its policy π_θ through repeated interactions with the checker agent C. At each iteration t, the assistant receives feedback f_t based on its current output $\mathbf{o}_t$, and adjusts its policy accordingly. This process continues until the checker validates the output or the iteration cap N is reached, ensuring that A incrementally optimizes its decisions to improve task performance over successive iterations.

3.1.1 Initialization Phase

Role Assignment. In the SAGE framework, three agents are introduced: the user U, the assistant A, and the checker C. The user, upon receiving prompt P_U, assumes the role of task proposer by specifying a task $\mathcal{T}_U$ and related constraints $\mathcal{C}_U$. The assistant, upon receiving prompt P_A, generates a sequence of actions $\mathbf{a}_t$ based on the observations $\mathcal{O}_t$ and environment $\mathcal{E}$. The checker C evaluates the output $\mathbf{o}_A$ produced by the assistant, providing feedback f_C based on the discrepancy between $\mathbf{o}_A$ and the expected result, updating its policy π_θ iteratively to minimize this gap.

Task Assignment. The task $\mathcal{T}_U$ provided by the user includes an initial task description $\mathbf{d}_U$ and an instance $\mathbf{i}_U$ that serves as the reference for correct output. This forms the input set $\mathcal{I}_A = (\mathbf{d}_U, \mathbf{i}_U)$ for the assistant to initiate its generative process. The assistant then proceeds by selecting an action $\mathbf{a}_t$ at each time step t, guided by π_θ, with the goal of maximizing the reward R_t for completing $\mathcal{T}_U$.

3.1.2 Actual Interaction Phase

Following the role assignment and task definition in the initialization phase, the assistant A transitions into the actual interaction phase to generate outputs aimed at accomplishing the task $\mathcal{T}_U$. In this phase, A iteratively produces outputs $\mathbf{o}_t$ at each time step t based on the task description $\mathbf{d}_U$ and instance $\mathbf{i}_U$ provided in the input set $\mathcal{I}_A = (\mathbf{d}_U, \mathbf{i}_U)$. At each time step t, the assistant selects an action $\mathbf{a}_t$ by following its policy π_θ, which is conditioned on the current state s_t, the reward signal R_t (the reward score for task performance), and feedback f_t^i from the checker C. This decision-making process is formalized as:

$$\mathbf{o}_t \sim \pi_\theta(\mathbf{o}_t \mid s_t, R_t, f_t^i), \tag{1}$$

where π_θ represents the assistant's policy, R_t reflects the reward signal based on task performance at time t, and f_t^i is the feedback provided by the checker during the i-th iteration.

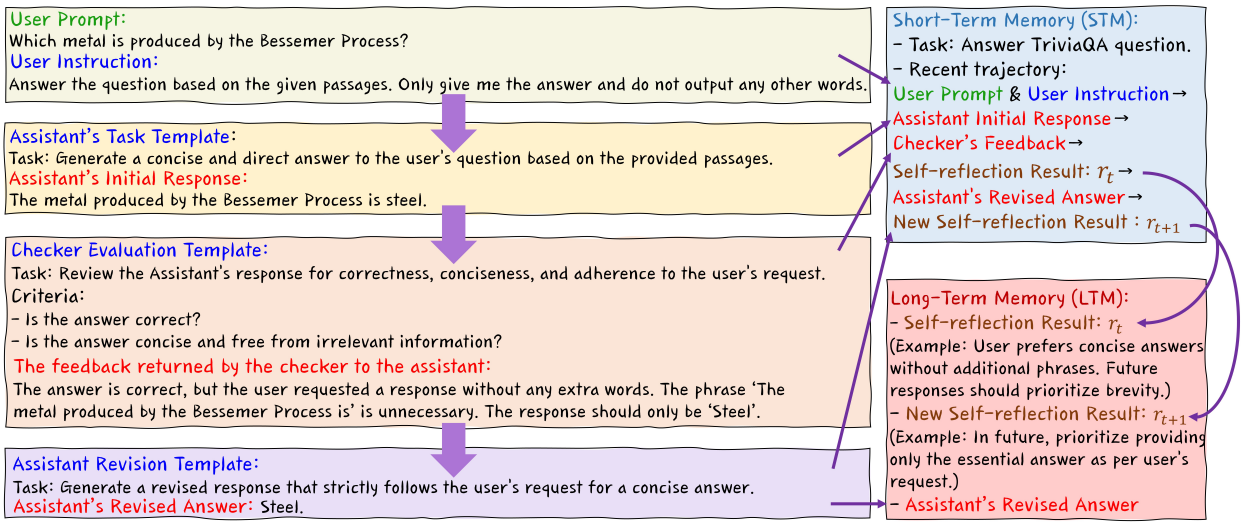

Figure 2: An example of the assistant's iterative workflow, including checker evaluation, prompt templates for feedback, and reflection processes integrating short-term and long-term memory.

As the interaction progresses, the checker C evaluates each output $\mathbf{o}_t$ generated by A, comparing it against the expected outcome derived from $\mathbf{i}_U$. Based on this comparison, the checker provides iterative feedback f_t^i to guide A in refining its actions $\mathbf{a}_t$ and outputs $\mathbf{o}_t$. The iterative refinement continues until either the checker validates the output as correct or the iteration limit N is reached.

3.1.3 Evolutionary Goals and Directions

Leveraging the feedback f_t^i obtained at each iteration t, the assistant A formulates new evolutionary objectives:

$$\mathcal{G}^{t+1} = (\mathcal{A}^{t+1}, \mathcal{D}^{t+1}), \tag{2}$$

where $\mathcal{A}^{t+1}$ represents the updated memory optimization mechanisms, and $\mathcal{D}^{t+1}$ refers to the refined self-adjustment capabilities. These evolutionary objectives guide the assistant in updating its policy π_θ for the subsequent iteration. The policy update is governed by the function ψ, which integrates the current policy π_θ^t with the new evolutionary objectives $\mathcal{G}^{t+1}$:

$$\pi_\theta^{t+1} = \psi(\pi_\theta^t, \mathcal{G}^{t+1}). \tag{3}$$

The iterative policy refinement enables the assistant A to continuously adapt its strategies based on cumulative feedback and evolving task requirements, thereby improving its overall performance in dynamic environments.

3.2 Memory Management

The SAGE framework implements a dual-memory system, consisting of Short-Term Memory (STM) and Long-Term Memory (LTM), to manage task-relevant information and enhance the agent's reasoning and decision-making capabilities (see Figure 2 for a visual representation of this process).

Short-Term Memory (STM). STM is responsible for storing immediate, task-specific data with limited capacity. It updates rapidly with new observations ($\mathcal{O}_t$) and actions ($\mathbf{a}_t$), maintaining a recent trajectory history $\mathcal{T}_t = (\mathcal{O}_t, \mathbf{a}_t)$. This allows the agent to make real-time decisions and respond quickly to dynamic changes in the environment (Mnih et al., 2015).

Long-Term Memory (LTM). LTM retains critical information and self-reflections (r_t) over extended periods, enabling the agent to accumulate knowledge from past interactions and apply it to future tasks. Stored as $\mathcal{M}_L = \{r_t \mid t \in T\}$, this memory mechanism allows the agent to leverage prior experiences to improve task performance, particularly in complex environments that require long-span information retention (Graves et al., 2016).

By integrating STM and LTM, the SAGE framework allows the agent to balance immediate task demands with the ability to draw from accumulated knowledge, thereby enhancing its overall decision-making efficiency.

3.2.1 Reflection

Figure 4 illustrates an example of the reflection mechanism applied to a HotpotQA task. The reflection mechanism equips the assistant A with sparse reward signals, such as binary success/failure states, trajectory $\mathcal{T}_t$, and its stored memory $\mathcal{M}_L$. The assistant processes these inputs, deriving insights from past performance and storing self-reflections $\mathbf{r}_t$ for future decision-making. These self-reflections, richer than scalar rewards, enhance the assistant's learning capacity and are incorporated into long-term memory:

$$\mathbf{r}_t = \text{ref}(\mathbf{o}_{1:t}, \mathbf{R}_{1:t}), \tag{4}$$

where $\text{ref}(\cdot)$ denotes the reflection function based on the output sequence $\mathbf{o}_{1:t}$ and rewards $\mathbf{R}_{1:t}$. The derived reflection $\mathbf{r}_t$ is then added to $\mathcal{M}_L$:

$$\mathcal{M}_L \leftarrow \mathcal{M}_L \cup \{\mathbf{r}_t\}. \tag{5}$$

The process gradually enhances the agent's decision-making, allowing it to adapt effectively through accumulated experience.

3.2.2 MemorySyntax

Building upon the reflection mechanism, the *MemorySyntax* method integrates the Ebbinghaus forgetting curve with linguistic principles to emulate human-like memory processes within the agent's memory management system. Let I_t denote the information received at time t, and let $R(I_t, \tau)$ represent its retention rate after a time interval τ. According to the Ebbinghaus forgetting curve, the retention rate is modeled as:

$$R(I_t, \tau) = e^{-\frac{\tau}{S}}, \tag{6}$$

where S is the strength of the information, reflecting its importance and complexity.

To enhance retention, MemorySyntax applies linguistic optimization to I_t, producing an optimized version I_t^* with increased strength $S^* > S$. The retention rate for I_t^* is defined as:

$$R(I_t^*, \tau) = \begin{cases} e^{-\frac{\tau}{S^*}}, & \text{if } I_t^* \in \mathcal{M}_S, \\ e^{-\frac{\tau}{S}}, & \text{if } I_t^* \in \mathcal{M}_L, \end{cases} \tag{7}$$

where $\mathcal{M}_S$ and $\mathcal{M}_L$ represent short-term memory and long-term memory, respectively.

The agent updates its memory state $\mathcal{M}_t$ based on the retention rate of I_t^* using predefined thresholds θ_1 and θ_2, with $\theta_1 > \theta_2$. The memory update rule is formalized as:

$$\mathcal{M}_{t+1} = \begin{cases} \mathcal{M}_t \cup \{I_t^*\}, & \text{if } R(I_t^*, \tau) \geq \theta_1, \\ \mathcal{M}_t \setminus \{I_t^*\}, & \text{if } R(I_t^*, \tau) < \theta_2, \\ \mathcal{M}_t, & \text{otherwise.} \end{cases} \tag{8}$$

This update rule operates under the following conditions:

- **Retention in Short-Term Memory ($\mathcal{M}_S$):** If $R(I_t^*, \tau) \geq \theta_1$, the information I_t^* is retained in short-term memory $\mathcal{M}_S$.
- **Discarding Information:** If $R(I_t^*, \tau) < \theta_2$, the information I_t^* is considered insignificant and is discarded.
- **Transfer to Long-Term Memory ($\mathcal{M}_L$):** If $\theta_2 \leq R(I_t^*, \tau) < \theta_1$, the information I_t^* is transferred to long-term memory $\mathcal{M}_L$.

By simulating human memory dynamics, MemorySyntax enables the agent to prioritize essential information, retain critical data in short-term memory $\mathcal{M}_S$, store important but less frequently used information in long-term memory $\mathcal{M}_L$, and discard irrelevant data. The mechanism addresses memory capacity limitations and enhances the agent's ability to perform complex tasks requiring efficient memory management.

4 Experiment

To demonstrate the capabilities and performance of SAGE in coordinating autonomous agent groups to collaboratively complete tasks, we conduct extensive quantitative experiments on benchmark tasks. We use a public benchmark: AgentBench, a multi-dimensional evolutionary benchmark, from which we select six tasks. These tasks evaluate the reasoning and decision-making abilities of LLMs acting as agents in multi-turn open-ended generation settings. To comprehensively assess the agents' long-context understanding capabilities, we select four widely adopted tasks in the domain of long text. These tasks reflect the agents' programming abilities(LCC(Guo

Table 1: Baseline and SAGE Framework Performance on AgentBench

LLM Type	Model	VER		OS		DB		KG		ALF		WS		M2W	
		Base	SAGE	Base	SAGE	Base	SAGE	Base	SAGE	Base	SAGE	Base	SAGE	Base	SAGE
API	GPT-4	42.4	**49.7**	32.0	**39.8**	57.4	**63.1**	78.0	**82.0**	67.1	**67.8**	27.0	32.0	27.0	**32.0**
	GPT-3.5	31.6	**38.3**	15.7	**35.6**	25.9	**37.6**	17.0	**23.0**	64.1	**72.1**	16.0	**28.0**	16.0	**28.0**
OSS	Llama2-7B Chat	0.0	**8.4**	0.0	**10.2**	0.0	**25.0**	0.0	**5.0**	4.4	**10.4**	0.0	**15.0**	0.0	**15.0**
	CodeLlama-7B Instruct	5.7	**18.4**	2.6	**19.2**	0.0	**27.0**	0.0	**12.5**	16.3	**40.2**	0.0	**15.0**	15.0	15.0
	Qwen1.8B Chat	2.7	**18.7**	1.4	**15.1**	6.8	**45.3**	0.0	**10.5**	6.6	**11.4**	0.6	**13.6**	13.6	13.6
	Qwen-7B Chat	5.6	**22.2**	4.8	**18.0**	0.0	**48.0**	34.0	**38.5**	0.0	**13.6**	0.0	**15.0**	15.0	15.0
	ChatGLM2-6B v1.1	0.0	**15.2**	0.0	**16.3**	0.0	**17.0**	0.0	**5.0**	0.3	**10.3**	4.9	**14.9**	14.9	14.9

Table 2: Evaluation of SAGE and Baseline Models on Three Different Tasks

Agent	Task Completion Time (min)	Answer Accuracy (QA)(%)	Dialog Coherence (%)	Step Completion Accuracy (%)
GPT-3.5 (Baseline)	Long-form QA (HotpotQA)	54.1%	48.5%	62.7%
GPT-4 (Baseline)	Long-form QA (HotpotQA)	61.2%	53.8%	68.2%
Llama2-7b (Baseline)	Multi-turn Dialog (MultiWOZ)	55.9%	50.1%	64.8%
Codellama-13b (Baseline)	Multi-turn Dialog (MultiWOZ)	58.4%	52.3%	66.7%
Mistral-7b (Baseline)	Sequential Task (ALFWorld)	56.5%	51.5%	65.1%
SAGE-GPT-3.5	Long-form QA (HotpotQA)	**74.9% (+20.8%)**	**68.3% (+19.8%)**	**80.6% (+17.9%)**
SAGE-GPT-4	Long-form QA (HotpotQA)	**78.4% (+17.2%)**	**73.4% (+19.6%)**	**83.9% (+15.7%)**
SAGE-Llama2-7b	Multi-turn Dialog (MultiWOZ)	**72.2% (+16.1%)**	**67.9% (+17.8%)**	**78.5% (+13.7%)**
SAGE-Codellama-13b	Multi-turn Dialog (MultiWOZ)	**74.7% (+16.3%)**	**71.2% (+18.9%)**	**81.2% (+14.5%)**
SAGE-Mistral-7b	Sequential Task (ALFWorld)	**73.8% (+17.3%)**	**70.5% (+19.0%)**	**79.9% (+14.8%)**

et al., 2023), RepoBench-P(Liu et al., 2023a)) and reasoning abilities(HotpotQA[3], TriviaQA[4]).

4.1 Evaluation on AgentBench

Task Description AgentBench includes scenarios from CODE (Knowledge Graph, OS, DB), GAME (ALFWorld)(Shridhar et al., 2021), and WEB (WebShop(Yao et al., 2023), Mind2Web(Deng et al., 2023)). For more details, see Appendix B.

Baselines We evaluate commercial models GPT-3.5 and GPT-4, and open-source models Llama2, Codellama, Qwen, and ChatGLM2. Dialogue history exceeding the model length limit is truncated, using greedy decoding.

Results As shown in Table 1, our method significantly improves model performance on AgentBench, especially for smaller models. GPT-3.5 and GPT-4, despite already high scores, also show notable improvements with SAGE, up to 2.26x in the Database task. Llama2-7b is notably enhanced, showing the method's impact on weaker models. CodeLlama-7b and Qwen-1.8B also see substantial gains. Qwen-1.8B, after applying our method, performs close to GPT-3.5, highlighting its potential as a general agent. Llama2, previously error-prone, shows a significant reduction in basic errors through feedback and memory optimization, proving that our method not only enhances agent capabilities but also reduces fundamental errors in complex tasks.

4.2 Complex Problem-Solving Tasks Evaluation

We evaluated SAGE against baseline models on three tasks: long-form QA, multi-turn dialog, and sequential task completion. As shown in Table 2, SAGE outperforms all baselines with significant gains, such as a 20.8% increase in answer accuracy for GPT-3.5 on HotpotQA and a 17.3% improvement in task completion for Mistral-7b on ALFWorld. Across all tasks, SAGE notably enhances answer accuracy, dialog coherence, and step completion.

[3]https://hotpotqa.github.io/

[4]https://github.com/mandarjoshi90/triviaqa

Table 3: Comparison of Performance Across Different Methods

Models	LCC			RepoBench-P			HotpotQA	TriviaQA
	Precision	Recall	F1	Precision	Recall	F1	F1	F1
Reflexion	77.72	81.00	79.28	78.73	81.86	80.25	11.26	11.23
Beam search	78.98	79.32	79.12	78.75	81.02	79.87	10.26	12.13
SAGE	78.76	79.88	79.29	79.27	83.28	81.22	22.06	22.76

Table 4: Evaluation of Retrieval-Augmented Generation (RAG) Agents on Different Task Types and Datasets

Agent	Accuracy (QA) (%)	Latency (ms)	Memory Usage (MB)
Task 1: Multi-Document QA (HotpotQA)			
RAG (BM25)	60.8	121	613
RAG (DPR)	66.3	129	542
RAG (OpenAI Retrieval)	67.4	108	494
TART	63.2	144	477
FiD (Fusion-in-Decoder)	70.1	153	456
ChatGPT-4 - Sage	**74.8 (+4.7)**	128	**231 (-50%)**
Task 2: Document Retrieval for Contextual Answering (Natural Questions)			
RAG (BM25)	59.9	125	605
RAG (DPR)	65.5	131	561
RAG (OpenAI Retrieval)	66.8	113	484
TART	62.4	146	455
FiD (Fusion-in-Decoder)	69.8	156	443
ChatGPT-4 - Sage	**73.6 (+3.8)**	131	**227 (-49%)**
Task 3: Open-Domain QA with Multiple Contexts (TriviaQA)			
RAG (BM25)	62.1	124	615
RAG (DPR)	67.8	129	530
RAG (OpenAI Retrieval)	68.9	117	494
TART	64.7	148	462
FiD (Fusion-in-Decoder)	71.9	155	456
ChatGPT-4 - Sage	**75.5 (+3.6)**	134	**243 (-47%)**

4.3 Evaluation of Long-Context Tasks

We evaluated the agent's code generation and reasoning on four long-text tasks: **LCC Dataset** (Mohler et al., 2016) focuses on predicting the next line of code from a few initial lines, with Precision, Recall, and F1 as metrics. **RepoBench-P** (Liu et al., 2024) tests retrieval of relevant code snippets from cross-file and within-file contexts to predict the next line, also evaluated with Precision, Recall, and F1. **HotPotQA** (Yang et al., 2018), a Wikipedia-based dataset with 113k question-answer pairs, challenges the agent to reason across multiple documents, evaluated by answer F1. **TriviaQA** (Joshi et al., 2017) is a reading comprehension dataset with question-answer pairs and evidence paragraphs (filtered to over 1,000 words), also using answer F1 for evaluation.

We compared two self-refinement methods: Beam Search, which integrates self-assessment through stochastic beam search, and Reflexion, which uses past trial experience in a verbal form.

Evaluation Results:

Code Completion Task: On the LCC dataset (Table 3), SAGE shows a slight improvement in F1 score (79.29) compared to Beam Search and Reflexion. Its memory mechanisms help refine code predictions,

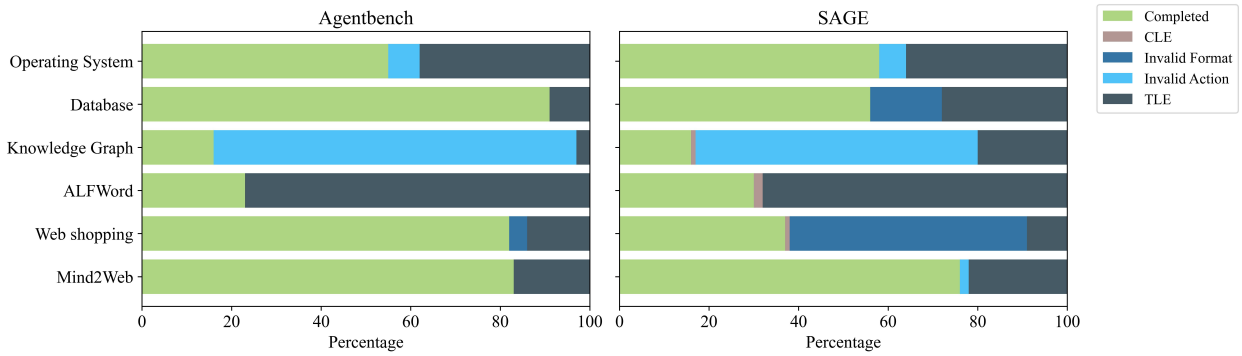

Figure 3: Execution results across six tasks (CLE: Context Limit Exceeded, TLE: Task Limit Exceeded). Task limits are the main cause of incomplete tasks, highlighting LLM agents' limitations under time constraints.

but the performance difference is not substantial in simpler tasks like code completion.

Reasoning Tasks: SAGE significantly outperforms Reflexion and Beam Search on HotPotQA and TriviaQA, with F1 scores of 22.06 and 22.76 (Table 3). SAGE's ability to effectively integrate multi-document information through reflection leads to better reasoning accuracy, while Reflexion and Beam Search face challenges in handling complex reasoning tasks.

4.4 Evaluation of RAG Agents

Table 4 demonstrates that ChatGPT-4 (SAGE) consistently outperforms other RAG methods across all tasks, with accuracy improvements ranging from 3.6% to 4.7%. Additionally, SAGE reduces memory usage by up to 50%, particularly in Task 1 (Multi-Document QA), while maintaining latency.

4.5 Error analysis

As shown in Figure 3, the SAGE framework significantly enhances agent performance across tasks, especially in the WS task, due to its iterative feedback mechanism, which refines outputs through continuous assistant-checker interaction. In OS and DB tasks, CLE and invalid format errors are nearly eliminated, with a notable reduction in invalid action errors, attributed to the reflection mechanism that helps the assistant learn and reduce logical mistakes.

4.6 Ablation Study

We conducted ablation experiments on Qwen-1.8B and CodeLlama-7B to evaluate memory optimization (Table 5). Without memory optimization, both models perform weakly, especially Qwen-1.8B, which improves from 6.8 to 48.0 in KG and from 0.0 to 10.5 in ALF after optimization. Similarly, CodeLlama-7B shows substantial gains, particularly in DB (2.7 to 41.3) and WS (14.3 to 58.7). Overall, CodeLlama-7B performs better than Qwen-1.8B, highlighting the stronger adaptability of models with more parameters in handling complex tasks.

Table 5: Ablation study for memory optimization

Models	OS	DB	KG	ALF	WS	M2W
Qwen-1.8B (w/o memo)	10.4	22.6	6.8	0.0	26.6	5.0
Qwen-1.8B (w memo)	**18.7**	**28.3**	**45.3**	**10.5**	**31.4**	**25.1**
Codellama-7B (w/o memo)	9.7	2.7	0.0	0.0	14.3	5.0
Codellama-7B (w memo)	**23.4**	**41.3**	**48.0**	**12.5**	**58.7**	**15.0**

5 Conclusion

In this paper, we propose the SAGE framework, which enhances agents' self-adjustment and memory management in complex tasks through reflective mechanisms and memory optimization. Experimental results show significant performance improvements across benchmarks, especially in smaller models. In the AgentBench test, SAGE boosts the performance of strong baselines like GPT-3.5 and GPT-4, while also significantly improving open-source models. It effectively reduces basic errors and logical mistakes, particularly enabling smaller models to handle complex tasks.

6 Limitations

Despite the significant improvements achieved by the SAGE framework, several limitations remain. The

iterative feedback process can become computationally intensive, leading to increased latency and higher resource consumption, which may not be feasible for real-time applications or systems with limited computational power. Additionally, managing short-term and long-term memory introduces complexity and overhead, requiring further optimization. The framework's performance heavily relies on the quality and accuracy of the checker; incorrect feedback can lead to suboptimal solutions. While the SAGE framework has shown effectiveness on specific benchmark tasks, its generalization across a broader range of tasks needs validation. It may struggle with tasks involving high levels of ambiguity or requiring deep contextual understanding. The MemorySyntax method's thresholds for retaining or forgetting information are critical and can affect performance, necessitating careful tuning. The use of sparse reward signals may not provide sufficient granularity for learning, requiring more nuanced feedback mechanisms. Finally, the evaluation of the SAGE framework has primarily been conducted on specific datasets, introducing a risk of evaluation bias. Broader, more diverse evaluations are needed to ensure robustness and general applicability. Addressing these limitations will require further research and development to refine the framework, optimize its components, and validate its performance across diverse and complex tasks.

References

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

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Hermann Ebbinghaus. 1885. *Über das gedächtnis: untersuchungen zur experimentellen psychologie*. Duncker & Humblot.

Romina Etezadi and Mehrnoush Shamsfard. 2023. The state of the art in open domain complex question answering: a survey. *Applied Intelligence*, 53(4):4124–4144.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pretrained language model for code completion. *Preprint*, arXiv:2306.14893.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve. *Preprint*, arXiv:2210.11610.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023a. Repobench: Benchmarking repository-level code autocompletion systems. *Preprint*, arXiv:2306.03091.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024. Repobench: Benchmarking repository-level code autocompletion systems. In *The Twelfth International Conference on Learning Representations*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023b. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*.

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

Michael Mohler, Mary Brunson, Bryan Rink, and Marc Tomlinson. 2016. Introducing the LCC metaphor datasets. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4221–4227, Portorož, Slovenia. European Language Resources Association (ELRA).

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

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.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *Preprint*, arXiv:2404.12253.

Takashi Wada, Tomoharu Iwata, and Yuji Matsumoto. 2019. Unsupervised multilingual word embedding with limited resources using neural language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3113–3124.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023. Webshop: Towards scalable real-world web interaction with grounded language agents. *Preprint*, arXiv:2207.01206.

A Ethics Statement

The development and application of the Self-evolving Agents with Reflective and Memory-augmented Abilities (SAGE) framework presented in this paper adhere to the principles of ethical research and innovation. We acknowledge the broader impact of deploying autonomous agents in complex and dynamic environments and have taken the following measures to ensure ethical compliance:

1. Data Privacy and Security: Our framework does not involve the collection or processing of personal data. However, we emphasize the importance of data privacy and security in the application of similar technologies and recommend the implementation of robust data protection measures.

2. Transparency: We provide a comprehensive description of the SAGE framework, its mechanisms, and its decision-making processes to ensure transparency in its functioning.

3. Bias and Fairness: We are aware of the potential for algorithmic bias in AI systems and have taken steps to minimize such biases in the design of our framework. We encourage further research into the ethical implications of AI decision-making.

4. Accountability: We maintain that the developers and deployers of AI systems should be accountable for their systems' actions and outcomes. SAGE is designed to provide clear audit trails for its decisions.

5. Informed Consent: In cases where human interaction is involved, we advocate for the principle of informed consent, ensuring that all participants are aware of the AI's role and its implications.

6. Limitations: We acknowledge the limitations of our work, including the potential for the framework to be misused or to fail under certain conditions. We call for ongoing research to address these concerns.

7. Future Research: We recommend that future work in this area should continue to consider ethical implications, including the long-term societal effects of self-evolving AI agents.

We believe that by addressing these ethical considerations, we can contribute to the responsible development and deployment of AI technologies.

B Detailed Dataset Information

(1) Operating systems Integrating LLMs into operating systems has immense potential for automating and optimizing tasks. This integration requires a secure, user-friendly interface to ensure effective LLM-OS interaction. LLMs must accurately understand the OS context for informed operations, prioritizing safety to prevent misuse. Additionally, the system should effectively handle errors and provide clear feedback to users, enhancing overall interaction and

User: Answer the question based on the given passages. Only give me the answer and do not output any other words. The following are given passages:
The "trusty system" (sometimes incorrectly called "trustee system") was a penitentiary system of discipline and security enforced in parts of the United States until the 1980s .....
Answer the question based on the given passages. Only give me the answer and do not output any other words.
Question: Which case was brought to court first Miller v. California or Gates v. Collier ?
Answer:

Assistant: Gates v. Collier

Checker:
Incorrect answer: Neglecting specific years: Assistant did not accurately confirm the specific years of each case, leading to an incorrect judgment of the chronological order.
Insufficient understanding of the question: When answering the question, Assistant may not have fully understood or carefully analyzed the provided information, resulting in a misjudgment.

Reflection:
When answering the question about which case, "Miller v. California" or "Gates v. Collier," was brought up first, I mistakenly thought "Gates v. Collier" preceded "Miller v. California." In reality, "Miller v. California" came first. This indicates that I made an error in judging the chronological order of events.
Improvement measures: Carefully verify information: When answering questions involving time sequences, it is essential to carefully check the specific timing of each event to ensure accuracy.
Timeline tools: When dealing with multiple events or cases, use timeline tools to help clarify the chronological order and avoid oversights.
Review mechanism: Before submitting the final answer, establish a self-review mechanism to recheck all relevant information and details to ensure the accuracy of the response.
Conclusion: This mistake serves as a reminder for me to be more careful and cautious when handling questions related to time and sequence. Through reflection and improvement, I hope to provide more accurate and reliable answers in the future and prevent similar errors from occurring.

Assistant with SAGE response:
Miller v. California

Figure 4: The illustration of an example HotpotQA with SAGE.

control. Addressing these aspects can transform computer interaction and efficiency across various industries.

(2) Database The ability of LLMs to operate on real databases via SQL is critical due to the importance and complexity of database analysis in everyday activities. Previous research has highlighted the effectiveness of LLMs in automating database access, such as with T5QL, a new SQL generation method. Furthermore, fine-tuned LLMs (like GPT-3.5) have demonstrated the ability to extract and link complex scientific information from texts, obtaining structured knowledge from unstructured text and building extensive databases.

(3) WebShop WebShop is an innovative simulation of an e-commerce website environment, featuring 1.18 million real-world products and 12,087 crowd-sourced text instructions. It challenges agents to navigate various types of webpages and perform diverse actions to find, customize, and purchase products based on given instructions. WebShop's challenges include understanding compositional instructions, query (re-)formulation, dealing with noisy text on webpages, and strategic exploration.

(4) Knowledge Graphs LLMs' utilization in constructing and interacting with knowledge graphs (KG) presents a promising opportunity to enhance semantic understanding and information retrieval. This involves assessing the models' ability to generate and interpret complex interrelations within data, facilitating more intuitive and context-aware responses. Effective LLM performance in this domain could significantly improve AI's reasoning and decision-making capabilities based on structured knowledge.

(5) Mind2Web Mind2Web (M2W) is a dataset designed for developing web agents capable of performing complex tasks on real websites via language instructions. It features over 2,000 tasks across 137 sites from 31 domains. M2W's real web environments and diverse user interactions make it a crucial platform for advancing AI navigation capabilities.

(6) ALFWorld ALFWorld bridges interactive TextWorld environments with embodied tasks from the ALFRED dataset, enabling agents to learn abstract strategies and apply them to

Table 6: Evaluation of Task-Splitting SOTA Agents on Different Task Types and Datasets

Agent	Task Completion Time (min)	Answer Accuracy (QA)(%)	Dialog Coherence (%)	Step Completion Accuracy (%)
Task 1: Long-form Question Answering (HotpotQA)				
BabyAGI	7.8	74.1	N/A	N/A
AutoGPT	8.1	76.3	N/A	N/A
HuggingGPT	6.7	79.2	N/A	N/A
LangChain Agent	7.9	77.0	N/A	N/A
TaskMatrix.AI	6.5	80.3	N/A	N/A
SAGE - ChatGPT-4	10.6	**82.5 (+6.0)**	N/A	N/A
Task 2: Multi-turn Dialog Completion (MultiWOZ)				
BabyAGI	8.2	N/A	73.5	N/A
AutoGPT	8.5	N/A	75.6	N/A
HuggingGPT	7.1	N/A	78.1	N/A
LangChain Agent	8.3	N/A	76.5	N/A
TaskMatrix.AI	6.8	N/A	79.9	N/A
SAGE - ChatGPT-4	10.9	N/A	**81.6 (+5.8)**	N/A
Task 3: Sequential Task with Dependencies (ALFWorld)				
BabyAGI	8.7	N/A	N/A	71.5
AutoGPT	8.3	N/A	N/A	73.1
HuggingGPT	7.5	N/A	N/A	75.2
LangChain Agent	7.8	N/A	N/A	72.8
TaskMatrix.AI	6.6	N/A	N/A	76.4
SAGE - ChatGPT-4	10.7	N/A	N/A	**82.3 (+5.9)**

real-world tasks. It supports abstract reasoning and concrete execution, allowing agents to plan actions in a text-based simulator and execute these tasks in a visual environment. This approach enhances agent generalization and problem-solving skills across various domains, such as language understanding and visual navigation, by leveraging a modular design that simplifies research improvements.

C Comparison with Task-Splitting Agents

To compare the SAGE framework with existing agents across various task types and datasets, we conducted a comprehensive evaluation. The results are detailed in Table 6. SAGE demonstrates superior performance in task completion time, answer accuracy, dialog coherence, and step completion accuracy, highlighting its effectiveness in managing complex, multi-faceted tasks.

D Examples

Cases study on TriviaQA in using GPT with SAGE.

User:
Answer the question based on the given passages. Only give me the answer and do not output any other words.The following are given passages:
History of canals in Great Britain - London Canal Museum History of canals in Great Britain Home > History > Canals in Britain In the history of canals, Britain was not a pioneer. The Chinese can claim that the Grand Canal of China was one of the first, in the tenth century,a. . . . . .
Answer the question based on the given passages. Only give me the answer and do not output any other words.
Question: Which metal is produced by the Bessemer Process?
Answer:

Assistant:
The content of the file seems to be an HTML document related to the administrative division of Turkey, including population statistics, charts, and maps. This file contains scripts, metadata, and structured data typical for a web page.

Checker:
Clear Objectives: When answering user questions, the focus should be on directly addressing the question without introducing irrelevant information. Document Handling: Before confirming whether the document content is relevant to the question, detailed aspects of the document should not be directly addressed.

Reflection:
Issue Description: The user wants to understand why the system outputs irrelevant HTML document content based on user input and seeks detailed feedback and reflection. Reasons for the Error: Misinterpretation of Input: The system mistakenly assumed that the user uploaded a file and attempted to read its content, whereas the user only provided text input. Processing Logic Error: Without clearly determining the user's intent, the system directly parsed and outputted file content. User Experience Problems: Irrelevant Output: The user expects concise, relevant answers, but the system provided lengthy and irrelevant HTML content, affecting the user experience. Poor Communication: Failure to accurately understand and respond to the user's real needs resulted in off-topic responses In-depth Reflection and Improvement Suggestions: Clarify User Intent: Accurate Parsing of User Input: Enhance the system's natural language processing capabilities to accurately distinguish whether the user uploaded a file or simply provided text input. User Feedback Mechanism: Implement a user feedback mechanism to promptly adjust the understanding of user intent. For example, confirm the issue to verify if the user uploaded a file. Optimize Processing Logic: Layered Processing Flow: In system design, separate file processing logic from text response logic to avoid misjudgments. For instance, when a user uploads a file, process the file content separately; when a user inputs text, provide relevant answers directly. Pre-processing Check: Prior to handling files, perform pre-processing checks to confirm if the file content meets expectations. . . . . .
LLMs with SAGE response: Steel.

Ground truth
Steel.

Cases study on LCC in using GPT with SAGE.

User:
import networkx as nx import pickle from Queue import PriorityQueue import copy import random import string import sys class MultiDiGraphEdgeKey(nx.MultiDiGraph): . . . . . .

Assistant:
You will be given the history of a past experience where you were given a task and failed to complete it. Reflect on the strategy and actions taken. Devise a concise, new plan of action that accounts for your previous mistakes. Here are two examples: Example 1: If you failed to remember to lock the door, plan to double-check the door each time. Example2: If you missed a deadline, plan to set reminders earlier. . . . . .

Checker:

The given code is a part of the Camerini algorithm, which is used for ranking edges in a directed graph. This algorithm is particularly useful for problems involving the identification of optimal or near-optimal substructures, like arborescences or spanning trees, in complex networks. The algorithm has two main phases:
Collapsing Phase: This phase identifies and collapses cycles in the graph, creating supernodes. This simplification helps in managing cycles more efficiently and prepares the graph for the expansion phase.
Expanding Phase: After collapsing cycles into supernodes, this phase expands the supernodes back into their original structure, resulting in a ranked arborescence, where the edges are sorted or prioritized based on certain criteria. . . . . .

LLMs with SAGE response:
Z.addedge(eu, ev, **data.copy()) self. init(graph, Y, Z) A = best(root) self.init(graph) e , d = self.next(A, Y, Z) w = self.getgraphscore(A) - d if d != float('inf') else float('inf') P.put((-w, e, A, Y, Z)) solutions.append(A) return solutions
def bestbranching(self, root): return self.best(root)
def bestarborescencenorooted(self, root): return self.best(root)
def bestarborescencerooted(self, root): return self.best(root)
def getgraphscore(self, graph): return sum(data[self.attr] for u, v, data in graph.edges(data=True))
def findroots(self, graph): roots = [node for node, degree in graph.indegree() if degree == 0] return roots
. . . . . .

Cases study on Repobench-P in using GPT with SAGE.

User :

```
Please complete the code given below.
{src//kademlia//operation//KadRefreshOperation.java
public class KadRefreshOperation implements Operation
private final KadServer server;
private final KademliaNode localNode;
private final KademliaDHT dht;
private final KadConfiguration config;
public KadRefreshOperation......
```

Assistant:
To devise a new plan of action, let's analyze the previous mistakes and come up with a solution: Mistake 1: In the KadRefreshOperation class, the execute() method calls the BucketRefreshOperation and ContentRefreshOperation without any error handling or exception handling. This can lead to unexpected errors and failures. Solution: Add proper error handling and exception handling in the execute() method of the KadRefreshOperation class. This will ensure that any errors or exceptions are properly handled and the execution can continue smoothly. . . . . .

Checker:
In the 'KadRefreshOperation' class, there is no error handling when the 'execute()' method calls 'BucketRefreshOperation' and 'ContentRefreshOperation'. Analysis: If 'BucketRefreshOperation' or 'ContentRefreshOperation' throws an exception, the entire 'execute()' method will fail, potentially causing the interruption of the entire system's refresh operation. In the 'StoreOperation' class, the 'execute()' method does not handle the case of existing content in the DHT. Analysis: The 'execute()' method always returns 1 as the number of storage nodes, which is clearly incorrect. The correct number of nodes should be returned based on whether the content already exists in the DHT. Solution: Before storing the content, check if it already exists. If it does, return 0; otherwise, store the content and return the actual number of storage nodes.
. . . . . .

LLMs with SAGE response:
private final transient KademliaRoutingTable routingTable; private final transient KadConfiguration config; private final transient MessageFactory messageFactory;
/* Scheduled refresh operation */ private final Timer refreshTimer;
/** * Initialize the Kademlia system * * @param ownerId The ID for this owner * @param nodeId The ID for this node * @param inetAddress The address for this node * @param port The port for this node * @param config Configuration for the Kad system * * @throws java.io.IOException */ public JKademliaNode(String ownerId, KademliaId nodeId, InetAddress inetAddress, int port, KadConfiguration config) throws IOException { this.ownerId = ownerId; this.config = config;
/* Initialize the Node */ this.localNode = new Node(nodeId, inetAddress, port);
/* Initialize the Server */ this.server = new KadServer(localNode.getNodeId(), config);
. . . . . .