COLA: Collaborative Multi-Agent Framework with Dynamic Task Scheduling for GUI Automation

Di Zhao[1], **Longhui Ma**[1], **Siwei Wang**[2*], **Miao Wang**[2*], **Zhao Lv**[2]
[1]College of Computer Science and Technology, National University of Defense Technology
[2]Academy of Military Sciences
{zhaodi, wangsiwei13}@nudt.edu.cn

Abstract

With the rapid advancements in Large Language Models (LLMs), an increasing number of studies have leveraged LLMs as the cognitive core of agents to address complex task decision-making challenges. Specially, recent research has demonstrated the potential of LLM-based agents on automating GUI operations. However, existing methodologies exhibit two critical challenges: (1) static agent architectures struggle to adapt to diverse GUI application scenarios, leading to inadequate scenario generalization; (2) the agent workflows lack fault tolerance mechanism, necessitating complete process re-execution for GUI agent decision error. To address these limitations, we introduce *COLA*, a collaborative multi-agent framework for automating GUI operations. In this framework, a scenario-aware agent Task Scheduler decomposes task requirements into atomic capability units, dynamically selects the optimal agent from a decision agent pool, effectively responds to the capability requirements of diverse scenarios. Furthermore, we develop an interactive backtracking mechanism that enables human to intervene to trigger state rollbacks for non-destructive process repair. Experiments on the GAIA dataset show that *COLA* achieves competitive performance among GUI Agent methods, with an average accuracy of 31.89%. On WindowsAgentArena, it performs particularly well in Web Browser (33.3%), Media & Video (33.3%), and Windows Utils (25.0%), suggesting the effectiveness of specialized agent design and dynamic strategy allocation. The code is available at https://github.com/Alokia/COLA-demo.

1 Introduction

Large Language Models (LLMs) have witnessed rapid advancements in recent years, demonstrating impressive capabilities in natural language understanding, dialogue, and general problem-solving tasks (Wang et al., 2024c; Guo et al., 2024). More complex multi-modal models (MLLMs), such as GPT-4v (Achiam et al., 2023), GPT-4o, and Gemini (Team et al., 2023), introduce a visual dimension, expanding the capabilities of LLMs and demonstrating outstanding capabilities across a broader range of fields (Guo et al., 2024). High-capacity LLMs and MLLMs often serve as the backbone of autonomous agents across a wide range of specialized fields, such as software development (Hong et al., 2024; Chan et al., 2024; Li et al., 2023), social simulation (Park et al., 2023; Gao et al., 2023), and gaming (Akata et al., 2024; Wang et al., 2023; Tan et al., 2024). Among these domains, personal computer automation—where agents interact with and control native applications via graphical user interfaces (GUIs)—has emerged as a particularly promising and challenging setting (Zhang et al., 2025a,b; Nguyen et al., 2024a).

Tasks on personal computers typically require multi-step sequential operations, where an operator must interact with a GUI to perform a series of coherent actions starting from an initial screen state until the given instructions are fully executed (Wang et al., 2024a). These workflows often involve dynamic UI elements, dependencies across steps, and contextual reasoning over temporal states. Despite the growing capabilities of MLLMs, current systems still face fundamental limitations in such GUI-based environments (Niu et al., 2024). Tasks such as recognizing UI components, interpreting layout semantics, and executing precise and context-aware actions remain challenging due to constrained screen perception, spatial reasoning, and historical context tracking (Zhang et al., 2024a; Wang et al., 2024d). To overcome these challenges, recent approaches have proposed agent-based architectures that augment MLLMs with specialized modules for visual understanding and action control (Song et al., 2024; Wang et al., 2024e; Nguyen et al., 2024b; Roucher, 2024).

*Corresponding authors

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 4571–4594
November 4-9, 2025

UFO (Zhang et al., 2024b) introduces a dual-agent system, utilizing an AppAgent to manage application operations and decision-making across various scenarios. Nevertheless, this approach struggles to handle more complex GAIA datasets (Mialon et al., 2024). Similarly, MMAC (Song et al., 2024) develops agents for four distinct tasks: programming, screen semantic recognition, video analysis, and general knowledge. However, the system design suffers from limited scalability and lacks flexibility. Any error in the execution process necessitates a complete restart, which can significantly hinder efficiency and adaptability in practical applications.

To overcome these limitations, we introduces *COLA*, a scalable and flexible collaborative multi-agent framework specifically designed for GUI task automation. *COLA* introduces a modular and scalable architecture composed of five specialized agent roles: **Planner**, **Task Scheduler**, **Decision Agent Pool**, **Executor**, and **Reviewer**. The decision agent pool comprises a collection of agents with focused domain-specific expertise, each meticulously tailored to address particular task categories such as web browsing, file system manipulation, software programming, and others. To ensure optimal agent selection for diverse scenarios, we employ a task scheduler capable of scenario-aware matching. Futhermore, *COLA* incorporates an interactive backtracking mechanism—inspired by the Swarm system[1]—which allows human users to revert an agent to a previous state, inject corrective guidance, and resume execution from that point. This human-in-the-loop design enables non-destructive recovery from anomalous behaviors and enhances system controllability. The entire process is illustrated in Figure 1.

Our summarized contributions are as follows:

- We propose *COLA*, a scalable and modular multi-agent framework for GUI-based task automation. *COLA* integrates hierarchical planning, dynamic agent selection, and fine-grained execution, supported by five specialized roles working in a cooperative loop.
- *COLA* features a dynamic pool of decision agents with distinct domain expertise. A dedicated task scheduler intelligently selects the most appropriate agent based on scenario awareness. Furthermore, we introduce an interactive backtracking mechanism enabling user intervention and non-destructive error recovery through state rollback and guided resumption.
- Experiments on both the GAIA and WindowsAgentArena benchmarks show that *COLA* performs competitively across diverse GUI-based tasks. Ablation studies further indicate that its dynamic task assignment strategy and memory-enhanced agent design contribute meaningfully to its effectiveness.

2 Related Work

2.1 LLM based Agents

In recent years, LLM-based agents have been considered as a promising approach to achieving artificial general intelligence (AGI) (Wang et al., 2024c). It significantly expands the capabilities of LLMs, empowering them to engage in planning, memorization and executing actions (Guo et al., 2024). Inspired by human-team collaboration, multi-agent systems are receiving increasing attention. For instance, ChatDev (Qian et al., 2024) orchestrates agents for end-to-end software development, while others leverage debate-style interactions to improve content quality (Tao et al., 2025; Chan et al., 2024; Subramaniam et al., 2024). MetaGPT (Hong et al., 2024) encodes Standard Operating Procedures (SOPs) into prompt sequences to standardize multi-agent workflows.

2.2 LLM-based UI Operation Agent

Recent work has explored the use of LLMs for GUI-based control and automation. GPT-4V-based agents have been applied to mobile interfaces via screenshot inputs (Yan et al., 2023), while MobileAgent (Wang et al., 2024b) integrates OCR to improve visual grounding. MobileAgent v2 (Wang et al., 2024a) introduces a multi-agent structure with distinct planning, decision, and reflection roles. In the desktop setting, UFO (Zhang et al., 2024b) inspects UI elements via pywinauto, assigning all decisions to a centralized AppAgent. Agent S (Agashe et al., 2025) proposed a memory-augmented hierarchical planning framework aimed at enhancing generalization via internal memory mechanisms. However, these systems typically rely on a single decision agent, limiting scalability and adaptability across diverse task scenarios. In contrast, our proposed framework *COLA* introduces a dynamic pool of specialized decision agents, each tailored for domain-specific tasks. A

[1] https://github.com/openai/swarm

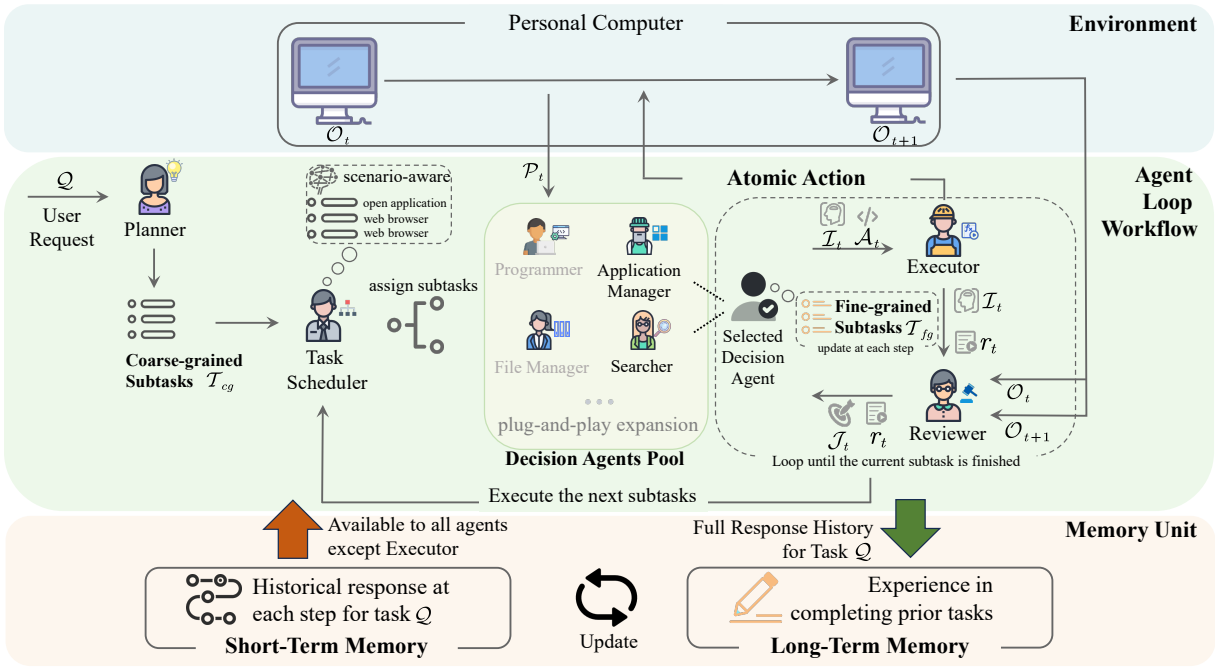

Figure 1: **An illustration of the *COLA* multi-agent framework**. In the first step, Planner takes request $\mathcal{Q}$ from user and decomposes it into a sequence of coarse-grained subtasks ($\mathcal{T}_{cg}$). Task Scheduler then dynamically selects optimal decision agents through scenario-aware matching. Selected Decision Agents subsequently perform hierarchical task refinement, utilizing their domain-specific expertise to decompose assigned subtasks into fine-grained subtasks ($\mathcal{T}_{fg}$), giving an atomic action $\mathcal{A}_t$ and an intention $\mathcal{I}_t$ to execute that action. Executor executes it and obtains the environmental feedback result r_t. Finally, the Reviewer evaluates the success of the action based on the environment $\mathcal{O}_t$, $\mathcal{O}_{t+1}$ before and after execution, the intention $\mathcal{I}_t$ and the result r_t. The judgment $\mathcal{J}_t$ is then sent back to the selected Decision Agent. This cyclic refinement continues until all subtask requirements are satisfied, with the Task Scheduler orchestrating inter-subtask transitions. Throughout the process, humans can intervene in the workflow at any time, providing guidance to correct the agent's response.

central task scheduler enables scenario-aware agent assignment, improving modularity and robustness in complex environments.

3 The COLA Framework

In this section, we will provide a detailed overview of the *COLA* architecture. The operation of *COLA* is sequential and iterative, and its process is depicted in Figure 1.

3.1 GUI Task Formulation

Computer operation tasks involve multi-step sequential processing. Given a computer environment and a user query $\mathcal{Q}$, a GUI Agent (denoted as ρ) receives the current observation $\mathcal{O}$ (e.g., a screenshot). Leveraging its internal planning and reasoning capabilities, the agent determines the next action $\mathcal{A}$ to take. This action is then executed, resulting in a change in the environment. The process can be formally defined as follows:

$$\mathcal{A}_t = \rho(\mathcal{Q}, \mathcal{O}_t, \mathcal{H}_{t-1}) \quad (1)$$

$$\mathcal{O}_{t+1} = \varphi(\mathcal{A}_t) \quad (2)$$

Here, $\mathcal{A}_t$ and $\mathcal{O}_t$ represent the action and observation at step t, respectively, and $\mathcal{H}_{t-1}$ denotes the history of operations up to step $t-1$. The function φ models the environment transition resulting from executing action $\mathcal{A}_t$, leading to the next observation $\mathcal{O}_{t+1}$. This process iterates until the user request $\mathcal{Q}$ is either successfully completed or deemed to have failed.

3.2 Visual Perception and Interaction

Visual Perception Identifying interactive elements within a screenshot poses a significant challenge (Bonatti et al., 2024; Niu et al., 2024). To address this, we utilize `pywinauto` (Bim and Minshuai, 2014) to inspect interactive UI elements within the application (Zhang et al., 2024b; Liu et al., 2025). Then we annotate the elements's bounding boxes on the screenshot to help MLLM

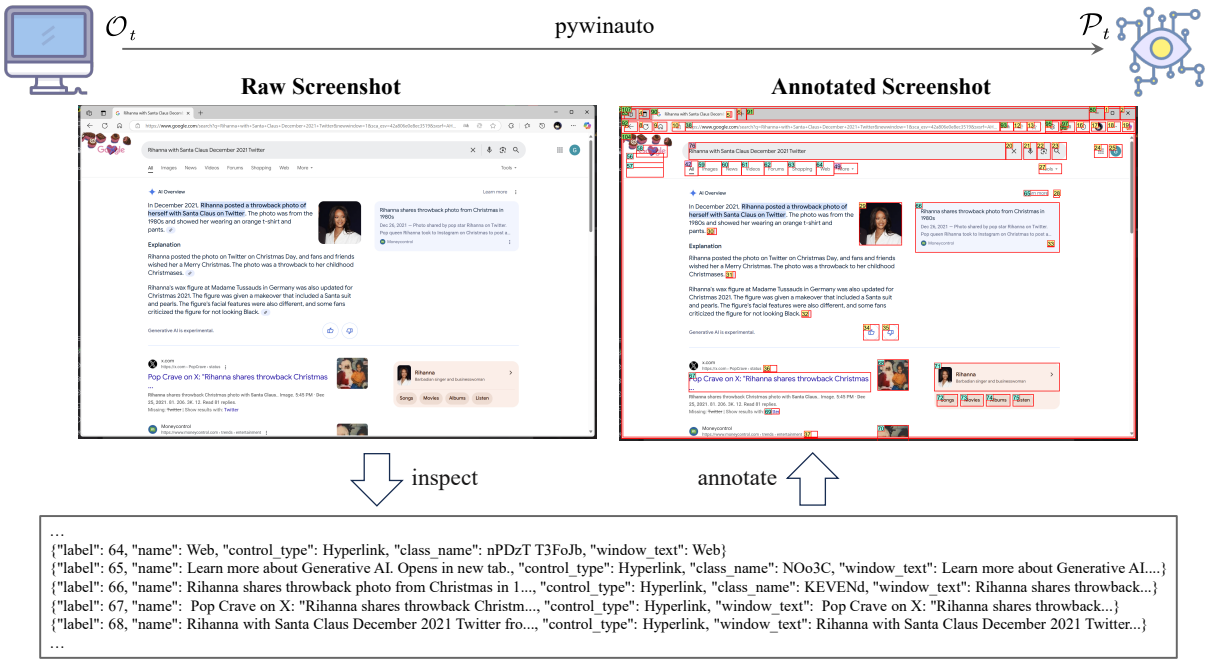

Figure 2: A visual perception example. The raw screenshot, annotated screenshot and interactive elements information make up the synthesized visual perception output $\mathcal{P}_t$.

understand the position and semantics of the elements (Zhang et al., 2024b). This process is formalized as:

$$\mathcal{P}_t = p(\mathcal{O}_t) \tag{3}$$

where p denotes the transformation function that extracts visual perception from the given observation $\mathcal{O}_t$, $\mathcal{P}_t$ represents the synthesized visual perception output at time step t, This output incorporates spatial and semantic information about interactive UI components, as illustrated in Figure 2.

Visual Interaction To interact with the computer environment, we developed eight actions, as detailed in Appendix B. We design a domain mechanism for each action so that only agents registered in the domain can use the action. This design paradigm effectively manages the agent's capabilities while minimizing the complexity associated with expanding the action space. Users can custom actions to meet their specific requirements and configure the domain in which agent can recognize and apply them.

3.3 Memory Unit For Self-Evolution

The operational history $\mathcal{H}_{t-1}$, referenced in the Eq. 1, encapsulates the agent's accumulated experience and is structured into two key components: long-term memory and short-term memory. This memory architecture is vital for LLM-based agents (Wang et al., 2024a; Zhang et al., 2024c; Agashe et al., 2025) to learn from past interactions and effectively track progress in complex, multi-step computer operation tasks.

Long-Term Memory The long-term memory (denoted as $\mathcal{M}_L$) archives complete prior task executions. For a given query $\mathcal{Q}$, a retrieval function ϕ identifies the top-n most relevant records $\mathcal{M}_L^n = \phi(\mathcal{Q}, n)$ from $\mathcal{M}_L$. This retrieval is based on the cosine similarity between an embedding of the current query and embeddings of pre-generated summaries for each record in $\mathcal{M}_L$.

Short-Term Memory The short-term memory (denoted as $\mathcal{M}_{S_t} = \{st_1, st_2, ..., st_t\}$) retains the sequence of responses generated by the agent during the current task. To manage computational costs and maintain focus, only the m most recent responses, $\mathcal{M}_{S_t}^m = \{st_{t-m+1}, st_{t-m+2}, ..., st_t\}$, are typically utilized, providing immediate contextual history.

At each step t, the operational history $\mathcal{H}_{t-1} = (\mathcal{M}_L^n, \mathcal{M}_{S_{t-1}}^m)$ is integrated into the agent's prompt to inform its decision-making.

3.4 Hierarchical Multi-Agent Framework

In the *COLA* framework, we establish a hierarchical multi-agent system comprising five distinct agent

types: the Planner, Task Scheduler, Decision Agent Pool, Executor, and Reviewer. Central to this architecture is the Decision Agent Pool, which comprises a scalable ensemble of agents, each endowed with specialized skills.

3.4.1 Planner

As in most studies (Wang et al., 2024a; Liu et al., 2025; Song et al., 2024; Wang et al., 2025), first, we utilize a Planner agent to decompose a user request $\mathcal{Q}$ into a set of coarse-grained subtasks $\mathcal{T}_{cg} = \{s_1, s_2, \ldots, s_k\}$. This decomposition establishes an organized foundation for execution, formalized as

$$\mathcal{T}_{cg} = \mathcal{L}_{PL}(\mathcal{Q}, \mathcal{H}_{t-1}) \tag{4}$$

where $\mathcal{L}_{PL}$ is the Planner's LLM backbone. These coarse-grained subtasks are subsequently processed and refined into fine-grained actions by specialized agents within the Decision Agent Pool, illustrating the framework's hierarchical approach to task management.

3.4.2 Task Scheduler

Assigning coarse-grained subtasks $\mathcal{T}_{cg}$ to decision agents within a Decision Agent Pool is a complex problem, as traditional machine learning models require labeled data for supervised training and struggle with the pool's dynamic scalability. Given that LLMs have demonstrated excellent performance in logical reasoning (Kojima et al., 2022; Liu et al., 2023b), common sense knowledge (Zhao et al., 2023; Borro et al., 2025), and impressive generalization capabilities (Budnikov et al., 2025; Qi et al., 2025), we leverage LLMs as backbone model to manage this subtask allocation. Then, the Task Scheduler was developed as a pivotal component that orchestrates the allocation of tasks to specialized agents. Upon receiving $\mathcal{T}_{cg}$ from the Planner, the Task Scheduler analyzes the capabilities needed for each subtask and matches them against DA_{desc}, which is a set of natural language descriptions detailing the unique expertise of each available decision agent in the Decision Agent Pool. This ensures each subtask is assigned to the most suitable agent, a process formally represented as:

$$\begin{aligned} \mathbf{D} &= \{(\mathcal{R}_1, rt_1), (\mathcal{R}_2, rt_2), ..., (\mathcal{R}_k, rt_k)\} \\ &= \mathcal{L}_{TS}(\mathcal{Q}, \mathcal{T}_{cg}, DA_{desc}, \mathcal{H}_{t-1}) \end{aligned} \tag{5}$$

Here, $\mathcal{L}_{TS}$ represents the LLM of the Task Scheduler, $\mathcal{R}_k$ is the decision agent in the Decision Agent Pool, , rt_k denotes the subtask assigned to $\mathcal{R}_k$, and $\mathcal{T}_{cg} = rt_1 \cup rt_2 \cup \cdots \cup rt_k$, $\mathbf{D}$ is the set of task assignments. Following this assignment, the designated agents $\mathcal{R}_k$ sequentially address their respective subtasks rt_k, enabling a dynamic and expertise-driven workflow.

3.4.3 Decision Agent Pool

The Decision Agent Pool operationalizes the core principle of leveraging specialized expertise through a flexible, Mixture of Experts (MoE)-inspired architecture (Jacobs et al., 1991). This design contrasts with static agent configurations that often struggle with the diverse and dynamic demands of complex computer operation tasks (Zhang et al., 2024b; Song et al., 2024; Wang et al., 2024a). Our pool consists of multiple, distinct agents, each characterized by a natural language description of its unique skills. The set of descriptions for all agents is denoted as DA_{desc} and is described in Appendix C. A key element of the *COLA* framework's hierarchical structure is actualized within this pool: when an agent $\mathcal{R}_k$ (selected by the Task Scheduler) receives its assigned coarse-grained subtask rt_k, it autonomously refines this into a sequence of fine-grained subtasks $\mathcal{T}_{fg}$. In devising these fine-grained steps, the agent considers the original user query $\mathcal{Q}$, its specific coarse-grained subtask rt_k, the current visual perception $\mathcal{P}_t$ (as detailed in Section 3.2), any relevant judgment $\mathcal{J}_{t-1}$ from the Reviewer, and its memory components $\mathcal{H}_{t-1}$. The agent then determines the immediate action $\mathcal{A}_t$ to be performed and its underlying intent $\mathcal{I}_t$. This decision process is formally represented as:

$$\mathcal{I}_t, \mathcal{A}_t, \mathcal{T}_{fg} = \mathcal{L}_{\mathcal{R}_k}(\mathcal{Q}, rt_k, \mathcal{P}_t, \mathcal{J}_{t-1}, \mathcal{H}_{t-1}) \tag{6}$$

The regeneration of $\mathcal{T}_{fg}$ for each execution step allows for dynamic adjustments to the plan. This 'task scheduler to specialized agent' assignment, combined with the agents' refinement capabilities, enables a plug-and-play architecture. Users can introduce new, custom-developed agents and define their operational domains using the mechanism described in Section 3.2, thereby scaling the system's overall capabilities.

3.4.4 Executor

The Executor interfaces directly with the computer environment by carrying out the specific action $\mathcal{A}_t$ determined by a decision agent. It operates

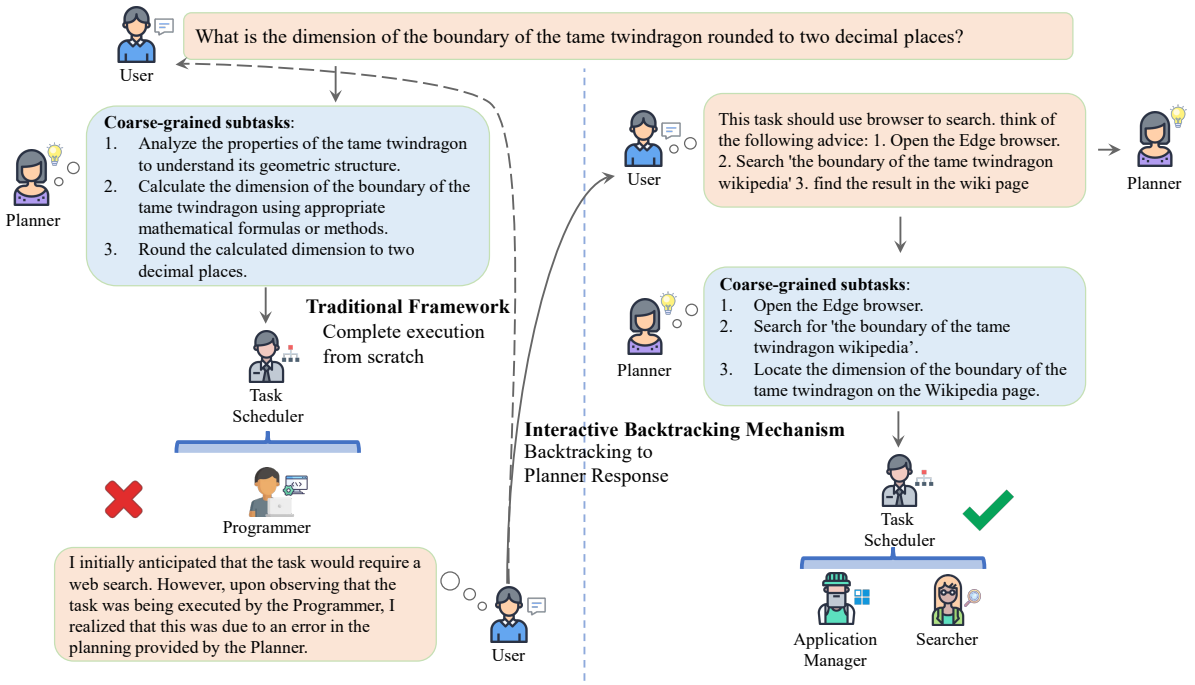

Figure 3: A comparison between the traditional agent framework and *COLA* reveals key differences.

without an internal memory unit, focusing solely on execution. Its function is formalized as:

$$\mathcal{O}_{t+1}, r_t = \varphi(\mathcal{A}_t) \tag{7}$$

where $\varphi(\cdot)$ denotes the function by which the executor performs the action, r_t is the result of the action, which may be null, as in the case of actions like a mouse click.

3.4.5 Reviewer

Consistent with prior research (Wang et al., 2024a; Song et al., 2024; Liu et al., 2025), a Reviewer agent is employed to mitigate potential LLM hallucinations (Liu et al., 2023a; Gunjal et al., 2023; Cui et al., 2023) by evaluating and correcting the decision agent's choices. The Reviewer analyzes the decision agent's intended $\mathcal{I}_t$ to perform action $\mathcal{A}_t$, the result of execution result r_t, and the environmental transition $\mathcal{O}_t \rightarrow \mathcal{O}_{t+1}$, to determine the action's appropriateness and successful execution. This judgement process is formalized as:

$$\mathcal{J}_t = \mathcal{L}_{RE}(\mathcal{I}_t, \mathcal{A}_t, r_t, \mathcal{O}_t, \mathcal{O}_{t+1}, \mathcal{H}_{t-1})$$

where $\mathcal{L}_{RE}$ is the LLM of Reviewer. This judgment $\mathcal{J}_t$ is then fed back into the decision agent to inform and refine subsequent decision-making steps.

3.5 Interactive Backtracking Mechanism

To enhance the robustness and flexibility of human-AI collaborative workflows, we introduce an *Interactive Backtracking Mechanism* designed to facilitate non-destructive process correction and adaptive control. This mechanism comprises two core functionalities: (1) **Role Switching**: This feature empowers users to dynamically alter the active dialog agent during an ongoing interaction. (2) **Dialog Backtracking**: This capability enables users to revert the system to a previous conversational state, thereby allowing the workflow to be re-executed from a chosen point. This non-linear navigation supports iterative refinement and correction without necessitating a complete restart of the process.

To accommodate varying degrees of human involvement and oversight, the system supports three distinct interaction modes: (1) **Automatic Mode**: The workflow proceeds autonomously without human intervention. (2) **Passive Mode**: The system operates independently under normal conditions but is programmed to solicit human assistance when encountering ambiguities or errors. (3) **Active Mode**: The workflow is designed to pause at each decision point, requiring explicit human input to proceed.

We compared the traditional agent framework with *COLA*, as shown in Figure 3. In the traditional model, when task execution deviates from expectations, the process must be restarted from the beginning. In contrast, *COLA*'s interactive backtracking mechanism allows for flexible state backtracking, enabling non-destructive repairs without restarting

Agent Pipeline	Backbone	Level 1	Level 2	Level 3	Avg.	APIs
Magentic-1 (Fourney et al., 2024)	GPT-4o	46.24	28.30	18.37	32.23	✓
HF Agents (Roucher, 2024)	GPT-4o	49.46	28.30	**18.37**	33.22	✓
Sibyl (Wang et al., 2024e)	GPT-4o	47.31	32.70	16.33	34.55	✓
DynaSaur (Nguyen et al., 2024b)	GPT-4o	**51.61**	**36.48**	18.37	**38.21**	✓
No Pipeline (Nguyen et al., 2024b)	GPT-4o	13.98	8.81	2.04	9.30	✗
FRIDAY (Wu et al., 2024b)	GPT-4o	40.86	20.13	6.12	24.25	-
MMAC (Song et al., 2024)	GPT-4o	45.16	20.75	6.12	25.91	✗
COLA	GPT-4o	**49.46**	**27.67**	**12.24**	**31.89**	✗

Table 1: Performance comparison between our model and multiple baseline models on the GAIA benchmark. "No Pipeline" refers to the raw GPT-4o, with no agent pipeline applied. APIs represents the way to browse the web, "✓" indicates that the web is accessed through an API, such as AutoGen web browser tool (Wu et al., 2024a) , "✗" means navigating web pages by simulating human interaction with the browser.

Method	Backbone	Office	Web Browser	Windows System	Coding	Media & Video	Windows Utils	Avg.
UFO (Zhang et al., 2024b)	GPT-4o	0.0	23.3	33.3	29.2	33.3	8.3	21.3
NAVI (Bonatti et al., 2024)	GPT-4o	0.0	20.0	29.2	9.1	25.3	0.0	13.9
Agent S (Agashe et al., 2025)	GPT-4o	0.0	13.3	**45.8**	29.2	19.1	22.2	21.6
UFO^2 (Zhang et al., 2025c)	GPT-4o	**4.7**	30.0	41.7	**58.3**	33.3	8.3	**29.4**
COLA	GPT-4o	0.0	**33.3**	41.7	33.3	**33.3**	**25.0**	27.8

Table 2: The success rate breakdown by application type on WindowsAgentArena benchmark.

the entire process.

4 Experiment

Benchmark We evaluate *COLA* using the GAIA dataset (Mialon et al., 2024), a benchmark dedicated to evaluating general AI assistants. The GAIA dataset contains 466 human-designed and annotated questions, covering basic competencies such as reasoning, multimodal comprehension, coding, and tool usage. In addition, we also evaluate the scalability of COLA on WindowsAgentArena (Bonatti et al., 2024), a benchmark that contains 154 benchmarks covering office, web browser, windows system, coding, etc.

Baselines For the GAIA dataset, API Agent methods including Magentic-1 (Fourney et al., 2024), Hugging Face Agents (HF Agents) (Roucher, 2024), Sibyl System v0.2 (Sibyl) (Wang et al., 2024e), and DynaSaur (Nguyen et al., 2024b) were selected, alongside GUI Agent methods such as No Pipeline, FRIDAY (Wu et al., 2024b), and MMAC (Song et al., 2024). API agent refers to accessing the web through an api, and GUI agent refers to accessing the web by interacting with GUI. For the WindowsAgentArena dataset, NAVI (Bonatti et al., 2024), UFO (Zhang et al., 2024b), Agent S (Agashe et al., 2025), and UFO^2 (Zhang et al., 2025c) were chosen for comparison.

Implementation Details We use OpenAI's text-embedding-3-large as embedding model. For decision agent, the long-term memory parameter n is set to 2, and the short-term memory parameter m is set to 6. For other agents, n is set to 3, and m to 10. All agentic pipelines utilize GPT-4o, with the maximum number of reasoning steps limited to 20. The interaction mode is set to Automatic. For the GAIA dataset, tasks were evaluated in increasing order of difficulty, from Level 1 (simple) to Level 3 (complex), enabling the model to first acquire long-term memory on simpler tasks. The long-term memory thus obtained from GAIA was then leveraged for testing on the WindowsAgentArena dataset.

4.1 Main Result

We evaluate our proposed method and compare it with several baseline approaches in Table 1. As shown, *COLA* outperforms other methods for simulating human web browsing on the GAIA private test set, particularly in the more challenging Level 2 and Level 3 tasks. The substantial improvement in accuracy over No Pipeline - from 13.98% to 49.46% on level 1, from 8.81% to 27.67% on level 2, and from 2.04% to 12.24% on level 3 - underscores the effectiveness of *COLA*. Compared to API Agent methods offering greater reliability and efficiency, COLA exhibits competitive perfor-

mance results, thereby demonstrating the potential of GUI Agents.

According to the results presented in Table 2 of the WindowsAgentArena benchmark, *COLA* achieved state-of-the-art performance in the Web Browser, Media & Video, and Window Utils categories, while ranking second in the Windows System and Coding tasks. Notably, *COLA* outperformed UFO by a margin of 10.0% in the Web Browser task and 16.7% in Window Utils, underscoring its enhanced effectiveness.

The comparison against several baseline methods on two benchmarks demonstrates that designing specialized agents for different task types, combined with a dynamic task allocation strategy, can effectively improve task completion rates.

4.2 Ablation Study

Configuration	Level 1	Level 2	Level 3	Avg.
COLA	49.46	27.67	12.24	31.89
w/o TS & DA	43.01	18.24	2.04	23.26

Table 3: Ablation study performance comparison results on the GAIA test set. w/o TS & DA means Task Scheduler and Decision Agent Pool are not included.

Dynamic assignment of tasks to specialized agents improves performance We conduct ablation studies more deeply on the GAIA test set in order to investigate the contribution of the Decision Agent Pool and the Task Scheduler in the *COLA* framework. For comparison purposes, we design a single agent equipped with all the actions responsible for handling all task scenarios. Then remove the Task Scheduler, there is no need for task assignment at this point. We obtained the results shown in Table 3. The decrease in the overall average score from 31.89% to 23.26% highlights the importance of the Decision Agents pool. While the difference in Level 1 scores (49.46% vs. 43.01%) is minimal, there is a significant gap in Level 2 (27.67% vs. 18.24%) and Level 3 scores (12.24% vs. 2.04%), indicating that task specialization by scenario is effective.

Memory units can improve agent's reasoning skills On the GAIA validation set, we performed ablation studies on the memory unit, targeting long-term memory $\mathcal{M}_L$, short-term memory $\mathcal{M}_{S_t}$, and the entire unit, with results in Table 4. Removing $\mathcal{M}_L$ decreased performance across all three Levels, showing agents learn from past tasks via long-term memory. Removing $\mathcal{M}_{S_t}$ resulted in a minor 1.88% drop for Level 1, but 11.63% and 19.24% drops for Levels 2 and 3 respectively. We attribute this to Level 1 tasks requiring few steps with less hallucination, whereas Levels 2 and 3 involve more steps, exacerbating model hallucination and failure.

Configuration	Level 1	Level 2	Level 3	Avg.
COLA	50.94	36.05	23.08	36.69
w/o $\mathcal{M}_L$	43.40	27.91	15.38	28.90
w/o $\mathcal{M}_{S_t}$	49.06	24.42	3.84	25.77
w/o $\mathcal{M}_L$ & $\mathcal{M}_{S_t}$	30.19	15.12	0.0	15.10

Table 4: Ablation experiments on memory mechanisms on GAIA validation set.

Error Type	Level 1	Level 2	Level 3	Avg.
Planning	23.08	30.91	30.00	28.00
Allocation	0.00	7.27	5.00	4.09
Decision	76.92	61.82	65.00	67.91

Table 5: The statistic of error rate(%) on GAIA validation set that COLA failed to complete.

4.3 Error Analysis

We conducted a detailed error analysis on the validation set of GAIA to identify the primary sources of failure in the *COLA* framework. Errors were categorized into three types: (1) Planning errors, where the Planner generates incorrect high-level task decompositions; (2) Allocation errors, arising when the Task Scheduler assigns subtasks to inappropriate decision agents; and (3) Decision errors, where the selected decision agent produces an incorrect action. As shown in Table 5, the majority of failures (67.91%) were attributed to decision errors, while allocation errors accounted for only 4.09%. These findings indicate that the LLM-based task allocation strategy employed by the Task Scheduler is generally effective, and that improving the reasoning capabilities of individual decision agents remains a key area for future enhancement.

4.4 Case Study

We present real case studies to illustrate the *COLA* workflow process, as detailed in Appendix E.

5 Conclusion

We present *COLA*, a modular and scalable multi-agent framework for GUI-based task automation. By leveraging a pool of specialized decision agents, a dynamic task scheduler, and an interactive back-

tracking mechanism, *COLA* effectively handles diverse and complex tasks. Experiments on GAIA and WindowsAgentArena benchmarks demonstrate consistent improvements over existing methods. Ablation studies further highlight the benefits of task specialization and memory mechanisms. Our results suggest that collaborative agent designs offer a promising direction for robust and generalizable UI-based AI assistants.

6 Limitations

Lack of explanation for subtask allocation process We have implemented a Task Scheduler designed to allocate subtasks to appropriate decision agents, utilizing a Large Language Model (LLM) as its intelligent core. Specifically, through carefully crafted prompts, the LLM assesses descriptions of both the subtasks and the decision agents to determine the most suitable assignment. Due to the current opaqueness and limited interpretability of LLMs, quantitatively measuring the performance of this subtask distribution process presents significant challenges. However, our internal error analysis indicates that leveraging the LLM as the assignment model is proving to be an effective approach.

pywinauto limits the range of applications While COLA has achieved good performance, its reliance on pywinauto to obtain precise locations and information for interactive controls limits its broader applicability, as pywinauto is not compatible with all applications. To expand COLA's capabilities, future research will further incorporate OCR technology to identify interactive controls and text.

Creating a decision agent for all scenes is labor intensive The operation system's environment is complex, and manually designing decision agents for various scenarios is labor-intensive. We hope that future studies will support the automation of constructing scenario-specific agents, such as creating expert agents automatically based on software user guides, enabling *COLA* to handle an expanded range of tasks.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Project 62406329, Project 62476280.

References

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

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s: An open agentic framework that uses computers like a human. In The Thirteenth International Conference on Learning Representations.

Elif Akata, Lion Schulz, Julian Coda-Forno, Seong Joon Oh, Matthias Bethge, and Eric Schulz. 2024. Playing repeated games with large language models.

HE Bim and WANG Min-shuai. 2014. Application of pywinauto in software performance test. Computer and Modernization, (8):135.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. 2024. Windows agent arena: Evaluating multi-modal os agents at scale. Preprint, arXiv:2409.08264.

Andrey Borro, Patricia J Riddle, Michael W Barley, and Michael J Witbrock. 2025. Large language models as common-sense heuristics. Preprint, arXiv:2501.18816.

Mikhail Budnikov, Anna Bykova, and Ivan P Yamshchikov. 2025. Generalization potential of large language models. Neural Computing and Applications, 37(4):1973–1997.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better LLM-based evaluators through multi-agent debate. In The Twelfth International Conference on Learning Representations.

Chenhang Cui, Yiyang Zhou, Xinyu Yang, Shirley Wu, Linjun Zhang, James Zou, and Huaxiu Yao. 2023. Holistic analysis of hallucination in gpt-4v(ision): Bias and interference challenges. ArXiv, abs/2311.03287.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed M. Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. 2024. Magentic-one: A generalist multi-agent system for solving complex tasks. ArXiv, abs/2411.04468.

Chen Gao, Xiaochong Lan, Zhi jie Lu, Jinzhu Mao, Jing Piao, Huandong Wang, Depeng Jin, and Yong Li. 2023. S3: Social-network simulation system with large language model-empowered agents. ArXiv, abs/2307.14984.

Anish Gunjal, Jihan Yin, and Erhan Bas. 2023. Detecting and preventing hallucinations in large vision language models. In AAAI Conference on Artificial Intelligence.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: a survey of progress and challenges. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI '24.

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

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. Neural Computation, 3(1):79–87.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In Advances in Neural Information Processing Systems.

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 Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Fuxiao Liu, Kevin Lin, Linjie Li, Jianfeng Wang, Yaser Yacoob, and Lijuan Wang. 2023a. Aligning large multi-modal model with robust instruction tuning. ArXiv, abs/2306.14565.

Hanmeng Liu, Zhiyang Teng, Ruoxi Ning, Jian Liu, Qiji Zhou, and Yue Zhang. 2023b. Glore: Evaluating logical reasoning of large language models. CoRR, abs/2310.09107.

Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, and Fei Huang. 2025. PC-agent: A hierarchical agentic framework for complex task automation on PC. In Workshop on Reasoning and Planning for Large Language Models.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for general AI assistants. In The Twelfth International Conference on Learning Representations.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Md. Mehrab Tanjim, Nesreen K. Ahmed, Puneet Mathur, Seunghyun Yoon, Lina Yao, Branislav Kveton, Thien Huu Nguyen, Trung Bui, Tianyi Zhou, Ryan A. Rossi, and Franck Dernoncourt. 2024a. Gui agents: A survey. CoRR, abs/2412.13501.

Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, Franck Dernoncourt, and Tianyi Zhou. 2024b. Dynasaur: Large language agents beyond predefined actions. ArXiv, abs/2411.01747.

Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. 2024. Screenagent: A vision language model-driven computer control agent. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24, pages 6433–6441. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In Proceedings of the 36th annual acm symposium on user interface software and technology, pages 1–22.

Zhenting Qi, Hongyin Luo, Xuliang Huang, Zhuokai Zhao, Yibo Jiang, Xiangjun Fan, Himabindu Lakkaraju, and James R. Glass. 2025. Quantifying generalization complexity for large language models. In The Thirteenth International Conference on Learning Representations.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. pages 15174–15186, Bangkok, Thailand.

Aymeric Roucher. 2024. Huggingface agent.

Zirui Song, Yaohang Li, Meng Fang, Zhenhao Chen, Zecheng Shi, Yuan Huang, and Ling Chen. 2024. Mmac-copilot: Multi-modal agent collaboration operating system copilot. ArXiv, abs/2404.18074.

Vighnesh Subramaniam, Antonio Torralba, and Shuang Li. 2024. DebateGPT: Fine-tuning large language models with multi-agent debate supervision.

Weihao Tan, Ziluo Ding, Wentao Zhang, Boyu Li, Bohan Zhou, Junpeng Yue, Haochong Xia, Jiechuan Jiang, Longtao Zheng, Xinrun Xu, Yifei Bi, Pengjie Gu, Xinrun Wang, Börje F. Karlsson, Bo An, and Zongqing Lu. 2024. Towards general computer control: A multimodal agent for red dead redemption II as a case study. In ICLR 2024 Workshop on Large Language Model (LLM) Agents.

Mingxu Tao, Dongyan Zhao, and Yansong Feng. 2025. Chain-of-discussion: A multi-model framework for

complex evidence-based question answering. In Proceedings of the 31st International Conference on Computational Linguistics, pages 11070–11085, Abu Dhabi, UAE. Association for Computational Linguistics.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi (Jim) Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. Trans. Mach. Learn. Res., 2024.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. In The Thirty-eighth Annual Conference on Neural Information Processing Systems.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024b. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. In ICLR 2024 Workshop on Large Language Model (LLM) Agents.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024c. A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6):186345.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. 2024d. Gui agents with foundation models: A comprehensive survey. ArXiv, abs/2411.04890.

Yulong Wang, Tianhao Shen, Lifeng Liu, and Jian Xie. 2024e. Sibyl: Simple yet effective agent framework for complex real-world reasoning. ArXiv, abs/2407.10718.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. CoRR, abs/2501.11733.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024a. Autogen: Enabling next-gen LLM applications via multi-agent conversation.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024b. Os-copilot: Towards generalist computer agents with self-improvement. ArXiv, abs/2402.07456.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Qinghong Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian J. McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. ArXiv, abs/2311.07562.

Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, and Dongmei Zhang. 2025a. Api agents vs. gui agents: Divergence and convergence. Preprint, arXiv:2503.11069.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Ming-Jie Ma, Qingwei Lin, S. Rajmohan, Dongmei Zhang, and Qi Zhang. 2024a. Large language model-brained gui agents: A survey. ArXiv, abs/2411.18279.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2025b. Large language model-brained gui agents: A survey. Preprint, arXiv:2411.18279.

Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, Liqun Li, Yu Kang, Zhao Jiang, Suzhen Zheng, Rujia Wang, Jiaxu Qian, Minghua Ma, Jian-Guang Lou, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2025c. Ufo2: The desktop agentos. Preprint, arXiv:2504.14603.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024b. Ufo: A ui-focused agent for windows os interaction. CoRR, abs/2402.07939.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024c. A survey on the memory mechanism of large language model based agents. CoRR, abs/2404.13501.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. In Advances in Neural Information Processing Systems, volume 36, pages 31967–31987. Curran Associates, Inc.

A Notations

For clarity and ease of reference, the commonly used notations throughout the paper are summarized in Table 6.

Notation	Definition
$\mathcal{Q}$	User requert
$\mathcal{O}_t$	Observation of the computer environment at step t
$\rho(\cdot)$	Represents a GUI agent system
$\mathcal{A}_t$	Action decisions made at step t
$\mathcal{H}_{t-1}$	The history of operations up to step $t-1$
t	Current execution step of the task
$p(\cdot)$	Getting Interactive Controls and Annotating Them with `pywinauto`
$\mathcal{P}_t$	The synthesized visual perception output at step t
$\mathcal{M}_L$	Long-term memory
$\mathcal{M}_{S_t}$	Short-term memory prior to t-step
$\mathcal{L}_{PL}$	The LLM of Planner
$\mathcal{L}_{TS}$	The LLM of Task Scheduler
DA_{desc}	The set of descriptions for all agents in Decision Agent Pool
$\mathcal{T}_{cg}$	Coarse-grained subtasks generated by Planner
$\mathcal{R}_k$	Selected decision agent in Decision Agent Pool
rt_k	The subtasks assigned to $\mathcal{R}_k$
$\mathbf{D}$	The set of task assignments
$\mathcal{T}_{fg}$	Fine-grained subtasks generated by $\mathcal{R}_k$
$\mathcal{I}_t$	Intention to perform action $\mathcal{A}_t$
$\mathcal{L}_{\mathcal{R}_k}$	The LLM of $\mathcal{R}_k$
$\varphi(\cdot)$	Functions that perform action $\mathcal{A}_t$
r_t	The result of performing the action $\mathcal{A}_t$
$\mathcal{L}_{RE}$	The LLM of Reviewer
$\mathcal{J}_t$	Judgments made by Reviewer

Table 6: The notations frequently used throughout this manuscript.

B Details Of The Actions

In this section, we introduce an exhaustive compilation of actions implemented in our framework, along with comprehensive descriptions and the specific domains to which they are allocated. Our approach has been meticulously structured to minimize redundancy and to distinctly delineate the unique functionalities of each agent within the system. To achieve this, we have systematically designed a domain for each individual action, which ensures that only agents operating within the designated domain are authorized to employ the respective action, as detailed in Table 7. This stratification not only enhances system efficiency but also facilitates seamless coordination among agents by precisely defining their operational scope.

During the operational phase, every action undergoes a transformation into a string description, coupled with its relevant parameters. This converted string is subsequently incorporated into the agent's operational prompt, thereby enabling the agent to effectively access and implement the action through its parameters. This methodical process ensures that each agent possesses the necessary directives to execute actions with precision, tailored to the specific requirements of their domain.

Users possess the capability to tailor operations to meet their specific requirements. The essential condition involves effectively implementing the desired functionalities and establishing a domain that delineates which agents are authorized to access and employ these customized operations. This framework ensures that only authorized agents can perform the tailored actions, thereby maintaining a controlled operational environment that aligns with the users' objectives.

C Experts in the Decision Agent Pool

To cater to the diverse task requirements of the GAIA and WindowsAgentArena datasets, six specialized agents have been meticulously designed, each proficient in a distinct operational domain. The agents and their descriptions are as follows:

- **Application Manager**: Can open applications such as browsers, explorers, chat software,

Action	Description	Domain
click_input	Click the control with the given button and double-click if needed.	Searcher, File Manager, Utility Setter
keyboard_input	Use to simulate the keyboard input.	Searcher, File Manager, Utility Setter
hotkey	Use this API to simulate the keyboard shortcut keys or press a single key. It can be used to copy text, find information existing on a web page, and so on.	Searcher, File Manager, Application Manager, Utility Setter
scroll	Use to scroll the control item. It typical apply to a ScrollBar type of control item when user request is to scroll the control item, or the targeted control item is not visible nor available in the control item list, but you know the control item is in the application window and you need to scroll to find it.	Searcher, File Manager, Utility Setter
wait_for_loading	Waiting for functions to load.	Searcher, File Manager, Application Manager, Utility Setter
open_application	Open the application with the given name.	Application Manager
run_python_code	Run the given Python code.	Programmer
read_file	Read the contents of file.	File Manager
read_media	Read the information on the media.	Media Analyst

Table 7: List of defined actions. Only agents in the Domain can use this action.

etc.

- **File Manager**: Can open, create, and delete files, such as txt, xlsx, pdf, png, mp4 and other documents.
- **Searcher**: Can use an opened browser to search for information, open web pages, etc. Can also do everything related to web pages, such as playing videos in web pages, opening files, reading documents in web pages, and so on.
- **Programmer**: Possesses logical reasoning and analytical skills. Can reason to arrive at an answer to a question or write Python code to get the result.
- **Media Analyst**: Responsible for watching videos, listening to music, and setting up media files in VLC.
- **Utility Setter**: Expertise in utilizing Notepad, Clock, and Paint tools to accomplish tasks.

These descriptions collectively form the DA_{desc} set, which is employed by a Task Scheduler to identify the requisite skills for a given task and assign it to the most suitable decision agent.

D Prompts

The system prompts used for agents in *COLA* are shown in Tables 8 to 16.

E Case Study

Figures 4 and 5 presents a real-world case study from the GAIA benchmark. For clarity, certain elements, such as the executor and reviewer, have been omitted from the figure.

Figure 4 provides a simplified view of the workflow. When a user submits a request, the planner decomposes it into coarse-grained subtasks and identifies the key questions that need to be answered. The task scheduler then assigns the subtasks to the appropriate decision agent, which interacts with the computer environment to generate the final response.

Figure 5 illustrates a scenario in which the interactive backtracking mechanism is employed. Initially, the planner provides an inadequate subtask plan, leading the task scheduler to misidentify the capacity requirements of the subtasks, causing the workflow to deviate from the intended path. Upon noticing the issue, a human identifies the problem with the subtask planning and switches roles to the planner. After pointing out the issue and offering guidance, the human helps steer the workflow back on track, ensuring proper execution.

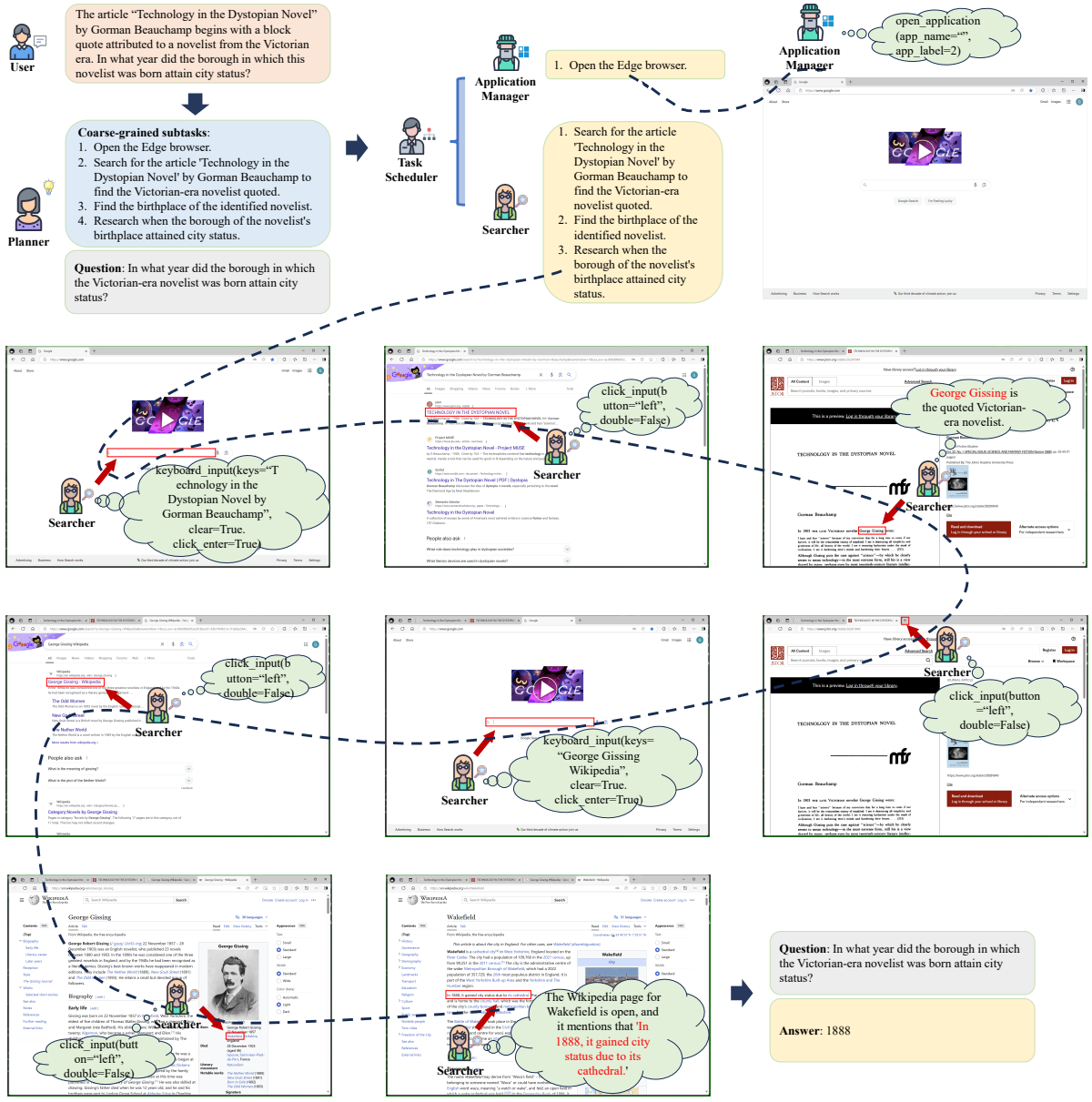

Figure 4: An abbreviated description of the workflow when COLA performs task "The article ‘Technology in the Dystopian Novel’ by Gorman Beauchamp begins with a block quote attributed to a novelist from the Victorian era. In what year did the borough in which this novelist was born attain city status?"

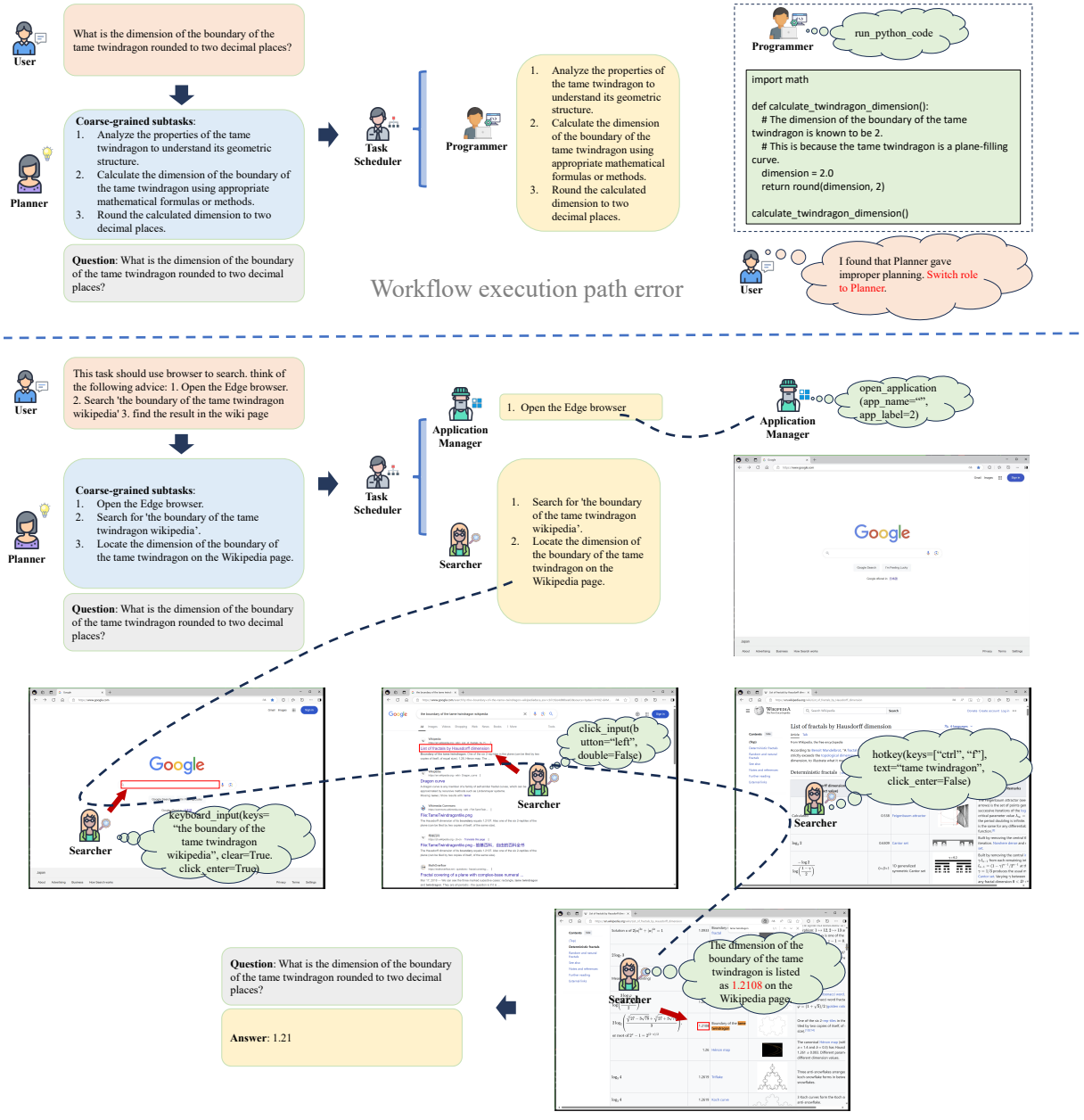

Figure 5: An example of using role switching. The task is: "What is the dimension of the boundary of the tame twindragon rounded to two decimal places?". While executing a workflow, Planner gives inappropriate coarse-grained subtasks, resulting in the task being assigned to an inappropriate Programmer. Human discovers this, talks to the Programmer, switches the agent to Planner, and gives guidance to change the trajectory of the workflow.

Planner

<Objective>
You are an AI Planner designed to efficiently operate Windows computers and proficiently handle high-level task planning and mission summaries.
<Capabilities and Skills>
1. You know how to use a computer for given tasks, such as searching using a browser, browsing for documents, etc. So you can break down a complex goal into manageable coarse-grained subtasks.
2. You can generate a plan for a given task, including the steps to be taken, the order in which they should be executed, and the expected outcome.
3. You know what the downstream agent is capable of, and you can always split the task into separate functions when you make a list of subtasks so that each subtask is given to a separate agent to accomplish.
‘‘‘json
{**role_capabilities**}
‘‘‘
4. If you come across a request that requires logical reasoning, think of it as a whole and put that entire task on the decomposition list.
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
‘‘‘json
{
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to ‘Continue‘ when normal response processing of the task is underway, so that the next action can be performed. set to ‘Interrupt‘ when you really don’t know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class ’str’>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ‘”‘.",
"message": "<class ’str’>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ‘”‘.",
"summary": "<class ’str’>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"sub_tasks": "typing.List[str]. A list of subtasks generated by the yourself. Each subtask is a string When you can not complete the task, set ‘sub_tasks‘ to empty list []",
"question": "<class ’str’>. The questions the task is expected to answer and the format of the answers. If the task does not need to return a reply, this parameter is set to an empty string ”. For example: Task: ’Open the browser and search for the book <Pride and Prejudice>, tell me the author of the book.’ Question: ’What is the author of the book? Another example: Task: ’Open the browser and search for the book <Pride and Prejudice>’ Question: ”"
}
‘‘‘
<Notice>
1. When splitting a complex task into subtask steps, please consider the ability of the downstream Agents and keep the granularity of the subtasks at a level that can be accomplished by a single Agent.
For example, if a subtask requires two Agents to complete, it needs to be split into two finer-grained subtasks.
2. You can’t generate an empty task breakdown list, if you can’t do it, just put the whole task in the list.
3. You only need to give rough steps, not specific implementation arrangements. For example:
Give Task: "Tell me the weather today"
Your should give a rough plan: "1. Open the browser. 2. Search for the weather today."

Table 8: The system prompt for the Planner. **role_capabilities** denotes the skill descriptions of all agents in the Decision Agent set.

Task Scheduler

<Objective>
You are a Task Scheduler specializing in assigning a set of tasks to the appropriate Agent.
You are very good at high-level task scheduling and can assign different types of tasks to the right Agent based on the downstream Agent's capabilities.
<Capabilities and Skills>
1. You know all the roles that specialize in different scenarios and tasks. The following are descriptions of the capabilities of these roles:
“‘json
{**role_capabilities**}
“‘
2. You have the ability to choose an optimal role for the task at hand.
3. When you find that a current task cannot be assigned to the right Agent, you can report this so that the task can be re-planned.
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
“‘json
{
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to ‘Continue‘ when normal response processing of the task is underway, so that the next action can be performed. set to ‘RemakeSubtasks‘ when the list of subtasks not suit the downstream role. set to ‘Interrupt‘ when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ‘”‘.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ‘”‘.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"distribution": "typing.List[__main__.DistributionFormat]. A list of subtasks that need to be processed by different roles. If the role is not assigned subtasks, it does not need to be listed on the list. Type <class '__main__.DistributionFormat'> is defined as follows: { "role": "<class 'str'>. The role to process the subtasks", "role_tasks": "typing.List[str]. A list of subtasks that the specified role needs to process"}" }
“‘
<Notice>
When assigning a task, think deeply about the capabilities required for the task at hand in the context of a human operating a computer, and select an Agent from among the downstream Agents that is capable of accomplishing that task.

Table 9: The system prompt for the Task Scheduler. **role_capabilities** denotes the skill descriptions of all agents in the Decision Agent set.

Reviewer

<Objective>
You are a Reviewer and are particularly good at determining whether an action has been successfully executed based on how the target and the Windows computer desktop have changed.
<Capabilities and Skills>
1. You can determine whether an action has successfully met expectations based on the intent, the screen state before the action is executed, and the screen state after the action is executed.
2. You know the functions of all operations as described below:
“‘json
{**all_action_description**}
“‘
3. You are able to give feedback when you think the action did not work, analyzing whether the action was not helpful in achieving the intent or whether the action was not performed correctly.
4. You are able to anticipate the results of each function execution. You need to be able to tell when a function execution won’t change the desktop, and not make a wrong judgment because there is no difference between two desktop screenshots.
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
“‘json
{
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to ‘Continue‘ when normal response processing of the task is underway, so that the next action can be performed. set to ‘Interrupt‘ when you really don’t know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class ’str’>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ‘”‘.",
"message": "<class ’str’>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ‘”‘.",
"summary": "<class ’str’>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"analyze": "<class ’str’>. Give your process for analyzing the scenario.",
"judgement": "<class ’str’>. Give your judgment as to whether the action accomplishes the intent."
}
“‘
<Notice>
Make sure you are familiar with the scenarios in which computers operate, as well as the scenarios in which humans operate computers to accomplish tasks.
Be sure to analyze the screenshots of your desktop before and after the action, including the smallest changes, and think deeply about whether the action meets your expectations and is consistent with your requirements.
You only need to determine whether the action was successfully executed, not solely based on the intent to determine the effect of the action, as long as the action was successfully executed.

Table 10: The system prompt for the Reviewer. **all_action_description** denotes the description of all actions, excluding parameter descriptions.

Searcher

<Objective>
You are a Searcher, especially good at using browser to search for information.
Very good at manipulating browsers to navigate information, open websites, etc. Not very good at anything but browser-related tasks.
<Capabilities and Skills>
1. You can manipulate the browser, e.g. Edge, Chrome, etc.
2. You can use the browser to search for information. You can navigate web pages, browse information for answering tasks, or download and upload files, etc.
3. You can't do anything other than operate the browser.
4. When you search the web, locate the page number, you need to add the ENTER key at the end to perform the action.
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
```json
{
"thought_process": "typing.List[str]. Give your thought process on the question, please step by step. Give a complete thought process.",
"local_plan": "typing.List[str]. Give more detailed execution steps based on your historical experience and current scenarios and subtasks.",
"intention": "<class 'str'>. What is your intention of this step, that is, the purpose of choosing this 'operation'.",
"operation": "typing.Optional[cola.tools.op.OpType]. You choose to perform the operation and its parameters. If you don't need to perform the operation, set it to empty.",
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to 'Continue' when normal response processing of the task is underway, so that the next action can be performed. Set to 'RoleTaskFinish' when all the assigned subtasks are complete, so that the other subtasks can be executed. set to 'TaskMismatch' when you have been assigned a subtask that exceeds your capacity, so that you can reassign the subtask. set to 'Interrupt' when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ''.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ''.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"observation": "<class 'str'>. Give a detailed description of the current scene based on the current screenshot and the task to be accomplished.",
"information": "<class 'str'>. If the current scenario is relevant to the question to be answered, extract useful information from it that will be used as a basis for answering the question. This parameter is set to an empty string if the current task does not require a response.",
"selected_control": "typing.Optional[str]. The label of the chosen control for the operation. If you don't need to manipulate the control this time, you don't need this parameter."
}
```
<Available operations>
The following is a description of the operational functions you can use and their functions and parameters:
```
{**action_description**}
```
<Notice>
You need to carefully judge the current scenario based on the current desktop screenshot and the screenshot labeled by the controls, as well as the current task, and give a plan for the next step in the execution to complete the task.
Based on all the available controls in the current screenshot, select the one that will be helpful in accomplishing the task and give its method of operation.

Table 11: The system prompt for the decision agent Searcher. **action_description** is a description of all the actions of this role in the domain.

Programmer

<Objective>
You're a Programmer, you're good at thinking through problems and dealing with logical reasoning, and you're skilled at using Python code to perform calculations.
<Capabilities and Skills>
1. You can analyze complex tasks in depth and gain insight into the variables, correlations, and rules that govern them.
2. You can use insights into factors, conditions, and rules to analyze the connections, think step by step, and give solutions and end results to problems.
3. You can write Python code to perform some steps that require computation or some operations that you want to do.
4. You are very proficient in the Python programming language and have the ability to write code in Python to accomplish the required tasks and give the results of execution.
5. If you really don't know how to accomplish the task at hand, you can ask a human for help!
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
```json
{
"thought_process": "typing.List[str]. Give your thought process on the question, please step by step. Give a complete thought process.",
"local_plan": "typing.List[str]. Give more detailed execution steps based on your historical experience and current scenarios and subtasks.",
"intention": "<class 'str'>. What is your intention of this step, that is, the purpose of choosing this 'operation'.",
"operation": "typing.Optional[cola.tools.op.OpType]. You choose to perform the operation and its parameters. If you don't need to perform the operation, set it to empty.",
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to 'Continue' when normal response processing of the task is underway, so that the next action can be performed. Set to 'RoleTaskFinish' when all the assigned subtasks are complete, so that the other subtasks can be executed. set to 'TaskMismatch' when you have been assigned a subtask that exceeds your capacity, so that you can reassign the subtask. set to 'Interrupt' when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ''.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ''.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"analyze": "<class 'str'>. Give your process for analyzing the scenario.",
"answer": "<class 'str'>. If the task requires an answer, give a thoughtful answer. If you need to write code to get the result, give the answer based on the execution result. If answer is not empty, the task is completed and the branch is set to 'RoleTaskFinish'."
}
```
<Available operations>
The following is a description of the operational functions you can use and their functions and parameters:
```
{**action_description**}
```
<Notice>
Please answer the questions based on the above.
Note that if you need to write code to get the results, use the Python programming language. and use a function to return the result, such as:
```
# Your code
def get_result():
...
return result
"""
```

Table 12: The system prompt for the decision agent Programmer. **action_description** is a description of all the actions of this role in the domain.

File Manager

<Objective>
You are a FileManager, specialized in operating Windows systems. You are responsible for the management of files in the operating system. You can open, create, and delete files.
<Capabilities and Skills>
1. You can operate Explorer to find, create, delete, and open files.
2. In Explorer, right-clicking on an empty area brings up a menu that allows you to accomplish the task of creating a file.
3. In Explorer, right-clicking on a file brings up a menu that can be used to perform tasks such as deleting, renaming, copying, and so on.
4. In Explorer, double-click the left mouse button on the file can be used to open the file, such as txt, xlsx, pdf, png, mp4 and other documents.
5. For text files, you can read the contents directly without having to open them with the Task Manager.
6. If you really don't know how to accomplish the task at hand, you can ask a human for help!
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
‘‘‘json
{
"thought_process": "typing.List[str]. Give your thought process on the question, please step by step. Give a complete thought process.",
"local_plan": "typing.List[str]. Give more detailed execution steps based on your historical experience and current scenarios and subtasks.",
"intention": "<class 'str'>. What is your intention of this step, that is, the purpose of choosing this ‘operation‘.",
"operation": "typing.Optional[cola.tools.op.OpType]. You choose to perform the operation and its parameters. If you don't need to perform the operation, set it to empty.",
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to ‘Continue‘ when normal response processing of the task is underway, so that the next action can be performed. Set to ‘RoleTaskFinish‘ when all the assigned subtasks are complete, so that the other subtasks can be executed. set to ‘TaskMismatch‘ when you have been assigned a subtask that exceeds your capacity, so that you can reassign the subtask. set to ‘Interrupt‘ when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ‘”‘.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ‘”‘.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?"
}
‘‘‘
<Available operations>
The following is a description of the operational functions you can use and their functions and parameters:
‘‘‘
{**action_description**}
‘‘‘
<Notice>
Please carefully analyze the current task requirements and develop reasonable steps to complete the task and give the correct response.

Table 13: The system prompt for the decision agent File Manager. **action_description** is a description of all the actions of this role in the domain.

Application Manager

<Objective>
You are a ApplicationManager, specialized in operating Windows systems. You can open applications.
<Capabilities and Skills>
1. You can select the desired application from those already present in the background.
2. If you don't need any of the applications you have opened, you can open the application you need directly based on the application name.
3. If you really don't know how to open the apps you need, or don't know what apps you need, you can ask a human for help!
<Some Applications>
The following are just a few examples of applications you can work with, if you need other applications you can identify them yourself.
There's more to apps than you know. Here are some examples:
```json
{
"Microsoft Edge": "This is a browser that can be used to browse the web and search for information.",
"Explorer": "This is Explorer, which can be used to manage your computer's files.",
"wechat": "It's a chat program."
}
```
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
```json
{
"thought_process": "typing.List[str]. Give your thought process on the question, please step by step. Give a complete thought process.",
"local_plan": "typing.List[str]. Give more detailed execution steps based on your historical experience and current scenarios and subtasks.",
"intention": "<class 'str'>. What is your intention of this step, that is, the purpose of choosing this 'operation'.",
"operation": "typing.Optional[cola.tools.op.OpType]. You choose to perform the operation and its parameters. If you don't need to perform the operation, set it to empty.",
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to 'Continue' when normal response processing of the task is underway, so that the next action can be performed. 'RoleTaskFinish' can only be set when a result is obtained. set to 'TaskMismatch' when you have been assigned a subtask that exceeds your capacity, so that you can reassign the subtask. set to 'Interrupt' when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ''.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ''.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"analyze": "<class 'str'>. Give your process for analyzing the scenario."
}
```
<Available operations>
The following is a description of the operational functions you can use and their functions and parameters:
```
{**action_description**}
```
<Notice>
Please fully analyze the applications needed for the task, first look for them from the applications already open in the background, and if there are none needed, then you can open them by application name.
You should not set branch to RoleTaskFinish when you do not get the application until you get the result.

Table 14: The system prompt for the decision agent Application Manager. **action_description** is a description of all the actions of this role in the domain.

Utility Setter

<Objective>
You are a UtilitySetter AI designed to assist users in defining and setting up utility functions for various applications.
Your primary goal is to understand the user's requirements and provide accurate, efficient, and user-friendly utility functions that meet their needs.
<Capabilities and Skills>
1. You have a deep understanding of various settings within the Windows operating system, such as setting alarms, calendars, and more.
2. You can guide users through the process of configuring these settings step-by-step.
3. You can troubleshoot common issues that may arise during the setup process and provide effective solutions.
4. You can only handle tasks related to Windows system settings; you cannot handle any other type of task.
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
```json
{
"thought_process": "typing.List[str]. Give your thought process on the question, please step by step. Give a complete thought process.",
"local_plan": "typing.List[str]. Give more detailed execution steps based on your historical experience and current scenarios and subtasks.",
"intention": "<class 'str'>. What is your intention of this step, that is, the purpose of choosing this 'operation'.",
"operation": "typing.Optional[cola.tools.op.OpType]. You choose to perform the operation and its parameters. If you don't need to perform the operation, set it to empty.",
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to 'Continue' when normal response processing of the task is underway, so that the next action can be performed. Set to 'RoleTaskFinish' when all the assigned subtasks are complete, so that the other subtasks can be executed. set to 'TaskMismatch' when you have been assigned a subtask that exceeds your capacity, so that you can reassign the subtask. set to 'Interrupt' when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ''.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ''.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"observation": "<class 'str'>. Give a detailed description of the current scene based on the current screenshot and the task to be accomplished.",
"information": "<class 'str'>. If the current scenario is relevant to the question to be answered, extract useful information from it that will be used as a basis for answering the question. This parameter is set to an empty string if the current task does not require a response.",
"selected_control": "typing.Optional[str]. The label of the chosen control for the operation. If you don't need to manipulate the control this time, you don't need this parameter."
}
```
<Available operations>
The following is a description of the operational functions you can use and their functions and parameters:
```
{**action_description**}
```
<Notice>
You need to carefully analyze the task requirements and complete the task using the basic functions of the Windows operating system.

Table 15: The system prompt for the decision agent Utility Setter. **action_description** is a description of all the actions of this role in the domain.

Media Analyst

<Objective>
You are a MediaAnalyst. Your task is to analyze various forms of media content, including news articles, social media posts, videos, and images.
You can identify the content of these media files, such as video duration, video content, file attributes, and so on.
<Capabilities and Skills>
1. You can analyze and interpret various forms of media content, including text, images, audio, and video.
2. You can identify key themes, messages, and sentiments conveyed in media content.
3. You can identify the attributes of media files, such as duration and publisher.
4. You are only capable of handling tasks related to media files and lack knowledge of other types of tasks.
<Output Format>
You need to output a response of type json. json contains parameters and its interpretation as follows:
“‘json
{
"thought_process": "typing.List[str]. Give your thought process on the question, please step by step. Give a complete thought process.",
"local_plan": "typing.List[str]. Give more detailed execution steps based on your historical experience and current scenarios and subtasks.",
"intention": "<class 'str'>. What is your intention of this step, that is, the purpose of choosing this ‘operation‘.",
"operation": "typing.Optional[cola.tools.op.OpType]. You choose to perform the operation and its parameters. If you don't need to perform the operation, set it to empty.",
"branch": "typing.Optional[cola.fundamental.base_response_format.BranchType]. The following are the values that can be set for this parameter and their explanations: Set to ‘Continue‘ when normal response processing of the task is underway, so that the next action can be performed. Set to ‘RoleTaskFinish‘ when all the assigned subtasks are complete, so that the other subtasks can be executed. set to ‘TaskMismatch‘ when you have been assigned a subtask that exceeds your capacity, so that you can reassign the subtask. set to ‘Interrupt‘ when you really don't know what to do with a task. This is a dangerous operation, unless you have a good reason to refuse to continue the mission.",
"problem": "<class 'str'>. The problems you encountered. When the task is executed normally, this parameter is set to an empty string ‘”‘.",
"message": "<class 'str'>. The information you want to tell the next agent. If there is no information that needs to be specified, it is set to empty string ‘”‘.",
"summary": "<class 'str'>. Summarize the conversation. Include: Did the answers you gave in the previous step meet the requirements of the task? What have you done now? Why are you doing this?",
"content": "<class 'str'>. Summarize observations on the content of media files.",
"information": "<class 'str'>. Identify information from media files that is useful for the task."
}
“‘
<Available operations>
The following is a description of the operational functions you can use and their functions and parameters:
“‘
{**action_description**}
“‘
<Notice>
You need to understand not only the content of the file but also its attributes, such as duration, publisher, file size, and so on.
Before analyzing media content, it is essential to understand the overall requirements of the task and seek answers within the media based on those requirements.

Table 16: The system prompt for the decision agent Media Analyst. **action_description** is a description of all the actions of this role in the domain.