OS-SYMPHONY: A Holistic Framework for Robust and Generalist Computer-Using Agents

Anonymous ACL submission

Abstract

While Vision-Language Models (VLMs) have significantly advanced Computer-Using Agents (CUAs), current agentic frameworks struggle with robustness in novel domains and long-horizon workflows due to the absence of visual-aware tutorial retrieval and the lack of granular control over historical visual context curation and pruning. To bridge these gaps, we introduce OS-SYMPHONY, a holistic framework that comprises an Orchestrator coordinating two key innovations for robust automation: 1) a Reflection-Memory Agent that utilizes milestone-driven long-term memory to enable trajectory-level self-correction, effectively mitigating visual context loss in long-horizon tasks; 2) Versatile Tool Agents featuring a Multimodal Searcher that adopts a *See-Act* paradigm to navigate a browser-based sandbox to synthesize live, visually aligned tutorials, thereby resolving fidelity issues in unseen scenarios. Experimental results demonstrate that OS-SYMPHONY delivers substantial performance gains across varying model scales, establishing new state-of-the-art results on three online benchmarks, notably achieving 65.84% on OSWorld. All research assets will be made publicly available.

1 Introduction

The landscape of digital task automation has been reshaped by the advancement of Vision-Language Models (VLMs) (Bai et al., 2025b,a; Wang et al., 2025b; Anthropic, 2025b; Hong et al., 2025), giving rise to vision-guided Computer-Using Agents (CUAs) (Sun et al., 2024c; Qin et al., 2025; Wang et al., 2025a; Xie et al., 2025a; Wu et al., 2025e; Liu et al., 2025b). By leveraging visual perception to interact with digital environments, these agents have significantly expanded the scope and applicability of general-purpose automation.

While native CUAs (Zhang et al., 2024; Hu et al., 2025a; Liu et al., 2025b; Wang et al., 2025c) trained on large-scale computer-using trajectories are capable of digital navigation tasks, they often struggle to generalize to complex scenarios under a single-agent paradigm. Consequently, modular CUA frameworks (Wu et al., 2024a; Song et al., 2025; Gonzalez-Pumariega et al., 2025; Guo et al., 2025b) have emerged by orchestrating multiple specialized sub-agents, *e.g.*, planner, grounder, and coder, to coordinate seamlessly, exhibiting significant potential for developing reliable generalist CUAs.

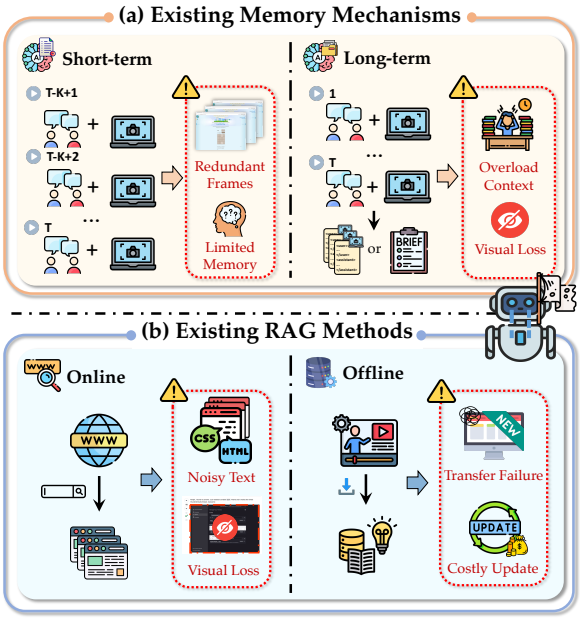


Figure 1: Current limitations in CUA framework.

Despite the promising progress in agentic frameworks, they face two critical challenges. First, while memory modules (Song et al., 2025; Cheng et al., 2025; Tian et al., 2025) are employed to support long-horizon tasks, current context management mechanisms often lack granular control over historical visual information curation and pruning. As illustrated in Fig. 1(a), this deficiency results in suboptimal utilization of historical visual information, rendering agents ill-equipped to identify potential errors like intent drift or cyclic behaviors.

This lack of retrospective insight ultimately prevents the generation of meaningful reflections to refine planning in complex, long-horizon tasks. Second, several works (Xu et al., 2024a; Agashe et al., 2025a; Guo et al., 2025b; Xu et al., 2025) incorporate external knowledge via Retrieval-Augmented Generation (RAG) in an effort to generalize to unseen scenarios; however, as shown in Fig. 1(b), they either excessively rely on unimodal information, thereby overlooking vital semantic cues in the visual modality, or depend on local knowledge bases that incur high maintenance costs and struggle to adapt to new software. Consequently, these approaches fail to achieve robust generalization on Out-of-Distribution (OOD) tasks.

To this end, we propose OS-SYMPHONY, a holistic CUA framework comprising a decision-making **Orchestrator** coordinating two core designs to bridge these gaps: 1) **Reflection-Memory Agent** that leverages long-term memory to retain key "milestone" screenshots alongside abstract trajectories, effectively mitigating visual context loss. By visually auditing historical states, the RMA generates critical trajectory-level reflections according to a structured message protocol, providing high-level guidance for the Orchestrator to ensure robust performance over long-horizon tasks. 2) **Versatile Tool Agents**, highlighted by a meticulously designed *Multimodal Searcher* alongside a Coder and Grounders that work synergistically to execute complex tasks. Specifically, our Searcher enables acquiring diverse tutorials via web browsers autonomously. By integrating visual information with spatial layouts, it provides high-fidelity, relevant tutorials, enabling the Orchestrator to leverage external multimodal knowledge for OOD scenarios.

We demonstrate the effectiveness of OS-SYMPHONY across diverse scales and benchmarks, achieving substantial performance leaps over current state-of-the-art methods with scores of 65.8% on OSWorld (↑2.4%), 63.5% on WindowsAgentArena (↑6.9%), and 46.0% on MacOSArena (↑38.0%). Beyond quantitative results, our rigorous ablation and granular analysis dissect the core drivers of this performance, offering valuable directions for future CUA development.

Our contributions are summarized as follows:

1) We propose OS-SYMPHONY, a full-fledged CUA framework which investigates a holistic and robust paradigm via diverse agent collaboration to solve complicated tasks in practice.

2) We design a Reflection-Memory Agent that integrates milestone-driven long-term memory with a structured auditing protocol to generate in-depth reflection for robust long-horizon planning.

3) We develop a suite of tool agents which facilitate orchestrator to solve tasks effectively. Among these tools, *Multimodal Searcher* is highlighted to harvest rich multimodal knowledge for OOD tasks by actively navigating the web pages.

4) Extensive evaluations across diverse operating systems and model scales validate the superior performance of OS-SYMPHONY. Furthermore, our framework empowers open-source VLMs to successfully execute unseen, long-horizon tasks that previously challenged their capabilities.

2 Related Work

Computer-Using Agents (CUAs). With the rapid development of Vision-Language Models (VLMs) (Anthropic, 2025a; Comanici et al., 2025; OpenAI, 2025c; Wang et al., 2025b; Bai et al., 2025a), Computer-Using Agents have become a novel paradigm to explore Human-Computer Interaction. Native CUAs pursue end-to-end digital autonomy, encompassing both general-purpose models (OpenAI, 2025a; Guo et al., 2025a; Anthropic, 2025b) adapted for agentic tasks and specialized models (Cheng et al., 2024; Wu et al., 2024b; Xu et al., 2024b; Qin et al., 2025; Wang et al., 2025c) fine-tuned on large-scale GUI datasets for dedicated computer use. In parallel, CUA frameworks (Wu et al., 2024a; Agashe et al., 2025b; Yang et al., 2025a; Wu et al., 2025c; Ye et al., 2025a; Zhang et al., 2025c) prioritize modularity by decomposing complex tasks. This approach enhances capability through modular collaboration while mitigating the data dependency of end-to-end training. Beyond architectural paradigms, the field is transitioning from purely vision-based approaches (Zhang et al., 2025a,b; Wang et al., 2025a) toward hybrid GUI-API strategies. While some methods (Sun et al., 2024a; Song et al., 2025; Gonzalez-Pumariega et al., 2025) leverage general-purpose interfaces like code execution, others (Lai et al., 2025; Yang et al., 2025b; Jia et al., 2025) depend on software-specific APIs via protocols such as MCP. However, relying on specific APIs often hinders generalization to niche or proprietary software. Consequently, retrieval-augmented strategies have emerged as a critical solution to bridge this gap.

RAG for CUAs. To strengthen the generalization of CUAs, an increasing number of stud-

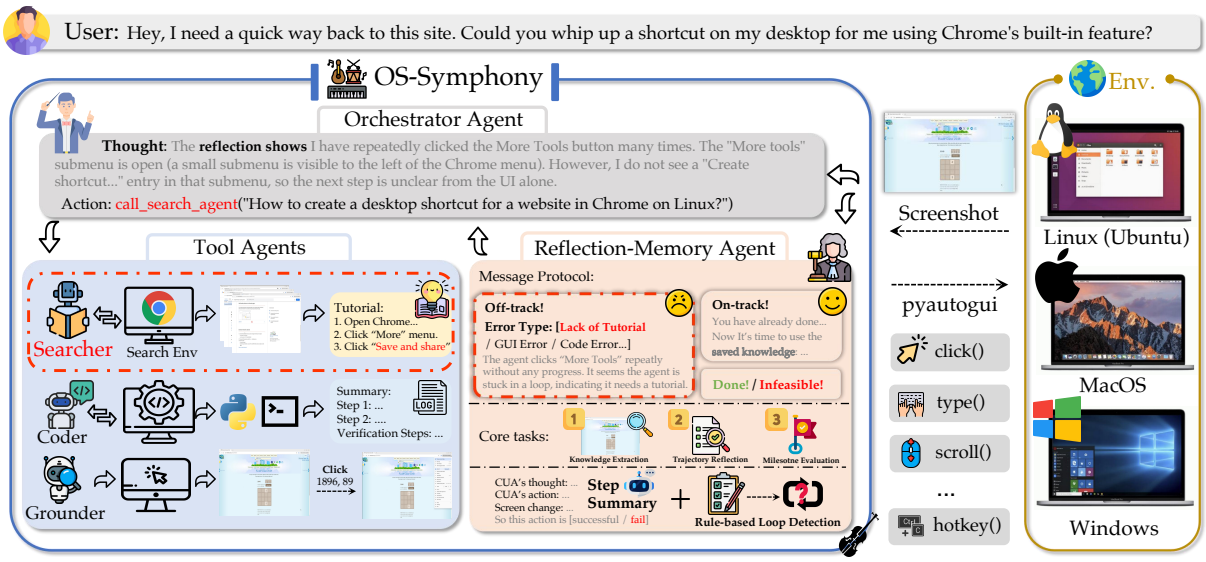


Figure 2: **Pipeline overview.** OS-SYMPHONY comprises three primary components: (1) The **Orchestrator**, acting as the system's brain, responsible for task understanding and action prediction; (2) Tool Agents, consisting of **Grounder**, **Coder**, and **Searcher**, where the Searcher retrieves up-to-date tutorials in a human-like manner; and (3) The **Reflection-Memory Agent**, which compresses execution trajectories to maintain long-term memory and facilitate trajectory-level reflection.

ies (Zhang et al., 2024; Nguyen et al., 2025; Sun et al., 2024b; Hu et al., 2025a) has integrated Retrieval-Augmented Generation (RAG) to access external knowledge. Early lines of research (Shi et al., 2025; Agashe et al., 2025a; Mei et al., 2025; Guo et al., 2025b) rely on general-purpose AI search engines (*e.g.*, Perplexica[1]) to perform static knowledge retrieval prior to task execution. In Parallel, other efforts construct task-specific knowledge databases or training corpora by manually curating software documentation (Xu et al., 2024a, 2025) and video tutorials (Zhao et al., 2025). More recently, inspired by the DeepResearch paradigm (Gemini Team, 2025; Team et al., 2025; Grok Team, 2025; Tao et al., 2025; Wu et al., 2025a,b), emerging methods have begun to tightly integrate GUI interaction with deep research capabilities, enabling agents to mine web-scale resources for complex reasoning (Wang et al., 2025a,e). However, purely text-based retrieval struggles with software tutorials and documentation that are inherently visual, such as step-by-step interfaces illustrated with screenshots. Moreover, the substantial redundancy in raw HTML introduces a serious burden for context management (Li et al., 2025a), especially when retrieval and parsing unfold over multiple interaction turns.

Memory for CUAs. Efficient memory management is pivotal for long-horizon tasks (Hu et al., 2025b; Zhang et al., 2025d; Hu et al., 2025a; Wang et al., 2024; Sun et al., 2025a). Recent LLM-based methods leverage history summarization (Wu et al., 2025d; Yu et al., 2025) or context folding (Ye et al., 2025b; Sun et al., 2025b) to compress interactions. Other approaches (Wu et al., 2024a; Li et al., 2025b) leverage multi-scale memory, fusing short-term working memory and long-term procedural memory to maintain a comprehensive repository of trajectories and acquired knowledge. In GUI domain, some approaches (Cheng et al., 2025; Tian et al., 2025) further distill trajectories, including screenshots, thoughts, and actions, into structured summaries. However, these approaches remain largely text-centric, often losing the visual semantics that are essential for progress tracking and downstream decision-making.

3 OS-SYMPHONY

In this section, we present the overall framework of OS-SYMPHONY, which comprises three synergistic components: an Orchestrator, specialized Tool Agents, and a Reflection-Memory Agent (RMA). In the case shown in Fig. 2, the Orchestrator first interprets feedback from the RMA, which identifies execution stagnation caused by a Chrome version mismatch between the task environment and the VLM's pre-training knowledge, together with a *Lack of Tutorial* error. It then invokes the Searcher to retrieve a relevant tutorial, and following this guidance, successfully completes the task, achieving a closed-loop self-improvement cycle.

[1] https://github.com/ItzCrazyKns/Perplexica

3.1 Orchestrator

We employ an Orchestrator, which serves as the core component, responsible for task interpretation, coordinating all Tool Agents, and finally selects an action. Formally, we model the decision-making process of the Orchestrator as:

$$t_i, a_i = \mathcal{F}_O(\mathcal{I}, \mathcal{R}_i, o_i, \mathcal{T}, \mathcal{H}_{\text{short}}), \qquad (1)$$

where $\mathcal{F}_O$ represents the Orchestrator and t_i and a_i denote the thought and action components of the agent's output. Additionally, $\mathcal{I}$ represents the instruction, $\mathcal{R}_i$ signifies the reflection feedback provided by the RMA, o_i denotes the current screenshot, and $\mathcal{T}$ is the retrieved tutorial, which is provided upon invoking the Searcher and is empty otherwise. Notably, $\mathcal{H}_{\text{short}} = \{(o_j, t_j, a_j)\}_{j=i-K+1}^{i-1}$ denotes the short-term memory. Assuming the significance of historical interactions decays with time, we restrict the Orchestrator's memory to a sliding window of the last K turns, capturing the immediate dialogue and screenshots essential for precise next-action prediction.

3.2 Reflection-Memory Agent

Current CUA frameworks suffer from intent drift and insufficient error awareness when executing long-horizon tasks, due to the lack of a concise yet effective memory mechanism. To address this, we introduce a Reflection-Memory Agent (RMA) that manages a milestone-driven long-term memory, which is primarily responsible for alleviating the contextual memory overhead of the Orchestrator. A crucial insight driving our design is that, despite their information density, screenshots exhibit high temporal redundancy. Consequently, our memory mechanism compresses interaction history while selectively retains only those screenshots identified as milestones. Based on the proposed memory, the RMA generates trajectory-level reflections via a meticulously designed message protocol, thereby providing effective error correction for the Orchestrator. The pipeline of the RMA is shown in Fig. 3.

Step-Level Summary. We first utilize an auxiliary VLM to fulfill two tasks: summarizing the latest action and verifying its correctness at the GUI execution level. It can be formally defined as:

$$\mathcal{S}_i, s_i = \mathcal{F}_S(\mathcal{O}_{i-1}, o_{i-1}, o_i, \tilde{o}_{i-1}), \qquad (2)$$

where $\mathcal{F}_S$ represents the auxiliary VLM, S_i is the execution summary and s_i indicates the success status of the GUI action. O_{i-1} is the Orchestrator's output from the previous turn, and $\tilde{o}$ is a localized zoom-in of the action area.

Trajectory-Level Reflection. Building upon step-level summaries, we construct a milestone-driven long-term memory module, denoted as $\mathcal{H}_{\text{long}} = \{(\mathcal{S}_j, o_j, m_j)\}_{j=1}^{i-1}$. This module aggregates historical summaries $\mathcal{S}_j$, observations o_j, and binary milestone markers m_j. Specifically, the marker m_j is generated by the RMA, serving as a gatekeeper such that the RMA processes the observation o_j exclusively when m_j is active (*i.e.*, true).

At each step, the RMA utilizes $\mathcal{H}_{\text{long}}$ to perform three core functions: (1) milestone identification; (2) trajectory-level reflection generation; and (3) relevant information extraction from visual inputs (*e.g.*, retrieving a restaurant detail). The formal RMA operation is defined as:

$$\mathcal{R}_i, m_i, k_i = \mathcal{F}_R(\mathcal{O}_{i-1}, o_i, \mathcal{H}_{\text{long}}), \qquad (3)$$

where $\mathcal{F}_R$ represents the RMA and k_i is potential knowledge (or empty if no useful information).

The trajectory-level reflection follows a structured message protocol that categorizes execution states into four classes: *On-track*, *Completed*, *Infeasible*, or *Off-track*. Specifically, we classify *Off-track* scenarios into four distinct error types: (1) GUI Error, where step-level actions fail to achieve expected results (*e.g.*, clicking wrong elements), derived from s_i (Eq. 2) as a key heuristic; (2) Lack of Tutorial, triggered when random actions or repetitive loops suggest a need for external guidance; specially, we design a rule-based loop detection algorithm to assist the RMA, and details are provided in the Appendix A.2; (3) Code Error, identified when a mismatch between the Coder Agent's execution and the actual GUI state; and (4) Other Error, such as intent drift.

Empowered by this streamlined yet effective memory mechanism, the RMA facilitates high-fidelity reflections for the Orchestrator, which guarantees robust performance across long-horizon tasks. For more details and qualitative case studies, please refer to Appendix A.2 and C.1, respectively.

3.3 Versatile Tool Agents

The Orchestrator resolves diverse tasks by synthesizing three specialized tool agents: a novel Searcher to handle out-of-distribution (OOD) knowledge, Grounders for precise UI localization, and a Coder for direct system interface interaction.

Multimodal Searcher. Current CUA frameworks still struggle to generalize to OOD tasks. So we in-

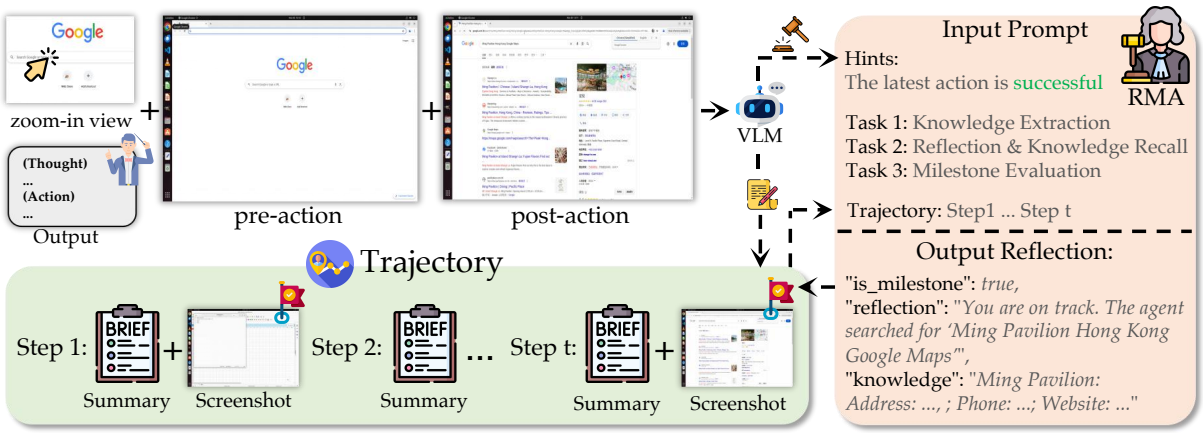


Figure 3: **Pipeline of RMA.** At each step, RMA summarizes the previous action using pre- and post-action screenshots and the Orchestrator's output, while evaluating the current GUI operation's correctness. It then generates a reflection from all summaries and milestone screenshots, and determines whether the latest step is a milestone.

troduce a *Visual-Centric Search as a Tool* paradigm, where the Searcher adopts a VLM-driven *See-Act* (Zheng et al., 2024) strategy to interact with rendered pages and synthesize tutorials. Compared to conventional RAG, it preserves critical visual cues beyond text parsing and makes retrieval truly on-demand, invoked only when execution reveals knowledge gaps. The Searcher operates as follows:

First, upon invocation, the Orchestrator formulates a "*How-to*" query q and pairs it with the current main-environment observation o_t, ensuring retrieval is aligned with the CUA's immediate execution state. To avoid disrupting the main environment, we construct an isolated *Search Environment*, a dedicated browser-based sandbox initialized on the search results page for q.

Then, the Searcher operates within a strictly bounded inner loop in the sandbox. We restrict its action space to $\mathcal{A}_{\text{search}} = \{\texttt{click}, \texttt{type}, \texttt{scroll}\}$, augmented with terminal actions $\{\texttt{done}, \texttt{fail}\}$. This compact design is sufficient for webpage navigation and reading while substantially reducing search complexity. We further encourage the Searcher to visit multiple pages to cross-check and triangulate information.

Finally, the loop terminates under strict criteria. The Searcher returns `fail` if no relevant information is found within the step budget. Conversely, it triggers `done` only when the tutorial is deemed highly relevant to q. Upon completing the exploration with `done`, the agent distills the visited content into a structured step-by-step tutorial $\mathcal{T}$, which is permanently appended to the Orchestrator's context for real-time reference during subsequent steps (see Appendix C.1 for a comprehensive workflow case study).

Grounders. For the common UI elements localization, we utilize two complementary agents: (1) a *General Grounder* that integrates low-level visual cues (*e.g.*, position, appearance) and high-level semantic context (*e.g.*, functionality, relevance); and (2) an *OCR-based Grounder* tailored for text-intensive apps (*e.g.*, PowerPoint, Word), which performs word-level OCR to construct a structured `{text,id,bbox}` table, followed by VLM-guided ID selection for precise coordinate lookup. A detailed mapping between actions and grounders is provided in Appendix A.3.

Coder. Existing grounding models still struggle with fine-grained localization and exhibit low efficiency in bulk manipulation scenarios. So inspired by (Gonzalez-Pumariega et al., 2025; Song et al., 2025), we also integrate a Coder specializing in file editing and configuration. At each invocation, the Coder receives a subtask from the Orchestrator, and then starts a strict internal workflow involving file localization, inspection, in-place modification, and verification. After execution, the Coder returns a concise synopsis, and the output is validated by the Orchestrator via GUI state checks. If the Coder fails or validation detects an error, it falls back to completing the subtask via GUI actions.

4 Experiment

4.1 Experiment Setup

Evaluation Benchmarks. Our evaluation centers on desktop environments, employing **OSWorld-Verified**[2] (Xie et al., 2024, 2025b) as our primary benchmark, which comprises 369 real-world tasks across five domains in a Ubuntu environment. Following common practice (Xie et al.,

[2]In this paper, "OSWorld" refers to "OSWorld-Verified".

Method	Step	Success Rate(%)					
		OS	Office	Daily	Prof.	Work.	Avg.
General Models & Specialist Model							
Qwen3-VL-32B-Inst. (2025a)	50	–	–	–	–	–	32.40
Qwen3-VL-32B-Thinking (2025a)	50	–	–	–	–	–	41.00
OpenCUA-72B (2025c)	100	61.13	44.73	49.95	72.58	22.16	44.91
UI-TARS (2025)	100	41.67	50.42	55.69	51.02	14.66	41.85
UI-TARS-2 (2025a)	100	41.67	61.11	62.12	61.22	34.13	53.10
DeepMiner-Mano-7B (2025)	100	50.00	39.28	44.87	73.47	17.20	40.15
DeepMiner-Mano-72B (2025)	100	66.67	63.22	52.51	**83.67**	24.41	53.91
Claude-Sonnet-4.5 (2025b)	100	70.83	**72.59**	61.35	63.27	49.54	62.84
Agentic Framework							
UiPath w/ GPT-5 (2025)	50	73.91	49.52	62.12	71.43	37.30	53.69
CoACT-1 w/ GPT-5 (2025)	50	70.83	60.65	54.09	69.39	42.37	56.39
CoAct-1 w/ GPT-5 (2025)	100	75.00	62.93	57.94	71.43	47.87	59.93
CoAct-1 w/ GPT-5 (2025)	150	75.00	62.93	61.78	71.43	47.87	60.76
GTA1 w/ GPT-5 (2025a)	100	**79.17**	63.91	62.56	79.59	50.91	63.41
Agent S3 w/ Qwen3-VL-32B-Inst. (2025)♣	50	50.00	36.67	50.62	61.22	21.96	40.11
Agent S3 w/ GPT-5-Mini (2025)♣	50	62.50	54.62	46.67	44.90	37.04	47.58
Agent S3 w/ GPT-5 (2025)	100	77.50	66.46	61.23	69.80	51.37	62.63
OS-SYMPHONY w/ Qwen3-VL-32B-Inst.	50	58.33	40.94	53.54	75.10	31.24	46.86
OS-SYMPHONY w/ Qwen3-VL-32B-Thinking	50	70.83	40.97	66.04	73.08	31.22	50.23
OS-SYMPHONY w/ GPT5-Mini	50	73.68	58.17	61.39	75.00	47.37	58.05
OS-SYMPHONY w/ GPT-5	50	75.00	64.85	61.19	69.23	54.86	63.61
OS-SYMPHONY w/ GPT-5	100	**79.17**	65.73	**67.76**	69.23	**57.98**	**65.84**

Table 1: Main results of OS-SYMPHONY on OSWorld. ♣ represents the result reproduced by us, others are sourced from the official leaderboard. Bold highlights the best performance, and underline denotes the runner-up.

2024), we exclude the 8 Google Drive tasks, yielding a final set of 361 tasks. To further probe cross-platform generalization, we incorporate WindowsAgentArena (Bonatti et al., 2024) and MacOSArena (Wang et al., 2025d) (see Appendix B.2). As extensions of the OSWorld paradigm, these benchmarks assess framework robustness in both cross-platform software consistency and system-specific configurations.

Baselines. We compare OS-SYMPHONY against leading general models, specialist native agents and agentic frameworks, highlighting Agent S3 (Gonzalez-Pumariega et al., 2025) and CoAct-1 (Song et al., 2025) as primary baselines, as they share a similar action space, characterized by the utilization of a coder for task execution.

Implementation Details. We select distinct based VLMs, encompassing both proprietary and open-source models across varying capability levels: GPT-5 (OpenAI, 2025b), GPT-5-Mini (OpenAI, 2025b) and Qwen3-VL series (Bai et al., 2025a). In our main experiments, we employ a single VLM to drive all core roles within OS-SYMPHONY, including Orchestrator, RMA, Searcher, and Coder. For grounding, we leverage UI-TARS-1.5-7B (Qin et al., 2025) as the General Grounder and EasyOCR[3] as the OCR Grounder. Regarding hyperparameters, we set the VLM temperature to 0.1 to maximize generation stability, with a maximum context window of 8 `1920x1080` images. Additionally, the EasyOCR width threshold is configured to 0.1 to ensure precise word-level recognition.

[3] https://github.com/JaidedAI/EasyOCR

4.2 Main Results

OSWorld. As shown in Tab. 1, OS-SYMPHONY with GPT-5 establishes a new SOTA with 63.61% (50-step) and 65.84% (100-step), surpassing the primary baseline (100-step Agent S3 w/ GPT-5) by ∼3%, demonstrating the effectiveness of our framework. Notably, in the *workflow* domain, which involves interactions across multiple applications, OS-SYMPHONY achieves a more substantial gain, outperforming the runner-up Agent S3 by 7%. This strong performance in long-horizon tasks is largely attributable to the RMA. With a streamlined yet robust design, the RMA acts as a critical safeguard against error accumulation, promoting long-term temporal stability and overall system resilience.

Furthermore, we reproduced Agent S3 using Qwen3-VL-32B-Instruct and GPT-5-Mini for a direct comparison. The results indicate that OS-SYMPHONY yields a gain of 8% with Qwen3-VL, with the improvement expanding to 10% when using GPT-5-Mini. Notably, these gains are more pronounced in models with relatively lower reasoning

Method	Step	Avg.(%)
Qwen3-VL-32B (2025a)	50	31.7
UI-TARS-1.5-7B (2025)	50	42.1
UI-TARS-2 (2025a)	50	50.6
Agent S3 w/ GPT-5 (2025)	50	54.1
Agent S3 w/ GPT-5 (2025)	100	56.6
OS-SYMPHONY w/ Qwen3-VL-32B-I.	50	45.3
OS-SYMPHONY w/ GPT-5-Mini	50	62.2
OS-SYMPHONY w/ GPT-5	50	**63.5**

Table 2: Main results of OS-SYMPHONY on WindowsAgentArena.

capacities. We attribute this to our *Visual-Centric Search as a Tool* design, which effectively compensates for the knowledge deficits inherent in smaller models. For instance, under a 50-step limit, the GPT-5-Mini variant invoked the Searcher 34 times more frequently than its GPT-5 counterpart. While stronger models leverage vast parametric knowledge to solve tasks directly, weaker models rely on our framework's search capabilities to bridge the information gap.

These results also highlight the exceptional cost-effectiveness of our approach. Specifically, OS-SYMPHONY leveraging GPT-5-Mini exhibits a marginal performance delta of only 3% compared to Agent S3 (driven by the significantly more powerful GPT-5), thereby attaining competitive efficacy with substantial cost reduction. Besides, when applied to open-source models, OS-SYMPHONY yields remarkable improvements: it achieves relative improvements of 45% and 23% for Qwen3-VL-32B-Instruct and Qwen3-VL-32B-Thinking, respectively, over their vanilla counterparts. This underscores a critical insight: OS-SYMPHONY not only establishes new SOTA standards but also democratizes advanced agentic capabilities, enabling smaller models to deliver competitive performance through a cohesive framework.

WindowsAgentArena. As shown in Tab. 2, OS-SYMPHONY establishes a new SOTA on WindowsAgentArena, achieving 63.5% with GPT-5 under a 50-step limit. This performance surpasses the 50-step and 100-step Agent S3 baselines by 9.4% and 6.9%, respectively. Remarkably, even the GPT-5-Mini variant demonstrates superior efficiency, exceeding the 100-step Agent S3 (GPT-5) by 5.6%. Furthermore, the framework exhibits strong robust adaptability across model scales. Specifically, when configured with Qwen3-VL-32B-Instruct, OS-SYMPHONY attains a success rate of 45.3%; while this trails the specialist UI-TARS-2, it represents a substantial 13.6% improvement over the vanilla baseline. These results indicate that our tailored designs not only accommodate models of varying scales but also enable effective generalization to distinct OS-level characteristics. Detailed results and analysis are provided in Appendix B.2.

Method	Success Rate(%)			Token(k)	Step
	Daily	Workflow	Avg.		
Combination Ablation					
w/o Search, Refl.	49.60	37.33	51.90	471	18.4
Search Ablation					
w/o Search	50.65	**48.06**	53.78	801	14.7
+) Unimodal	56.10	39.30	54.81	900	15.2
+) Multimodal (Ours)	**61.86**	47.37	**58.05**	900	15.2
Reflection & Memory Ablation					
w/o Reflection	60.20	39.23	54.38	535	18.5
+) Refl. w/ STM.	56.10	39.49	54.01	843	14.9
+) Refl. w/ LTM.(Ours)	**61.86**	**47.37**	**58.05**	900	15.2

Table 3: Ablation study on OSWorld. Experiments utilize GPT-5-Mini and UI-TARS-1.5-7B with a 50-step limit. STM: Short-Term Memory ("Last-K turns"); LTM: Long-Term Memory (our method).

4.3 Ablation Study

In this section, we conduct a comprehensive ablation analysis focusing on the core contributions of our framework: the Searcher and RMA, with quantitative results presented in Tab. 3.

For Unimodal Search, we adopt an inner-loop search paradigm consistent with our Multimodal Search, utilizing `search` and `parse` tools via SearXNG[4] and Crawl4AI (UncleCode, 2024). Our approach consistently outperforms all ablation baselines overall. Notably, the *Daily* domain, which inherently demands extensive external knowledge, benefits significantly from the search module. Specifically, Multimodal Search achieves substantial relative gains of 22.1% and 10.3% over the w/o Search and Unimodal Search baselines, respectively. Further, manual trajectory inspection confirmed that Multimodal Search performance aligns with expectations.

The *Workflow* domain typically involves cross-application, long-horizon tasks that demand robust long-term memory storage and comprehension, rendering the RMA module pivotal. Compared to the *Refl. w/ STM* paradigm and the *w/o Reflection* setting, our RMA delivers substantial improvements of approximately 20.0% and 20.7%, respectively. This result strongly validates the efficacy of abstract trajectory memory with granular error reflection in handling complex interactions. Notably, the *w/o Reflection* baseline marginally outperforms *Refl. w/ STM* while reducing token consumption by

[4] https://github.com/searxng/searxng

Model	Success Rate(%)		Cost($)
	Workflow	Avg.	
Different Based VLMs w/ UI-TARS-1.5-7B			
Claude-Sonnet-4.5 (2025b)	**57.21**	-	≈ 500
GPT-5	54.86	63.61	≈ 150
GPT-5-Mini	47.37	58.05	≈ 30
Qwen3-VL-32B-Inst.	31.24	46.86	0
Different Grounders w/ Qwen3-VL-32B-Instruct			
UI-TARS-1.5-7B	31.24	46.86	0
GTA1-32B (2025a)	28.54	46.32	0
ScaleCUA-32B (2025b)	31.76	45.76	0
Holo2-30B-A3B (2025)	24.45	43.92	0

Table 4: Impact of based VLMs and grounders configurations on OSWorld (50-step limit). 'Cost' represents the total expenditure for the *Workflow* domain, where $0 denotes local deployment of open-source models.

∼36.5%. This suggests that naive Last-K reflection may be ineffective or even detrimental, implying that omitting reflection entirely is preferable to deploying a suboptimal implementation.

Finally, regarding step efficiency, we observe that ablating RMA results in an increase of 3.3 steps per task. This underscores the reflection module's capacity for timely error identification and rectification, thereby preventing futile exploration and streamlining task completion.

4.4 Discussion

Impact of Based VLMs and Grounders. This section analyzes the sensitivity of our framework to different based VLMs and grounding models. Note that Claude-Sonnet-4.5 was tested only on the *Workflow* domain due to high inference costs.

Tab. 4 reveals a strong correlation between model scale and performance. Claude-Sonnet-4.5 achieves the highest *Workflow* score (57.21%) but comes with a steep price tag (∼$500). In contrast, cost-efficient GPT-5-Mini offers a compelling balance: it trails GPT-5 by only 5% on average while reducing costs by ∼80%. On the other hand, the performance remains stable across different grounding models, underscoring the robustness of OS-SYMPHONY. This efficacy is rooted in our collaborative architecture, which delegates fine-grained tasks to the Coder, thereby mitigating dependency on the Grounder's specific proficiency.

Key Insights. Finally, we present three key insights derived from our experiments:

The Granularity Gap in Visual Perception. While generally robust, the RMA falters against subtle visual nuances. Current VLMs often fail to resolve fine-grained cues like highlighting or overlapping windows. This *perceptual blindness* leads the RMA to issue false positive errors, paradoxically allowing the *w/o RMA* baseline to outperform the full framework in visually complex domains (see Appendix C.2 for illustrative cases). Future advancements must pivot towards targeted prompt engineering or enhanced image post-processing to bridge this granularity gap.

Bottlenecks in the Planner-Worker Paradigm. Precise textual articulation of visual affordances (*e.g.*, "boundaries") is challenging, causing information bottlenecks between the Orchestrator and Grounder. We argue that end-to-end native CUAs are essential to resolve this by eliminating textual abstraction. Future work should pivot towards hybrid paradigms, integrating native CUA capabilities to break through the ceiling of current paradigm.

Stochasticity: Volatility *vs.* Latent Potential. Despite strict control over prompts and temperature, we observe significant task-level volatility. While detrimental to deployment stability, this variance masks high latent competence: aggregating results via Pass@5 boosts performance to 79.40%, significantly surpassing the human baseline (72.4%). This suggests that the model *can* solve the tasks but lacks consistency. Harnessing this "volatility" remains a critical frontier for future framework design.

5 Conclusion

In this work, we present OS-SYMPHONY, a holistic Computer-Using Agent (CUA) framework containing an Orchestrator that synergistically coordinates specialized modules, to address the two challenges of long-horizon robustness and domain generalization. To mitigate the visual loss inherent in extended workflows, our Reflection-Memory Agent employs a milestone-driven strategy for granular memory curation, enabling retrospective auditing to rectify errors such as intent drift and loops. Besides, we employ the Versatile Tool Agents featuring a *Multimodal Searcher* that transcends text-based retrieval limitations by adopting an active *SeeAct* paradigm, synthesizing high-fidelity, visually aligned tutorials for unseen environments. Extensive experiments confirm that OS-SYMPHONY not only achieves state-of-the-art performance across diverse operating systems but proves that complex problems can be effectively solved using open-source VLMs. We hope our insights will provide a resilient blueprint for future real-world CUAs.

Limitations

Despite the robust performance of OS-SYMPHONY across major desktop environments, several limitations remain.

Environmental Generalization. Our current evaluation is strictly confined to desktop ecosystems. The framework's adaptability to mobile platforms, such as Android and iOS, remains unverified due to the necessity of distinct action space adaptations for mobile interfaces. Consequently, achieving full cross-platform universality remains a subject for future exploration.

Structural Complexity and Efficiency. The multi-agents system introduces inherent overhead. Although we employ strategies to mitigate error accumulation, the extensive inter-agent interactions result in high token consumption and significant latency. Specifically, our execution speed is tens of times slower than human performance, which currently precludes real-time deployment. Future iterations may address this via dynamic "fast and slow" reasoning mechanisms or simplified architectures.

Ethical Considerations

The development of autonomous Computer-Using Agents (CUAs) introduces significant ethical and safety responsibilities. In this work, we prioritized operational safety by conducting all evaluations within strictly isolated, sandboxed virtual environments (*e.g.*, Docker containers and virtual machines). This isolation ensures that the agent's exploratory actions can't inadvertently damage host systems or access unauthorized external networks during the research phase.

However, transitioning from controlled benchmarks to real-world deployment necessitates a rigorous re-evaluation of system security and user privacy. Since visual-centric agents like OS-SYMPHONY inherently process continuous streams of screenshots, they possess the capability to "see" sensitive personal identifiable information displayed on the screen. Consequently, deploying such agents requires the implementation of strict, granular permission controls and robust data sanitization protocols. Users must retain absolute authority to define the agent's operational boundaries, ensuring that sensitive applications (*e.g.*, banking, private messaging) remain inaccessible unless explicitly authorized.

Furthermore, the robust automation capabilities demonstrated by our framework carry inherent dual-use risks. While designed to enhance human productivity, these systems could potentially be exploited for auto-execution of malicious workflows if not properly safeguarded. We posit that the advancement of CUAs' capabilities must proceed in lockstep with the development of "Safety by Design" principles. Future research must focus not only on increasing success rates but also on embedding deep alignment mechanisms, ensuring that agents remain reliable, transparent, and strictly adherent to human ethical standards in open-ended digital environments.

Information About Use Of AI Assistants

In the preparation of this manuscript, we utilized LLMs to refine the writing, including grammar and spelling corrections.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025a. Agent S: An Open Agentic Framework that Uses Computers Like a Human. In *International Conference on Learning Representations (ICLR)*.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025b. Agent s2: A compositional generalist-specialist framework for computer use agents. *Preprint*, arXiv:2504.00906.

Anthropic. 2025a. Claude opus 4 & claude sonnet 4 system card. https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf. Accessed: 2025-08-04.

Anthropic. 2025b. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5. Accessed: 2025-09-30.

Sonnet Anthropic. 2025c. Claude 3.7 sonnet system card.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and 1 others. 2025a. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025b. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, and 1 others. 2024. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*.

Weihua Cheng, Ersheng Ni, Wenlong Wang, Yifei Sun, Junming Liu, Wangyu Shen, Yirong Chen, Botian Shi, and Ding Wang. 2025. Mga: Memory-driven gui agent for observation-centric interaction. *arXiv preprint arXiv:2510.24168*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

H Company. 2025. Holo2 - open foundation models for navigation and computer use agents.

Horia Cristescu, Charles Park, Trong Canh Nguyen, Sergiu Talmacel, Alexandru-Gabriel Ilie, and Stefan Adam. 2025. Ui-cube: Enterprise-grade computer use agent benchmarking beyond task accuracy to operational reliability. *arXiv preprint arXiv:2511.17131*.

Aarash Feizi, Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Kaixin Li, Rabiul Awal, Xing Han Lù, Johan Obando-Ceron, Juan A Rodriguez, Nicolas Chapados, and 1 others. 2025. Grounding computer use agents on human demonstrations. *arXiv preprint arXiv:2511.07332*.

Tianyu Fu, Anyang Su, Chenxu Zhao, Hanning Wang, Minghui Wu, Zhe Yu, Fei Hu, Mingjia Shi, Wei Dong, Jiayao Wang, and 1 others. 2025. Mano technical report. *arXiv preprint arXiv:2509.17336*.

Gemini Team. 2025. Gemini deep research. Accessed: 2025.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*.

Grok Team. 2025. Grok-3 deeper search. Accessed: 2025.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, and 1 others. 2025a. Seed1. 5-vl technical report. *arXiv preprint arXiv:2505.07062*.

Liangxuan Guo, Bin Zhu, Qingqian Tao, Kangning Liu, Xun Zhao, Xianzhe Qin, Jin Gao, and Guangfu Hao. 2025b. Agentic lybic: Multi-agent execution system with tiered reasoning and orchestration. *arXiv preprint arXiv:2509.11067*.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, and 1 others. 2025. Glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, and 1 others. 2025a. Os agents: A survey on mllm-based agents for general computing devices use. *arXiv preprint arXiv:2508.04482*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, and 28 others. 2025b. Memory in the age of ai agents. *Preprint*, arXiv:2512.13564.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. 2025. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. *arXiv preprint arXiv:2510.24563*.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. 2025. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. *arXiv preprint arXiv:2508.14040*.

Baixuan Li, Jialong Wu, Wenbiao Yin, Kuan Li, Zhongwang Zhang, Huifeng Yin, Zhengwei Tao, Liwen Zhang, Pengjun Xie, Jingren Zhou, and Yong Jiang. 2025a. Nested browser-use learning for agentic information seeking. *Preprint*, arXiv:2512.23647.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Ji-Rong Wen, Yuan Lu, and 1 others. 2025b. Deepagent: A general reasoning agent with scalable toolsets. *arXiv preprint arXiv:2510.21618*.

Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, and Fei Huang. 2025a. Pc-agent: A hierarchical multi-agent collaboration framework for complex task automation on pc. *arXiv preprint arXiv:2502.14282*.

Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, and 1 others. 2025b. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*.

Kai Mei, Jiang Guo, Shuaichen Chang, Mingwen Dong, Dongkyu Lee, Xing Niu, and Jiarong Jiang. 2025. R-wom: Retrieval-augmented world model for computer-use agents. *arXiv preprint arXiv:2510.11892*.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, and 1 others. 2025. Gui agents: A survey. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22522–22538.

OpenAI. 2025a. Computer-using agent: Introducing a universal interface for ai to interact with the digital world. https://openai.com/index/computer-using-agent. 2025a.

OpenAI. 2025b. Openai gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf. Accessed: 2025-08-07.

OpenAI. 2025c. Openai o3 and o4-mini system card. System card, OpenAI. 2025b.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Chenrui Shi, Zedong Yu, Zhi Gao, Ruining Feng, Enqi Liu, Yuwei Wu, Yunde Jia, Liuyu Xiang, Zhaofeng He, and Qing Li. 2025. Gui knowledge bench: Revealing the knowledge gap behind vlm failures in gui tasks. *arXiv preprint arXiv:2510.26098*.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, and 1 others. 2025. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, and 1 others. 2024a. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, and 1 others. 2024b. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, and 1 others. 2024c. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*.

Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, and 1 others. 2025a. Scienceboard: Evaluating multimodal autonomous agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025b. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*.

Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, and 1 others. 2025. Webshaper: Agentically data synthesizing via information-seeking formalization. *arXiv preprint arXiv:2507.15061*.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, and 1 others. 2025. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*.

Shizuo Tian, Hao Wen, Yuxuan Chen, Jiacheng Liu, Shanhui Zhao, Guohong Liu, Ju Ren, Yunxin Liu, and Yuanchun Li. 2025. Agentprog: Empowering long-horizon gui agents with program-guided context management. *arXiv preprint arXiv:2512.10371*.

UncleCode. 2024. Crawl4ai: Open-source llm friendly web crawler & scraper. https://github.com/unclecode/crawl4ai.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, and 1 others. 2024. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, and 1 others. 2025b. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, and 23 others. 2025c. Opencua: Open foundations for computer-use agents. *Preprint*, arXiv:2508.09123.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, and 1 others. 2025d. Mmbench-gui: Hierarchical multi-platform evaluation framework for gui agents. *arXiv preprint arXiv:2507.19478*.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

Zihao Wang, Xujing Li, Yining Ye, Junjie Fang, Haoming Wang, Longxiang Liu, Shihao Liang, Junting Lu, Zhiyong Wu, Jiazhan Feng, and 1 others. 2025e. Game-tars: Pretrained foundation models for scalable generalist multimodal game agents. *arXiv preprint arXiv:2510.23691*.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, and 1 others. 2025a. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and 1 others. 2025b. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. 2025c. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. *arXiv preprint arXiv:2505.20660*.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, and 1 others. 2025d. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*.

Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Qiushi Sun, Zhaoyang Liu, Zhoumianze Liu, Yu Qiao, Xiangyu Yue, Zun Wang, and 1 others. 2025e. Os-oracle: A comprehensive framework for cross-platform gui critic models. *arXiv preprint arXiv:2512.16295*.

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

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024b. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025a. Scaling computer-use grounding via user interface decomposition and synthesis. *Preprint*, arXiv:2505.13227.

Tianbao Xie, Mengqi Yuan, Danyang Zhang, Xinzhuang Xiong, Zhennan Shen, Zilong Zhou, Xinyuan Wang, Yanxu Chen, Jiaqi Deng, Junda Chen, Bowen Wang, Haoyuan Wu, Jixuan Chen, Junli Wang, Dunjie Lu, Hao Hu, and Tao Yu. 2025b. Introducing osworld-verified. *xlang.ai*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Ran Xu, Kaixin Ma, Wenhao Yu, Hongming Zhang, Joyce C Ho, Carl Yang, and Dong Yu. 2025. Retrieval-augmented gui agents with generative guidelines. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17877–17886.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. 2024a. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024b. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, and 1 others. 2025a. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*.

Yuhao Yang, Zhen Yang, Zi-Yi Dou, Anh Nguyen, Keen You, Omar Attia, Andrew Szot, Michael Feng, Ram Ramrakhya, Alexander Toshev, and 1 others. 2025b. Ultracua: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*.

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

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025a. Mobile-agent-v3: Foundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, and 1 others. 2025b. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and 1 others.

2025. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*.

Christoph Zauner. 2010. Implementation and benchmarking of perceptual image hash functions.

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

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, and 1 others. 2024. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*.

Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, and 1 others. 2025b. Ufo2: The desktop agentos. *arXiv preprint arXiv:2504.14603*.

Junlei Zhang, Zichen Ding, Chang Ma, Zijie Chen, Qiushi Sun, Zhenzhong Lan, and Junxian He. 2025c. Breaking the data barrier–building gui agents through task generalization. *arXiv preprint arXiv:2504.10127*.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025d. A survey on the memory mechanism of large language model-based agents. *ACM Trans. Inf. Syst.*, 43(6).

Henry Hengyuan Zhao, Kaiming Yang, Wendi Yu, Difei Gao, and Mike Zheng Shou. 2025. Worldgui: An interactive benchmark for desktop gui automation from any starting point. *arXiv preprint arXiv:2502.08047*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, pages 61349–61385. PMLR.

A Details of OS-SYMPHONY

In this section, we provide a comprehensive elaboration on the implementation details of our OS-SYMPHONY framework to facilitate a deeper understanding. So, how exactly is this **Symphony** composed?

A.1 Task Definition

The interaction process of Computer-Using Agents (CUAs) can be modeled as a Partially Observable Markov Decision Process (POMDP), defined by the tuple $(S, A, O, T, \mathcal{O})$. Here, S represents the set of environmental states; A denotes the finite set of executable actions available to the agent; and O represents the set of observations the agent can receive. The state transition function $T(S'|S, A) \rightarrow [0, 1]$ defines the probability of transitioning to a new state given the current state and the action. The observation function $\mathcal{O}(O|S, A) \rightarrow [0, 1]$ defines the probability of receiving a specific observation given a state and action. For brevity, the discount factor and the reward function are omitted in this formulation. Addressing the partial observability, the state of a GUI environment consists of a complex composition of numerous GUI elements, resulting in an infinite state space. However, for a purely vision-based approach, the observation of the GUI environment is a screenshot. For an RGB screenshot with dimensions $H \times W$, the possible observations are limited to $H \times W \times 3$ variations. Consequently, there is an inherent information compression from the state space to the observation space. The following discussion focuses on the observation o.

To formalize the task, we introduce the task goal (user instruction) $\mathcal{I}$, where the agent receives observations from the environment and executes corresponding actions to complete the task. To further enhance the agent's reasoning capabilities and support more complex decision-making processes (Yao et al., 2023), a "thought" component t_i is incorporated prior to each action a_i. This thought process serves as a critical intermediate step that analyzes the current situation and historical actions, ensuring the rationality and intentionality of each decision. Consequently, the entire task process can be formalized as a trajectory:

$$\mathcal{I}, (o_1, t_1, a_1), (o_2, t_2, a_2), \ldots, (o_n, t_n, a_n) \quad (4)$$

In a POMDP, the agent can only infer the current true state through observations; thus, a single

observation is insufficient to fully depict the environmental state. To address this, an implicit Belief State can be constructed by aggregating historical information to more accurately approximate the current true state. The update process of this belief state is essentially an extension of the Markov property. Although a single observation may fail to satisfy Markovian conditions, the aggregation of historical information allows the belief state to be viewed as adhering to Markov properties.

In an ideal CUA architecture, to achieve optimal decision-making, the model should theoretically access and process the entire interaction history from the inception of the task. This complete historical record contains all contextual cues regarding environmental evolution and serves as the foundation for constructing a precise belief state. Consequently, conditioned on the task instruction and the full history preceding the current step, the model iteratively predicts the thought t_i and action a_i until the action $\{\texttt{done}, \texttt{fail}\}$ appears. This probabilistic model is formalized as:

$$P(t_i, a_i | \mathcal{I}, (o_j, t_j, a_j)_{j=1}^{i-1}, o_i) \tag{5}$$

While theoretically optimal, the complete history model in Eq. 5 faces practical limitations in long-horizon GUI tasks. Despite extended LLM contexts ($\geq 128K$), directly incorporating full multimodal histories incurs significant drawbacks: (1) **Efficiency and Reliability Issues**: Processing dozens of screenshots and textual histories increases computational overhead and may induce hallucinations due to information overload; (2) **Historical Redundancy**: Many intermediate observations (*e.g.*, failed attempts or trivial clicks) provide diminishing informational value for future decisions.

Thus, effective **context compression** becomes crucial—filtering redundant content while preserving decision-critical information. We formalize this as finding an optimal compression function $\mathcal{C}$ that transforms raw history $\mathcal{H}_{1:i-1}$ into condensed representation $\tilde{\mathcal{H}}_{1:i-1}$, maximizing the likelihood of correct actions:

$$\max_{\mathcal{C}} \mathbb{E}\left[P\left(t_i^*, a_i^* \mid \mathcal{I}, \mathcal{C}(\mathcal{H}_{1:i-1}), o_i\right)\right] \tag{6}$$

This yields the practical decision model:

$$P(t_i, a_i \mid \mathcal{I}, \tilde{\mathcal{H}}_{1:i-1}, o_i) \tag{7}$$

The design of $\mathcal{C}$ is therefore central to enabling efficient long-horizon GUI task performance. In this paper, we dedicate our research to this pivotal challenge, presenting in-depth explorations into the optimization of such compression mechanisms.

A.2 More Details of OS-SYMPHONY

We provide further implementation details of the modules that were omitted in Section 3 due to space constraints. The following paragraphs elaborate on the Orchestrator, Reflection-Memory Agent, General Grounder, OCR Grounder, Searcher, and Coder.

Orchestrator Agent. Serving as the cognitive core of OS-SYMPHONY, the Orchestrator manages short-term memory to iteratively generate actions, as defined in Eq. 7. To balance context retention with efficiency, it implements the compression function $\mathcal{C}$ by synthesizing a sliding window of recent interactions with high-level semantic insights. Formally, the condensed history is constructed as:

$$\tilde{\mathcal{H}}_{1:i-1} = \{(o_j, t_j, a_j)\}_{j=i-K-1}^{i-1} \cup \{\mathcal{R}_i\}, \tag{8}$$

where the context combines the raw trajectory of the most recent K steps with reflections on the immediate previous execution and the retrieval of relevant procedural knowledge for the current step provided by the Reflection-Memory Agent ($\mathcal{R}_i$). This hybrid memory structure allows the Orchestrator to maintain minimal context length without sacrificing the critical semantic cues necessary for precise decision-making.

Reflection-Memory Agent. Serving as a key component of our symphony and specifically engineering for long-term memory management, RMA is designed to provide precise feedback for the subsequent step. It distills the current trajectory into an abstract representation comprising multiple milestone screenshots and a comprehensive history of step-wise transitions. Furthermore, it extracts critical memories to construct a procedural knowledge base. In this subsection, we further elaborate on the two corresponding auxiliary detection methods with defined core error types, as shown in Tab. 5.

Step Summary. Before the Orchestrator's decision making process based on the current observation o_t, we execute a retrospective analysis of the last interaction, formalized as eq. 2. To bolster the VLM's perception, we introduce $\tilde{o}_{t-1}$, a "zoom-in" augmentation applied solely when a_{t-1} involves coordinates; this consists of a 400-pixel radius crop centered on the element's coordinate, overlaid with a eye-catching red visual marker.

Error Type	Description	Aux. Detection	Method Desc.	Type	Rel. Tool
GUI Error	Failures at the execution level where the intended action (e.g., click, type) is unsuccessful.	Step Summary	A low-level analysis correlating actions with pre- and post-transition screenshots.	VLM	Grounder
Lack of Tutorial	The agent performs technically correct operations but lacks procedural logic to advance the workflow.	Loop Detection	A loop detection algorithm evaluating similarity metrics across sequential actions and visual states.	Rule	Searcher
Code Error	Post-execution discrepancies identified during verification. Output fails to align with instructions.	–	–	–	–
Other Error	Deviations not covered by other categories, such as drifting from the primary goal, factual errors, or hallucinations.	–	–	–	–

Table 5: Definitions of Error Types, Auxiliary Methods, and Relevant Tools.

Notably, we refrain from augmenting o_t similarly, as the visual consequences of a GUI interaction often manifest in regions distinct from the initial actuation coordinates. Ultimately, this low-level verification focuses on single-step fidelity, providing s_{t-1} as a critical input for the step behavior history and a decisive signal for RMA's detection of *GUI Error*.

Loop Detection. While chaotic and disorganized trajectory states are difficult to probe through rule-based methods, repetitive and cyclical trajectories are easily perceptible; therefore, we designed a similarity-rule-based loop detection algorithm:

$$D_{loop}(H, N) = \bigvee_{k=T-2N}^{1} \bigwedge_{j=0}^{N-1} \mathcal{M}(k+j, T-N+j), \quad (9)$$

where $\mathcal{M}(u, v) \triangleq \mathcal{S}_{img}(o_u, o_v) \wedge \mathcal{S}_{act}(a_u, a_v)$ denotes the joint similarity check for observation o and action a at time steps u and v. H represents the trajectory history, and N is the sliding window size (default set to 3). In practice, we scan k in reverse order (from $T-2N$ to 1) with early stopping: once $\bigwedge_{j=0}^{N-1} \mathcal{M}(k+j,\, T-N+j)$ holds, we terminate and return the most recent matched segment.

To implement the metric defined in Eq. 9, we prioritize high precision to ensure that feedback provided to the Memory Agent is factually grounded, adopting a strict "coarse-to-fine" matching protocol. For visual state similarity $\mathcal{S}_{img}$, we employ a cascaded verification strategy: a Perceptual Hashing (pHash) (Zauner, 2010) check first filters distinct states using a tight Hamming distance threshold (≤ 1), followed by a Structural Similarity Index (SSIM) (Wang et al., 2004) calculation with a high threshold (0.99) to confirm identity despite minor rendering artifacts. Action similarity $\mathcal{S}act$ is determined based on action semantics: coordinate-dependent actions (*e.g.*, *click*, *scroll*) require Euclidean distances within a relative threshold (5% of the screen diagonal) alongside matching discrete parameters; discrete actions (*e.g.*, *type*, *open*) demand exact argument matching; and natural language queries (*e.g.*, *search*) utilize Levenshtein distance with a high similarity threshold to tolerate minor phrasing variations. The search process iterates backwards from $k = T - 2N$ to identify the most recent historical interval $[k, k+N-1]$ identical to the current window. To ensure real-time performance, we adopt a space-for-time optimization strategy by caching image features, effectively reducing the computational complexity from $O(T \cdot N \cdot C_{img})$ to $O(T \cdot (N + C_{img}))$ by avoiding redundant image processing during the sliding window comparison, C_{img} represents the computational complexity of pHash and SSIM algorithm between two images. The loop detection algorithm prioritizes high precision, a successful loop identification provides a strong signal for RMA's detection of *Lack of Tutorial*.

General Grounder. Serving as the visual grounding engine, this VLM-based Grounder localizes UI elements by processing hybrid descriptions that integrate low-level visual cues (position, appearance) with high-level semantic context (functionality, instruction relevance). Based on our empirical evaluation of existing grounding models including UI-TARS-1.5-7B (Qin et al., 2025), ScaleCUA-32B (Liu et al., 2025b), Holo1.5-72B [5], Holo2-30B-

[5] https://www.hcompany.ai/blog/holo2

A3B (Company, 2025), GTA1-32B (Yang et al., 2025a) and GroundNext-7B (Feizi et al., 2025). UI-TARS-1.5-7B demonstrates the best performance among open-source models in desktop environments, followed by GTA1-32B and ScaleCUA-32B.

OCR-based Grounder. Serving as a strategic complement to the General Grounder, the OCR-based Grounder operates in synergy with the VLM Grounder to enhance element localization, aiming to address the General Grounder's limitations in resolving precise word-boundary coordinates. The workflow entails a word-level OCR scan that generates a structured table of `{text, id, bbox}`. This table is subsequently processed by the general VLM for semantic ID selection, enabling precise coordinate retrieval via index lookup. This approach effectively mitigates performance deficits in text-dense domains such as PowerPoint and Word. However, it represents a pragmatic compromise because current OCR models still lack the granularity for character-level localization, a capability not yet strictly demanded by existing benchmarks. Currently, resolving this limitation necessitates either using code tools for intrinsic file manipulation or rewriting large text blocks as a shortcut to bypass precise localization. Achieving a more human-like editing capability, however, requires further refining the General Grounder's native localization skills to handle such granularity.

Searcher. Serving as the core module for external knowledge retrieval, the Searcher employs a visual browsing strategy to navigate the open web and synthesize step-by-step tutorials. Crucially, we enforce a strict validity constraint: the agent is instructed to return a tutorial only when high relevance is guaranteed, defaulting to a `fail` state otherwise. This design prevents the contamination of the Orchestrator's context with lengthy or erroneous information that could disrupt downstream decision-making.

Coder. Serving as the core execution module for system-level tasks, the Coder interacts directly with the environmental CLI to execute Shell and Python code, excelling in file revision and configuration. To mitigate complexity, the Orchestrator delegates sub-tasks to the Coder, which follows a strict internal workflow involving file localization, content inspection, in-place modification via complete overwrites, verification and visualization. Following execution, a summary agent module provides an execution synopsis to the Orchestrator as a textual observation. While the Coder's GUI-free design ensures high efficiency during iterative processes, the resulting file modifications can manifest as abrupt GUI mutations imposing a high cognitive load, thereby necessitating a unified verification protocol across all agents(Coder, RMA, Orchestrator) to minimize side effects deviating from user instructions.

Ultimately, the designs of both the Searcher and Coder embody the principle of *Context Folding*. This paradigm entails offloading independent subtasks to isolated execution contexts to prevent polluting the Orchestrator's context window. By "folding" the detailed execution trajectory and results into a concise summary, we maintain a seamless logical flow for the Orchestrator. This encapsulation strategy offers a generalizable method for managing context in complex agentic systems.

A.3 Action Space

Designing an optimal action space is of paramount importance for CUAs, and its impact on task success rates and execution efficiency may even outweigh that of the system architecture itself. Building upon the insights and foundational methodologies of the AgentS series work (Gonzalez-Pumariega et al., 2025), we have formulated our design based on the following core principles:

- **Cross-Platform Abstraction.** For cross platform CUAs, it is essential to establish a set of intermediate-level actions, which are subsequently mapped to platform-specific executable primitives (always be `PyAutoGUI`[6]).
- **Conciseness and Generality.** For general-purpose CUAs, the action space must be concise, pragmatic, and universally applicable. Redundant actions inevitably introduce additional context overhead and cognitive load, whereas overly specialized actions, such as manually constructed MCP tools, impose severe constraints on the agent's operational versatility.

Consequently, our action space is detailed in Tab. 6. We categorize the action space into three distinct types: GUI actions, proprietary actions, and special actions. Notably, we have excluded application-specific primitives, such as

[6] https://pyautogui.readthedocs.io/

Action	Category	Parameter Specification
click	GUI (General Grounding)	**Format:** `[desc,num_clicks,button,hold_keys]` **Details:** Target element description; clicks number; button to click; keys to hold.
type	GUI (General Grounding)	**Format:** `[desc,text,overwrite,enter,terminal]` **Details:** Target element description; text content; overwrite flag(bool); press enter after typing(bool); terminal flag(bool).
scroll	GUI (General Grounding)	**Format:** `[desc,clicks,shift]` **Details:** Target element description; clicks (+up/-down); shift for horizontal scroll(bool).
drag_and_drop	GUI (General Grounding)	**Format:** `[start_desc,end_desc,hold_keys]` **Details:** Descriptions for start/end locations; keys to hold during drag.
highlight_text_span	GUI (OCR Grounding)	**Format:** `[start_phrase,end_phrase,button]` **Details:** Unique anchor phrase; button to hold.
locate_cursor	GUI (OCR Grounding)	**Format:** `[phrase,pos,text]` **Details:** Unique anchor phrase; position(start/end); optional text to insert immediately.
hotkey	GUI (No Grounding)	**Format:** `[keys]` **Details:** List of keys to press in combination (e.g., ['ctrl', 'c']).
hold_and_press	GUI (No Grounding)	**Format:** `[hold_keys,press_keys]` **Details:** Keys to hold down while pressing a sequence of other keys.
open	GUI (No Grounding)	**Format:** `[app_or_filename]` **Details:** Name of the application or file to open.
call_search_agent	Proprietary	**Format:** `[query]` **Details:** A "How to" question targeting a specific tutorial (e.g., "How to apply filters in Excel?").
call_code_agent	Proprietary	**Format:** `[task]` **Details:** A self-contained goal executable via code (e.g., data analysis, file processing).
wait	Special	**Format:** `[time]` **Details:** Time to wait in seconds.
done	Special	**Format:** `[]` **Details:** Signals successful completion of the entire task.
fail	Special	**Format:** `[]` **Details:** Signals that the task is impossible to complete.

Table 6: Action Space of OS-Symphony.

set_cell_value for LibreOffice Calc, from our action space. While precise table localization remains a challenge for current Grounders, our extensive testing suggests that call_code_agent serves as a superior alternative for such granular tasks. Extending beyond specific benchmarks, our design of action space and overall framework enables high performance without relying on defining specialized actions for every software feature. Nevertheless, the design of optimal action spaces, including the action design and the action parameters' design, remains an open question worthy of further investigation.

B More Results

In this section, we first show the more results on OSWorld, then present the comprehensive evaluation results for both WindowsAgentArena and MacOSArena. We also explored the instruction rewriting method followed by an in-depth analysis of various statistical metrics.

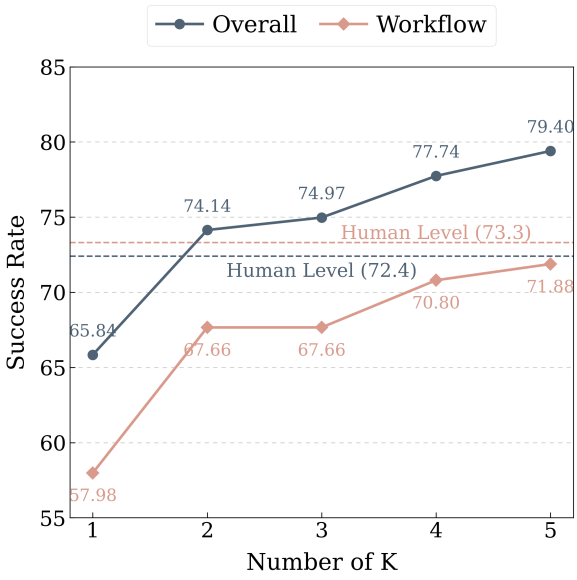


Figure 4: The Pass@K results on OSWorld. All experiments are carried out with GPT-5 and 100 steps limit.

B.1 More Results on OSWorld

Results with Pass@K. As shown in Fig. 4, OS-SYMPHONY surpasses human performance (72.4%) at Pass@2 (74.14%) and achieves a success rate approaching 80% at Pass@5 (79.4%) on OSWorld. To explore the performance limits, we incrementally increased the temperature of both the Orchestrator and Reflection-Memory Agent by 0.1 at each pass, reaching a 0.5 increase at Pass@5 to encourage diverse solution attempts. This demonstrates, first, that our framework has a high performance ceiling and can attain excellent results through such test-time scaling. Second, consistent with our analysis, zero-shot agentic frameworks exhibit substantial stochasticity on GUI tasks, which is evident in both our experimental results and qualitative observations. Future work should therefore prioritize improving deployment-time stability.

Method	Workflow
w/ rewritten instruction	51.09
w/o rewritten instruction(Ours)	54.86

Table 7: Results in OSWorld with GPT-5+UI-TARS-1.5-7B and 50 steps limit.

Results with Instruction Rewriting. Given the colloquial ambiguity inherent in current benchmark's user instructions and their frequent misalignment with the initial visual state (*e.g.*, omitting specific target applications), we initially explored an instruction rewriting mechanism to mitigate task deviation. Specifically, we employed a VLM that accepts the raw user instruction and the initial screenshot to generate a refined instruction via a predefined prompt. The purpose is to produce professional, concise instructions that are visually grounded. For instance:

- **Original:** I need to include the experiment results from ~/Documents/awesome-desktop/expe-results.xlsx into the currently writing report. Specifically, extract the results of GPT-4 and insert a table into the "Main Results" section of my report.
- **Rewritten:** Insert a table into the "Main Results" section of the open document awe_desk_env.docx in LibreOffice Writer containing the GPT-4 experiment results extracted from ~/Documents/awesome-desktop/expe-results.xlsx.

However, as shown in Tab. 7, preliminary experiments indicated that performance improvements were negligible. Furthermore, we determined that altering the test instructions might compromise the integrity of the evaluation (i.e., potential data leakage or simplification). Consequently, we pivoted to a keeping first image strategy combined with Reflection-Memory Agent (RMA) and prompt engineering. Nevertheless, we maintain that instruction rewriting remains an indispensable component for robust real-world deployment.

Method	Avg.
w/o Coder	57.42
w/ Coder(Ours)	63.61

Table 8: Results in OSWorld with GPT-5+UI-TARS-1.5-7B and 50 steps limit.

Impact of Coder. While the Coder is not the central contribution of our framework, we performed an ablation study to assess its individual contribution, as detailed in Tab. 8. For the "w/o Coder" setting, we disabled the `call_code_agent` action and eliminated all relevant prompts, constraining the model to rely solely on GUI interactions. The results reveal a performance degradation of approximately 6.2% in the absence of the Coder. This finding not only underscores the robustness of the Coder in handling tasks such as batch file processing and content editing, but also substantiates the necessity of a hybrid GUI-API paradigm for the advancement of CUAs.

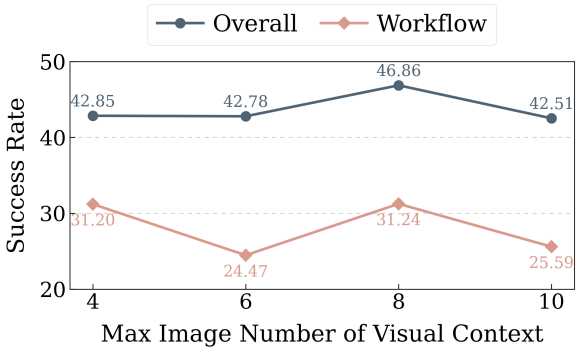


Figure 5: Discussion on the impact of varying the maximum number of images in agent's trajectory. All experiments are carried out with Qwen3-VL-32B-Instruct and UI-TARS-1.5-7B.

Impact of Visual Context Length. In this subsection, we investigate the sensitivity of performance to the maximum visual context length. As illustrated in Fig. 5, the quantity of retained images significantly influences framework efficacy. A sparsity of images fails to provide adequate context regarding interaction history, whereas an excessive visual load saturates the context window, degrading the model's reasoning capabilities due to information overload. This aligns with our earlier insights regarding the context-reasoning trade-off. Empirically, our results confirm that a cap of 8 images strikes the optimal balance, achieving a peak performance of 46.86%.

B.2 More Results on WindowsAgentArena & MacOSArena

WindowsAgentArena. The comprehensive results on WindowsAgentArena are presented in Tab. 9. Our analysis yields the following insights: First, the three backbone configurations of OS-SYMPHONY demonstrate progressively higher performance, which aligns with general expectations. In the particularly challenging Office domain, our framework achieves a top score of 54.76%. Next, the tasks in WindowsAgentArena are partially inherited from OSWorld, while the remainder are system-level software tasks. This benchmark's lack of complex multi-apps tasks may mean that our framework has not yet fully exploited its potential, despite already achieving SOTA performance, indicating room for further gains. Furthermore, the vanilla Qwen3-VL-32B-Instruct baseline shows relatively low metrics. Our case study suggests this is likely due to the model's insufficient training on Windows-specific data, hindering its adaptation to the Windows. Our framework effectively mitigates this limitation, delivering improvements across nearly every domain and achieving a relative increase of approximately 24.6% on average. Therefore, we reiterate that the OS-SYMPHONY framework provides substantial and consistent benefits for models of different strengths and scales.

MacOSArena. The comprehensive results on MacOSArena are presented in Tab. 10. Due to the cost constraint, we didn't use GPT-5 as the backbone. Our analysis yields the following insights: First, OS-SYMPHONY establishes a new SOTA. Even with the Qwen3-VL-32B-Instruct backbone, OS-SYMPHONY achieves 19.05%, surpassing all competing baselines. Furthermore, OS-SYMPHONY with GPT-5-Mini reaches a remarkable 46.03%, significantly outperforming other methods, suggesting that GPT-5-Mini is a highly cost-effective and capable choice for MacOS tasks. In stark contrast, existing strong baselines struggle severely on this benchmark, with some yielding 0% success rates. This highlights a critical generalization gap: models trained primarily on Linux or Windows fail to adapt to the MacOS environment. We attribute this to two factors: (1) Data scarcity for the MacOS domain (Liu et al., 2025b); and (2) Intrinsic UI challenges, such as the minute "traffic light" window controls (red/yellow/green buttons) which are difficult for models to locate, complicating window management and app

Method	Step	Success Rate(%)							
		Office	Web	Sys.	Code	Media	Util.	Inf.	Avg.
Qwen3-VL-32B-Inst.	50	19.05	49.66	54.17	21.05	42.19	25.00	0.00	31.68
UI-TARS-1.5-7B	50	-	-	-	-	-	-	-	42.10
UI-TARS-2	50	-	-	-	-	-	-	-	50.60
Agent S3 w/ GPT-5	50	-	-	-	-	-	-	-	54.10
Agent S3 w/ GPT-5	100	-	-	-	-	-	-	-	56.60
OS-SYMPHONY w/ Qwen3-VL-32B-Inst.	50	26.19	46.33	75.00	47.37	27.90	41.67	**69.23**	45.32
OS-SYMPHONY w/ Qwen3-VL-32B-Thinking	50	17.07	56.33	68.18	64.29	29.33	63.64	58.33	45.95
OS-SYMPHONY w/ GPT-5-Mini	50	42.86	**73.00**	**79.17**	**68.42**	48.66	66.67	**69.23**	62.15
OS-SYMPHONY w/ GPT-5	50	**54.76**	**73.00**	75.00	42.11	**70.09**	**75.00**	53.85	**63.45**

Table 9: Main results of OS-SYMPHONY on WindowsAgentArena which has 154 tasks. Office includes LibreOffice Writer and LibreOffice Calc tasks; Web (Web Browsing) includes Edge and Chrome tasks; Sys.(Windows System) includes File Explorer and Settings tasks; Code includes VSCode tasks; Media(Media & Video) includes VLC tasks; Util.(Windows Utilities) includes Notepad, Clock, Paint and WindowsCalc tasks; Inf.(Infeasible) includes 13 infeasible tasks.

Method	Step	Success Rate(%)		
		Single-Apps	Multi-Apps	Avg.
GPT-4o (2024)	50	3.57	0.00	1.59
Claude-3.7-Sonnet (2025c)	50	14.29	2.86	7.94
Aguvis-72B (2024b)	50	0.00	0.00	0.00
UI-TARS-1.5-7B (2025)	50	14.29	2.86	7.94
UI-TARS-72B-DPO (2025)	50	14.29	5.71	9.52
Qwen2.5-VL-72B (2025b)	50	7.14	0.00	3.17
Qwen3-VL-32B-Inst. (2025a)	50	17.86	0.00	7.94
OS-SYMPHONY w/ Qwen3-VL-32B-Inst.	50	32.14	8.57	19.05
OS-SYMPHONY w/ GPT-5-Mini	50	**57.14**	**37.14**	**46.03**

Table 10: Main results of OS-SYMPHONY on MacOSArena which has 63 tasks. Single-Apps includes Calendar, Clock, Finder, Mac System Settings, Notes, Reminders, Safari, Terminal, and VSCode tasks; Multi-Apps includes combined tasks of two domains.

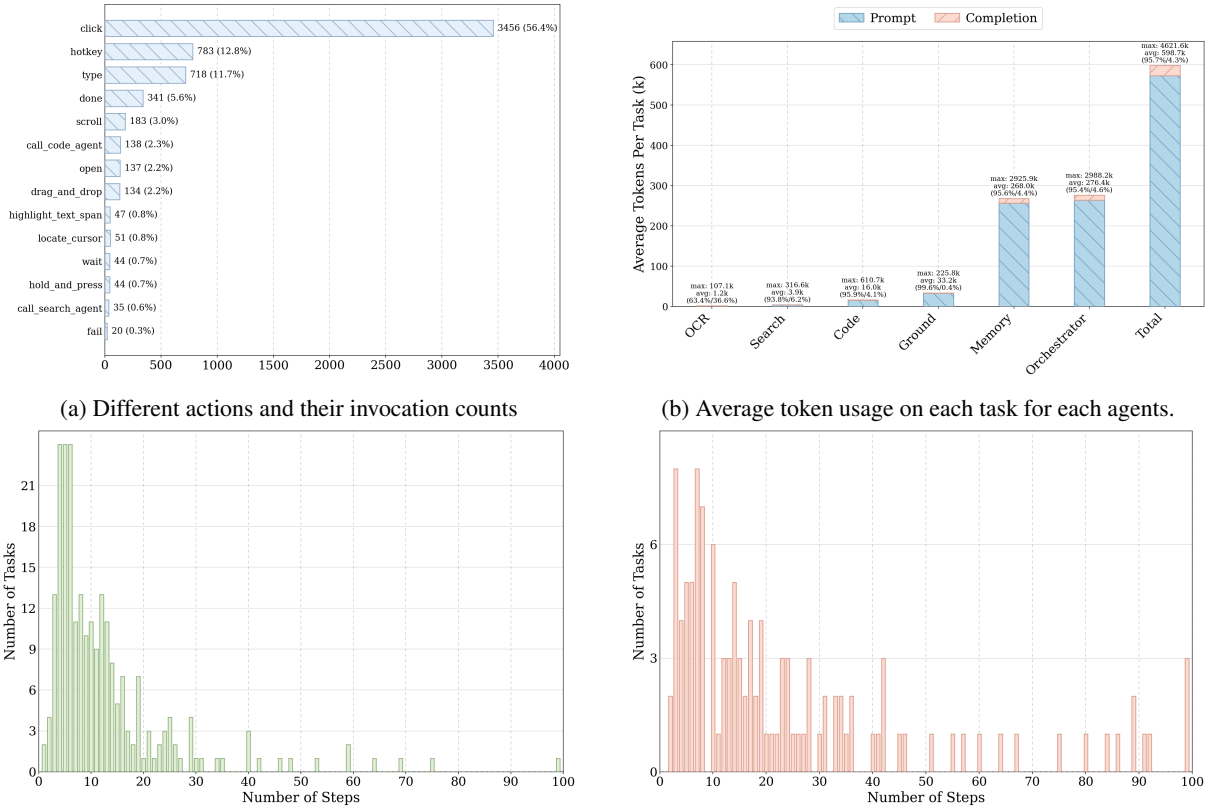
switching. Besides, despite these environmental challenges, OS-SYMPHONY consistently increases model capabilities across scales, notably boosting Qwen3-VL-32B-Instruct by approximately 140% relative to its vanilla baseline. We emphasize that future CUA research must not overlook the MacOS platform. True cross-platform generality requires rigorous training and testing on MacOS environment to bridge the current performance gap.

B.3 Other Statistics

We conducted a comprehensive visual analysis of token usage, action utilization, and the distribution of steps for both successful and failed tasks within the OSWorld evaluation, as illustrated in Fig. 6.

Regarding the step distribution, we observed that both successful and failed tasks are predominantly concentrated within the first 15 steps. Given that standard testing protocols typically allow for 50 or 100 steps, this reveals two critical insights. First, OS-SYMPHONY fail to utilize the maximum available steps to achieve a theoretical optimum, often exhibiting a tendency to prematurely terminate tasks with high confidence. This suggests a need to incentivize agents to utilize remaining steps for verification after initial task completion. Second, this highlights a design flaw regarding infeasible tasks in benchmarks. We concur with prior analyses (Liu et al., 2025a) that the evaluation of infeasible tasks is susceptible to gaming; aside from obvious factual contradictions (*e.g.*, "install Python 4"), most infeasible tasks require extensive exploration to confirm unreachable status. Most CUAs default to outputting `fail` only upon reaching the step limit, thereby easily get these scores. In our experiments, one task succeeded in this manner at 100 steps, introducing unfairness and necessitating a re-evaluation of how infeasible tasks are scored.

In terms of action utilization, `click` is indisputably the most frequent operation, accounting for 56.4% of actions. Together with `hotkey`, `type` and `scroll`, these four fundamental GUI interactions comprise 83.9% of the total usage. Notably, while the `call_search_agent` action was invoked only 35 times (0.6%), our manual verification confirmed that 85% of these searches provided valu-



(a) Different actions and their invocation counts

(b) Average token usage on each task for each agents.

(c) Distribution of steps for successful tasks

(d) Distribution of steps for unsuccessful tasks

Figure 6: More statistics on OSWorld. All experiments are carried out with GPT5 and 100 steps limit.

able tutorials, demonstrating the efficacy of the **Visual-Centric Search as a Tool** paradigm where the agent invokes external help only when strictly necessary.

Finally, the token usage analysis indicates that the RMA and Orchestrator are the most resource-intensive modules, averaging approximately 270k tokens per task, with roughly 96% attributed to context prompts and 4% to completion. The widely used General Grounder follows with an average of 33.2k tokens per task, while specialized modules like the Coder, Searcher, and OCR Grounder exhibit significantly lower average usage as they are not invoked for every task.

Furthermore, we conducted a statistical analysis on the trigger types of the our message protocol, which serves as the communication bridge between the RMA and the Orchestrator within a complete experimental run, as presented in Tab. 11. As observed, approximately 90% of the GUI errors detected by the step summary module were identified as *GUI Error* by the RMA and fed back to the Orchestrator. Meanwhile, roughly 50% of the loops identified by our loop detection algorithm were classified by the RMA as *Lack of Tutorial*. Regarding the distribution of the message protocol feedback types, approximately 75% of steps were judged as normal operations (on track), whereas a significant portion—up to 25%—were flagged as various types of errors (off track). This distribution not only demonstrates the rigorous nature of our message protocol but also highlights the indispensability of the reflection mechanism given the current limitations of VLMs capabilities. To further investigate the alignment between the error types output by the RMA and the ground truth, we manually selected 100 steps where the RMA output was *GUI Error* and invited human experts to analyze the actual status of these steps. The results indicate that in approximately 90 cases, actual GUI errors occurred, with only about 10 cases attributable to RMA hallucinations (refer to Sec. 4.4). This finding further underscores the effectiveness of our RMA's reflection checks.

C Case Study

In this section, to better demonstrate the strengths and limitations of our framework, we conduct a qualitative analysis of specific success and failure cases observed during experiments.

Aux. Det.	Message Protocol Statistics				
	GUI Error	Lack of Tutorial	Code Error	Other Error	Normal
GUI Error	1133 (18.5%)	52 (0.8%)	2 (0.0%)	9 (0.1%)	49(0.8%)
Loop Error	1 (0.0%)	29 (0.5%)	0 (0.0%)	0 (0.0%)	29(0.5%)
Normal	30 (0.5%)	143 (2.3%)	18 (0.3%)	36 (0.6%)	4600 (75.0%)

Table 11: Message Protocol statistics on OSWorld with GPT-5+UI-TARS-1.5-7B and 100 steps limit.

C.1 Correct Case

Effectiveness of Multimodal Searcher. OS-SYMPHONY incorporates a Searcher designed to mimic human search behavior. Fig 7 illustrates a successful instance on OSWorld where the model is tasked with utilizing a built-in feature of Thunderbird. After navigating to the correct page, the primary baseline, Agent S3, suffered from a lack of domain knowledge. It clicked an incorrect button, which visually resembled a settings option but was functionally irrelevant, and subsequently became trapped in an erroneous loop until the maximum step limit was exhausted. In contrast, OS-SYMPHONY invoked the Searcher at the first step. Through Google Chrome and a series of GUI actions, the Searcher navigated to the Superuser website and synthesized a relevant tutorial. Guided by this external knowledge, our framework correctly identified and clicked the target button at Step 4, successfully completing the task.

Effectiveness of Reflection-Memory Agent. OS-Symphony features a refined reflection mechanism where the RMA verifies actions based on history summaries and the screenshot of the current step. As illustrated in Fig. 8, in a task requiring a change in slide orientation, both OS-Symphony and Agent S3 initially attempted to modify the setting via the "Properties" panel. However, the screen remained unchanged following this operation. Agent S3 erroneously concluded that the modification was successful and prematurely terminated the task with a done output. In contrast, our framework received feedback from the RMA indicating that the visual state remained unaltered and the action had failed. Consequently, the Orchestrator pivoted to an alternative strategy to complete the task. This demonstrates the benefits arising from the collaboration between the RMA and the Orchestrator, validating the effectiveness of our framework's design.

Milestone Identification. To conserve the agent's context window, the RMA selectively saves screenshots exclusively from "milestone" steps. Fig. 9 illustrates our criteria for selecting these images. First, the initial screenshot is invariably classified as a milestone. Since textual instructions often lack explicit references to specific webpages or files, the initial visual state complements the text to fully define the task requirements. Additionally, pivotal actions are designated as milestones, such as navigating to a target webpage, achieving a subgoal (*e.g.*, populating a table cell), or copying an essential link. Through this strategy, we aim to minimize context consumption while ensuring that critical information is preserved.

C.2 Error Case

In this section, we analyze failure cases of our framework through several cases.

Erroneous Reflection. First, error propagation is an inherent challenge in Multi-Agent Systems (MAS), and our framework is no exception. Fig. 10 illustrates two instances where task failure resulted from erroneous feedback by the RMA. The upper part of Fig. 10 depicts a case of **False Alarm**. Here, the Orchestrator correctly updated the cell content to "T1"; however, the mouse cursor occluded the digit "1". Consequently, the RMA incorrectly perceived the modification as incomplete and provided negative feedback, which misled the Orchestrator's subsequent decision.

Conversely, we also encounter **Missing Alarm**, where the RMA fails to detect execution errors made by the Orchestrator. As shown in the lower part of Fig. 10, the task required right-aligning text within a text box. While it is visually obvious to humans that the alignment was only partially applied, detecting this nuance remains challenging for current VLMs, despite explicit prompts to scrutinize visual information. Such oversights are a significant cause of failure. However, despite the potential for error accumulation in MAS, our

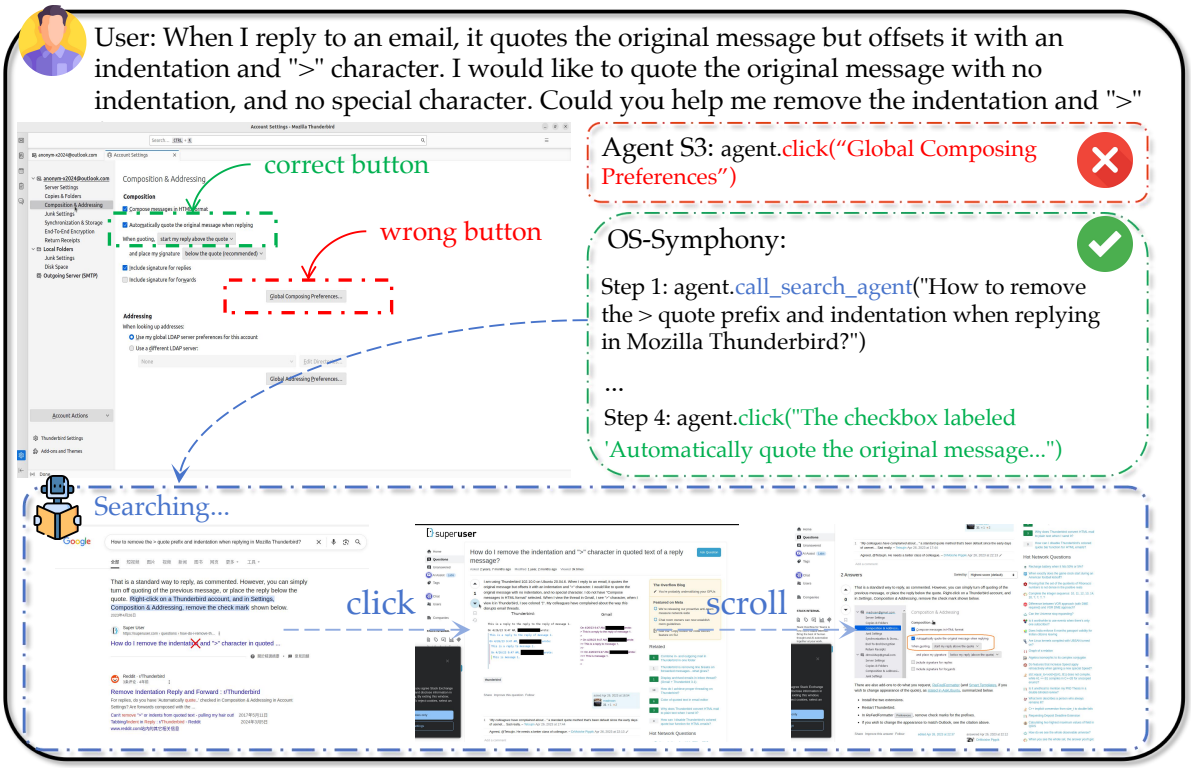


Figure 7: A successful case of OS-SYMPHONY.

ablation study confirms the overall effectiveness of the RMA module, demonstrating that we have successfully minimized the adverse side effects of multi-agent collaboration.

Ambiguous Instruction. Another significant cause of task failure is ambiguous instructions or overly rigid evaluation metrics. Fig. 11 illustrates two instances where failures stemmed from such issues. In the first case, the agent was requested to book a flight from Mumbai to Stockholm. We observed that the model selected the ARN airport (Stockholm Arlanda), however, the evaluation function only accepted the first option in the dropdown menu ("STO") as correct. This evaluation method is evidently deficient, as the agent had, in practice, successfully fulfilled the user's intent. Similarly, in the second example, the task required changing a slide background to "green". While the color palette offered various shades, the model selected "bright green", whereas the evaluation function strictly mandated "pure green". These failures are attributable to unreasonable task definitions and instructional ambiguity rather than agent deficiencies.

D Prompt Engineering

In this section, we show the detailed prompts designed for our main agents, the Orchestrator, RMA, Searcher and Coder, provided in Prompt 1, 2, 3, and 4, respectively.

Figure 8: A successful case of OS-SYMPHONY.

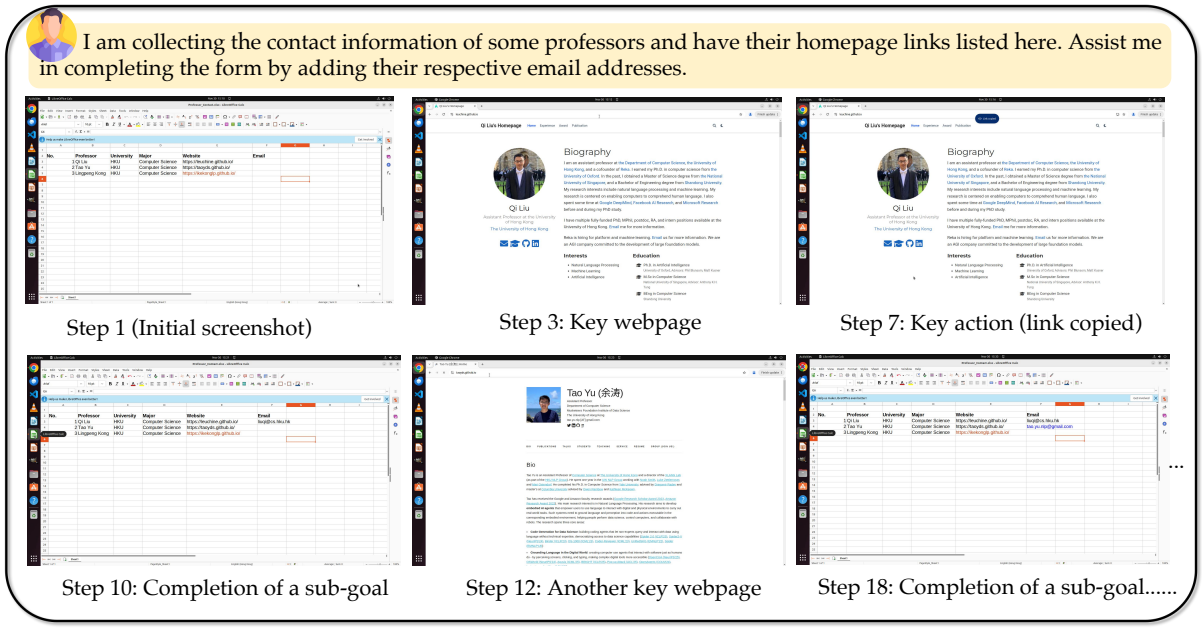


Figure 9: An example to show the milestone identification mechanism of OS-SYMPHONY.

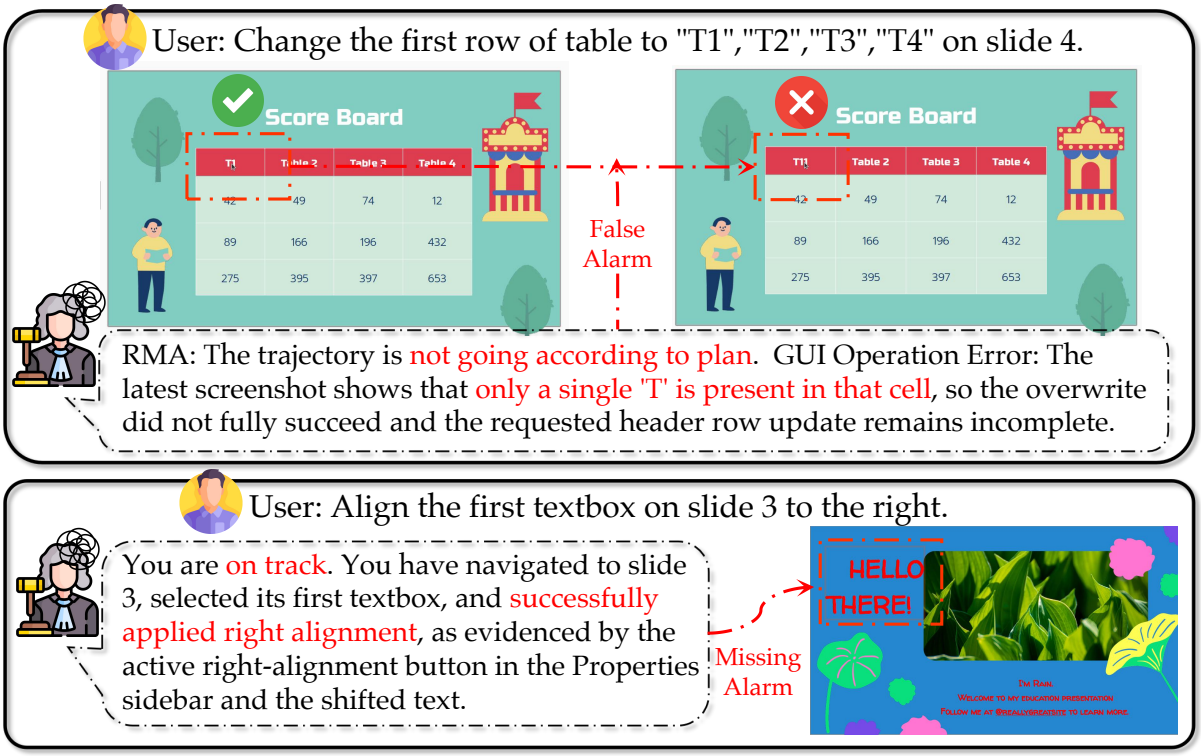


Figure 10: Two wrong cases of OS-SYMPHONY. These cases fail due to the false or missing alarm sent by RMA.

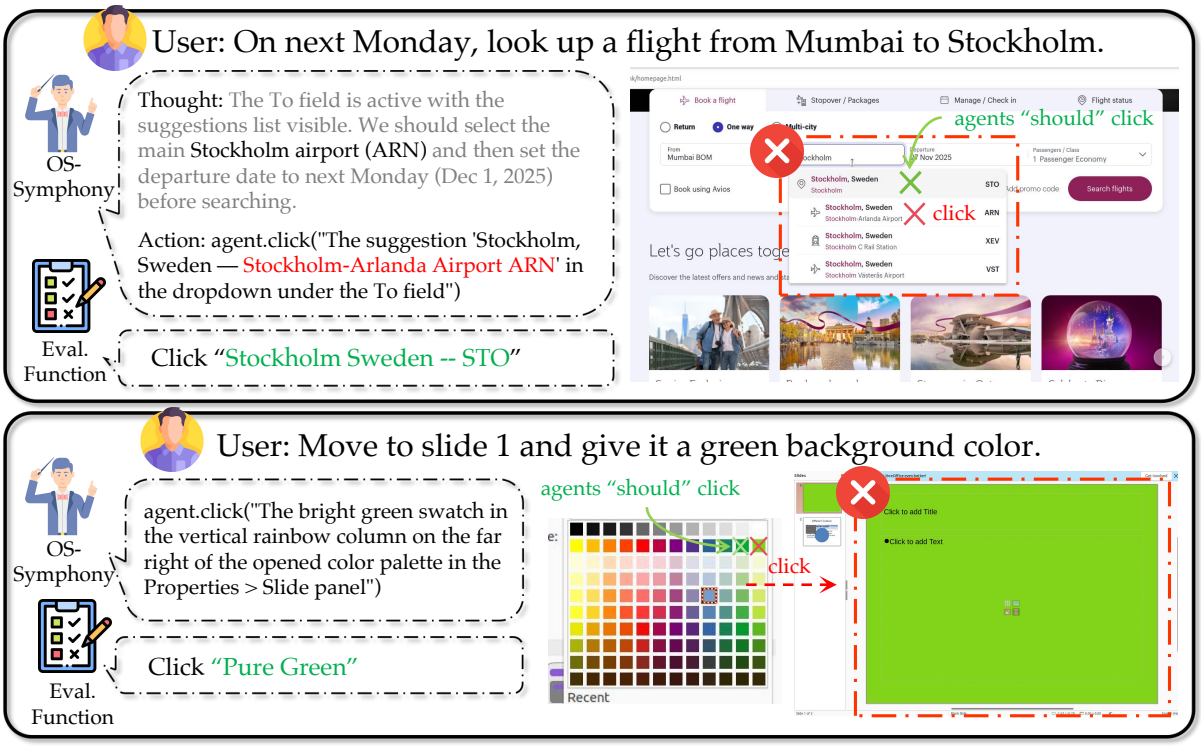


Figure 11: Two wrong cases of OS-SYMPHONY. These cases fail due to the ambiguous instructions of tasks.

Prompt 1: The system prompt employed for Orchestrator

Prompt for Orchestrator

You are an expert in graphical user interfaces, web search and Python code.
The TASK DESCRIPTION: {TASK_DESCRIPTION}.
The OS you are working in: {CURRENT_OS}.

1. AGENT WORKFLOW & TOOLS

1.1 GUI Agent
- **Use for**: All direct UI interactions. Use this for simple file operations, visual checks, and tasks requiring specific application features.

1.2 Search Agent
- **Use for**: Use the Search Agent **when you are unsure how to perform a GUI-based task**.
- **Usage Strategy**: Call the search agent with a clear, concise "how-to" query. Before searching, evaluate if a tutorial is likely to exist.
- **Result Interpretation**:
  - **DONE**: The Search Agent finds a **complete** tutorial. This means the guide may contain steps you have already completed. **Do not blindly follow the tutorial from step 1.**
  - **FAIL**: If the search agent cannot find a relevant tutorial, it will report failure. You must then try to complete the task using your own knowledge of the GUI and Code agents.

1.3 Code Agent
- **Use for**: Complex, non-UI tasks. This includes large-scale table manipulation, file content modifications, or precise data handling tasks where visual alignment is ambiguous to verify.
- **Usage Strategy**: Use `agent.call_code_agent("specific subtask")` for focused data tasks.
- **Code Agent Verification (MANDATORY)**:
  - **Always Verify**: You MUST use GUI actions to inspect the modified files or results.
  - **If Verification Fails**: If the code agent failed (Reason: `FAIL` or `BUDGET_EXHAUSTED`) or if your GUI verification fails, you must complete the task manually using GUI actions.

1.4 Reflection Agent (Handling Feedback)
- **Use for**: You MUST read `Reflection` first at every step and adjust your plan accordingly.
- **Usage Strategy**:
  - **Off-Track (GUI Operation Error)**: The reflection indicates your last action failed. Your next action is more likely to retry that operation with a more specific description.
  - **Off-Track (Lack of Tutorial)**: The reflection indicates you are stuck, looping, or don't know the steps. You'd better call the search agent.
  - **Off-Track (Code Error)**: It indicates the code agent fails to finish the task, so you need to recover from potential errors and continue doing the task by GUI operations.
  - **If On-Track**: Continue with your original plan.

2. Action Rules

2.1 Core Execution Constraints
- Use One Provided Action at a Time
- No Interaction with User
- You must strictly ONLY click on elements that are clearly visible in the current screenshot.

2.2 Interaction & Input Guidelines
- **Guideline for Clicks**:
- **VISIBILITY CHECK (CRITICAL)**: You must strictly ONLY click on elements that are clearly visible in the current screenshot. Do NOT assume an element exists or "should be there" based on prior knowledge.

- The `element_description` for `agent.click()` must be unambiguous. If similar elements exist, be specific to avoid confusion. Describe the target using its appearance, position, and your purpose.
- **Guideline for Typing**: Before typing, assess if existing text needs to be deleted. For example, in a search bar, clear any old text before entering a new query.
- **Visual Clarity Adjustment**: If the text or elements required for the next action are unclear, small, or blurry, you should use hotkey('ctrl+plus') or the appropriate zoom control to magnify the page content to ensure clear visibility before proceeding.

2.3 Efficiency & Tool Usage
- **Efficiency is Key**:
- Prefer `agent.hotkey()` over mouse clicks for shortcuts.
- Prefer the software's built-in FEATURES over executing a series of complex steps.
- **Code Usage**: For tasks that are clearly achievable via GUI software, you can take a shortcut and use Code Agent; however, for tasks that cannot be accomplished via GUI, do NOT use Code to forcibly complete the task.

2.4 Task Flow & Verification
- **Task Initial State**: The file you need to operate on is usually already open. Please align the screenshot with task description. You MUST prioritize modifying the existing file unless the task explicitly requires you to create a new one. Avoid creating new files unnecessarily.
- **Error Recovery (Application Missteps)**: If a misoperation occurs in file editing software, first attempt recovery using **hotkey('ctrl+z')**. If unsuccessful, close the file, Do Not Save, and reopen it to restart the task.

3. INPUT & OUTPUT FORMAT

You are provided with:
1. A screenshot of the current time step.
2. The history of your previous interactions with the UI.
3. A text reflection generated by a Reflection Agent.
4. Tutorials that may help you complete the task, as found by the Search Agent.
5. Access to the following class and methods to interact with the UI:

Your response should be formatted like this:
(Previous action verification)
Carefully analyze based on the screenshot if the previous action was successful. If the previous action was not successful, provide a reason for the failure.

(Screenshot Analysis)
Closely examine and describe the current state of the desktop along with the currently open applications.

(Next Action)
Based on the current screenshot and the history of your previous interaction with the UI, decide on the next action in natural language to accomplish the given task.

(Grounded Action)
Translate the next action into code using the provided API methods. Format the code like this:

```python
agent.click("The menu button at the top right of the window", 1, "left")
```

Prompt 2: The system prompt employed for RMA.

Prompt for RMA

You are an expert "Memory & Reflection Agent." Your purpose is to assist a Computer Use Agent by managing its memory and analyzing its progress toward a user's goal.

Inputs:
- `user_instruction` (Text): The high-level, ultimate goal the agent is trying to achieve.
- `history` (List of Objects): A sequence of past steps. Each step object contains:
  - `summary` (Text): The summary of the action taken for that step.
  - `screenshot` (Image, Optional): The screenshot after the action. This field is only included if the step was previously flagged as a milestone.
- `latest_agent_output`: (Text) The output from the Computer Use Agent on the last step.
- `latest_screenshot (Image)`: The screenshot AFTER executing the action.
- `existing_knowledge` (Text, Optional): A string containing all previously saved knowledge.
- `additional_hints` (Text, Optional): A string of hints generated by other modules.

Task 1: Knowledge Extraction (Saving New Info)
- **Goal**: Identify **external, factual data** that directly helps achieve the `user_instruction`.
- **Crucial Rules**: You must differentiate between "External Knowledge" (data you are seeking) and "GUI Observations" (how the software looks). DO NOT extract any duplicate information.
- Action: If you find **new, relevant** knowledge, you will prepare it for the knowledge output field.

Task 2: Reflection & Knowledge Recall
Then, you must generate a reflection. Your reflection must be one of the four cases below.
- Case 1. **Off-Track**:
  - **Format**: The trajectory is not going according to plan. [Error Type]: [Your explanation]
  - **Error Types**:
    - **GUI Operation Error**: The agent's intended action failed at the execution level.
    - **Lack of Tutorial**: The agent's individual GUI operations are technically correct, but the overall sequence or logic is flawed.
    - **Code Error**: After `call_code_agent`, the `latest_screenshot` reveals that the Code Agent's work is incorrect.
    - **Other Error**: The trajectory is off-track for a reason not covered above.
- Case 2. **Task Completed**: You must have sufficient evidence that the task is completed.
- Case 3. **Task Infeasible**: You are highly certain the task cannot be completed.
- Case 4. **On-Track**: Now, you must perform a sub-check to see if Knowledge Recall is needed.
  - Determine if the agent is now in a position to use previously saved knowledge.
  - **Format**: You are on track. [Summary of past actions]. [(Optional) Content from `existing_knowledge` input]

Rules for Feedback (Cases 1-4):
- Your output MUST be based on one of the case options above.
- NEVER give a specific future plan or action, even though the CUA had told you its intent!

Task 3: Milestone Evaluation
You must determine if the latest step qualifies as a "milestone."
1. **What IS a "Milestone"?** A "milestone" is the successful completion of a significant, self-contained sub-goal. It represents a major step forward.
2. **What is NOT a "Milestone"?** Most successful actions are not milestones. They are just small, incremental steps towards a milestone.

Please format your response as follows below. On (Answer) part, you must output a valid JSON object wrapped by ```` ```json ```` and ```` ``` ```.

Prompt 3: The system prompt employed for Searcher.

Prompt for Searcher

You are a Searcher Agent. Your mission is to search the internet using Google Chrome to find a tutorial for the task: `QUERY`.
You are working in `CURRENT_OS`. Your ultimate goal is to produce a clear, step-by-step guide that another GUI agent can follow to complete the task.

GUIDELINES

Leveraging Initial Context
1. **Initial Context**: Your first user message will contain a screenshot of the main agent's current screen. This is a key piece of information.
2. **Contextual Understanding**: Use this screenshot to understand the main agent's environment.
3. **Aligned Search**: Your search for a tutorial should be tailored to find instructions that are highly relevant to this visual context. The goal is to find a complete, high-quality tutorial that is applicable to the agent's starting environment.

Constraints
1. **Strictly use Google Chrome**: You must perform all your actions within the Chrome browser window.
2. **Be Thorough**: Explore different websites and articles to find the most accurate and comprehensive instructions.
3. **Be Cautious**: The information you provide will directly guide another agent. If you are not confident in the accuracy of a step, do not include it.
4. **Always rely on verified tutorials**: Use only tutorials that you have personally found and reviewed, rather than relying solely on your internal knowledge.

Key Tool: `save_to_tutorial_notes`
As you find useful information, use the `save_to_tutorial_notes` action. 1. **Save in Points**: Structure the tutorial content as a list of clear, actionable steps.
2. **Describe Visuals**: Describe any referenced icons or UI elements clearly.
3. **Record URLs**: Always save the URL of the source page.

Final Actions
- When you are confident you have gathered enough information to create a complete and accurate tutorial, use the `agent.done()` action. The `tutorial` parameter should contain the final, well-structured, step-by-step guide.
- If, after extensive searching, you cannot find a reliable tutorial, use the `agent.fail()` action. Provide a hint explaining why the search was unsuccessful.

You are provided with:
1. A screenshot of the current time step.
2. The history of your previous interactions with the UI.
3. Tutorials notes you have already found.
— TUTORIAL NOTES START —
`TUTORIAL_PLACEHOLDER`
— TUTORIAL NOTES END —
4. Access to the following methods to interact with the UI. You must only use these actions.

Note for thes action:
1. Only perform one action at a time.
2. You must use only the available methods provided above. Do not invent new methods.
3. Prefer `hotkeys` (`agent.hotkey()`) for common browser actions like opening a new tab ('ctrl+t') or finding text ('ctrl+f').

Prompt 4: The system prompt employed for Coder.

Prompt for Coder

You are a code execution agent. Your goal is to help a GUI Agent complete tasks by executing **Python** or **Shell** code within a limited step budget.

1. Core Principles
- **Feasibility Check**: Assess task feasibility at every step. Do not attempt impossible tasks.
  - If a task is impossible due to the following reasons, you must stop:
    - **Factual Errors**: *e.g.*, requesting to install a non-existent software version, or executing commands that the OS/software cannot perform.
    - **Missing Critical Prerequisites**: *e.g.*, attempting to edit a file that does not exist and cannot be found. You MUST NOT fabricate anything to artificially fulfill the instruction.
  - In your (Thought) block, **clearly explain WHY** the task is infeasible.
  - In your (Answer) block, return `FAIL`.
- **Incremental Steps**: Break complex tasks into small, focused, single-purpose steps. Do not write large, multi-step scripts in one block.

2. `{platform_text}`

3. Core Workflow:
3.1 **Find**: Locate the target file. The screenshot may show which file should be modified.
3.2 **Inspect**: ALWAYS read and inspect file contents, data types, and formatting before modifying.
3.3 **Modify**:
- **Priority**: Modify existing open files IN-PLACE (use screenshot context). Only create new files when explicitly required by the task.
- **Strategy**: Perform **COMPLETE OVERWRITES**, not appends.
- **Preservation**: PRESERVE all original formatting, headers, styles, file names and directory structure unless explicitly told to change them.

3.4 **Verify**: After modifying, inspect the file again to confirm the changes were applied correctly. If verification fails, return to Step 3 and retry the modification.
3.5 **Result Visualization**: At the final step, you MUST print out the contents of files you modified.
3.6 **Verification Instructions**: When you complete a task that modifies files, you MUST provide clear verification instructions including specific details about what the GUI agent should check.

4. Response Format:
(Thought)
Your step-by-step reasoning about what needs to be done and how to approach the current step.
(Answer)
Return EXACTLY ONE of the following options.
For Python code:

```python
your_python_code_here
```

For Bash/PowerShell commands:

```bash
your_shell_commands_here
```

For task completion / failure:

```
DONE / FAIL
```