Plan-Action-Reflection: A Three-Role Agentic Framework For Computer Use Agent Task

Xin Su[1] Man Luo[1] David Cobbley[1] Shachar Rosenman[1] Vasudev Lal[1] Phillip Howard[2]
[1]Intel [2]Thoughtworks
{xin.su, man.luo, david.j.cobbley, shachar.rosenman, vasudev.lal}@intel.com
phillip.howard@thoughtworks.com

Abstract

Recent advancements in multimodal foundation models have enabled agents to perform complex computer use tasks by interpreting and interacting with graphical user interfaces. However, these agents often struggle with task decomposition and error reflection. To address these limitations, we propose a three-role agentic framework that enhances performance through structured task planning and reflection. Our framework consists of: (1) a planning agent that decomposes high-level user goals into actionable sub-tasks, (2) an action agent that executes the sub-tasks via grounded multimodal actions, and (3) a reflection agent that monitors execution outcomes and provides feedback to update the plan or correct the action. Our experiments on benchmark computer use tasks demonstrate that the proposed framework significantly boosts task completion rates.

1. Introduction

The Computer Use Agent (CUA) task aims to develop robust agents capable of autonomously operating graphical user interfaces (GUIs) to complete real-world tasks on a computer, such as booking a flight, managing files, or adjusting system settings [17, 20, 21]. Such agents have the potential to significantly enhance human productivity, accessibility, and automation across a wide range of applications. However, building effective CUAs remains a challenging goal due to the need for complex task decomposition, multimodal reasoning, visual grounding, and error recovery in dynamic environments.

Recent advances in vision language models (VLMs) [2, 11] and agent-based frameworks have led to encouraging progress on this front [7, 10, 25, 26]. An increasing number of benchmarks [12, 22, 23, 28] provide structured environments to evaluate agents' capabilities in web navigation and GUI control. Meanwhile, models like GPT-4V [15] and Claude [1] have demonstrated initial success in integrating reasoning and action in multimodal settings. Despite these developments, existing systems still fall far short of human performance on benchmarks [22] due to the lack of explicit planning and robust error correction. Another major challenge is acquiring training data, as such data are difficult to collect in the wild.

Without relying on large-scale training data collection, we propose a three-role agentic framework that improves the performance and reliability of CUAs. Our framework introduces three specialized agents: a planning agent that decomposes high-level user instructions into coherent sub-tasks, an action agent that executes these sub-tasks in the environment, and a reflection agent that monitors outcomes and provides corrective feedback. This modular design explicitly incorporates planning and self-monitoring into the agent loop. Moreover, this plug-and-play framework allows each agent to adopt different backbone models.

To evaluate the effectiveness of our framework, we use GPT-4.1 [1] as the backbone model and conduct experiments on the OSWorld benchmark [22]. Our results show that the proposed framework significantly improves task completion rates across domains. Through further analysis, we identify the primary bottleneck to be the action agent's visual grounding process, which often fails to accurately localize interactive elements, leading to many of the observed task failures. This finding suggests that future efforts in CUA should prioritize improving the reliability and precision of action execution, particularly in the grounding component.

2. Related Work

We have seen rapid progress in autonomous agents conducting computer use tasks in recent years, driven by advances in large language models (LLMs) such as GPT-4 [14] and Claude [1], as well as multimodal vision-language models like GPT-4V [15], Qwen-VL [2] and LLaVA [11]. We dis-

[1]https://openai.com/index/gpt-4-1/

cuss existing work into two major directions: the design of benchmarks and the development of methods that enhance agent capabilities. In terms of benchmarks, the field has evolved from simple, single-page tasks in synthetic environments (e.g., MiniWoB++[12]) to long-horizon, realistic workflows involving complex tools and dynamic UIs (e.g., WebArena[28] and OS-World [22]). These benchmarks simulate web browsing and desktop interactions, and provide challenging settings to evaluate planning, memory, and generalization abilities. Methodologically, early approaches relied on structured textual inputs such as HTML and DOM trees to enable LLM-based planning [8, 24, 27]. More recent works adopt vision-based agents that operate purely on pixel-level observations, mimicking human interaction with GUIs. For example, Pix2Act [9] demonstrated that a Transformer-based vision agent, pre-trained on screen-text prediction tasks, could outperform crowdworkers on MiniWoB++ without any DOM access. Hybrid systems have also emerged, combining textual and visual signals to leverage the strengths of both modalities. These have been particularly effective in real-world deployments such as OpenAI's GPT-4V-based Computer-Using Agent [16] and Anthropic's Claude with computer-use capabilities [1]. These agents integrate multimodal reasoning to perform tasks across diverse environments. Beyond single-agent performance, various techniques have been introduced to improve accuracy performance. These include reinforcement learning [19], visual grounding [4], self-reflection [8, 18], and in-context learning with few-shot demonstrations [27]. In parallel, there is a growing interest in agentic frameworks that structure decision-making into modular components. AgentStore [6] introduces a meta-agent that coordinates specialized sub-agents through dynamic retrieval. Agent-S [5] which incorporates a planner, retriever, and specialized modules to support decomposition, memory, and tool use. Navi [3] integrates vision and tool-use capabilities in a real Windows OS environment, and demonstrates competitive results on both Windows tasks and Mind2Web. Despite these advances, current agents still lag behind human performance and lack of robustness and task generalization. Handling unexpected interface changes (e.g. random pop-up message windows and different image resolution) and aligning plans with user intent remain open challenges for the future development of computer-use agents.

3. Methodology

Overview In this work, we propose a vision-only solution that enables a general-purpose, task-agnostic VLM to perform human-like computer interactions using visual input—such as screenshots—without any task-specific fine-tuning. To achieve this, we adapt a common agentic framework design pattern based on the plan–action–reflection loop, and incorporate tool use to bridge the general capabilities of a VLM with the specific demands of computer operation tasks. We structure the VLM's role through prompting, allowing it to instantiate different types of agents within the framework. Given a computer operating environment current GUI and a user-defined task, our method unfolds as follows:

1. Planning Agent: Based on the user's instruction and the initial GUI, the planning agent generates a plan consisting of a sequence of sub-tasks necessary to accomplish the overall goal.
2. Action Agent: The action agent extracts the next sub-task from the plan and treats it as the current objective. It then generates an appropriate action based on the current GUI and executes it in the environment. To improve the precision of identifying interactive GUI elements, we incorporate a grounding tool within the action agent.
3. Reflection Agent: The reflection agent monitors execution outcomes by comparing the interface state before and after each action. It determines whether the current sub-task has been successfully completed and guides the control flow accordingly—either progressing to the next step, retrying the current sub-task, or returning to the planner for re-planning.

A central aim of our approach is to offer a lightweight and accessible framework that enables standard VLMs—without additional training—to function effectively as agents for computer use. An overview of our method is illustrated in Figure 1.

3.1. Planning Agent

In our framework, the planning agent plays a central role in both task decomposition and state management, as well as decision control.

Task Decomposition and State Management Given a high-level computer use task and the initial interface state, the planning agent decomposes the user's goal into a sequence of concrete, manageable sub-tasks. It also tracks the completion status of each sub-task to monitor overall task progress and maintain clarity on what remains to be done. This decomposition significantly reduces the complexity of each step, ensuring that sub-tasks are simple, well-bounded, and easier to evaluate for success or failure by other agents. For example, if an agent is directly asked to "change Chrome's default search engine to Bing" and fails, it is difficult to diagnose what went wrong. In contrast, if the agent is assigned the sub-task of "click the settings button in Chrome," the failure mode becomes easier to detect—e.g., whether it failed to locate the correct UI element or used the wrong action (right-click instead of left-click). This decomposition also supports global task coordination. By

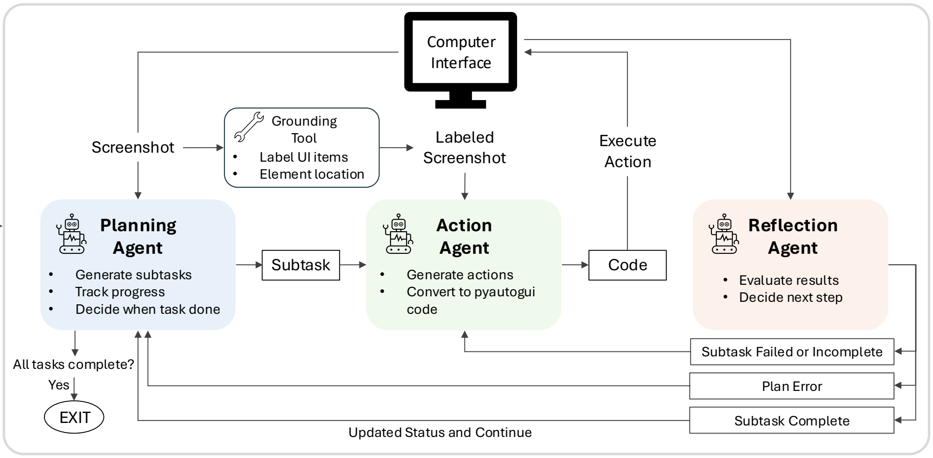

Figure 1. Overview of our approach.

maintaining awareness of the entire task structure, the planning agent ensures that each sub-task aligns with the user's original intent and prevents the system from drifting away from the overall goal. When the reflection agent detects errors in execution or planning, the planner can dynamically revise the sub-task history and replan accordingly. This enables the system to recover from failures through flexible replanning and avoid repeated attempts of previously failed steps.

Decision Control Beyond generating the initial high-level plan, the planning agent controls the execution flow by selecting the next sub-task based on the current GUI or screenshot and determining when the overall task is complete. It defines termination conditions for the framework, preventing infinite execution loops.

3.2. Action Agent and Tool Using

As illustrated in Figure 1, each input screenshot is first processed by a grounding tool before being passed to the action agent. We employ the lightweight *OmniParser* [13] as the grounding tool. It identifies interactive GUI elements by generating short captions, locating their bounding boxes, and overlays the boxs on the screenshot.

This design choice is motivated by our empirical observations: most VLMs , without task-specific fine-tuning, struggle to reliably localize and refer to user interface components in visually complex environments. The grounding tool complements the VLM by providing precise and consistent identification of interactive elements.

The action agent is responsible for executing individual sub-tasks, such as "clicking the settings button in Chrome." Given the sub-task description, the grounded screenshot, and bounding box metadata, the agent generates a structured representation of the intended action. This can be viewed as a form of multimodal semantic parsing—translating visual and textual inputs into an executable command.

For action representation, we adopt the form from OmniParser's code repository [2]. The model is prompted to produce a JSON object specifying the `action_type`, the target `box_id`, and an optional `value`. This output is then deterministically translated into executable Python code using libraries such as `pyautogui`. For example:

```
{"action": "left_click",
"box_id": 42,
"value": null}
```

The corresponding Python code is:

```
import pyautogui
pyautogui.moveTo(box_42_center_x,
                 box_42_center_y)
pyautogui.click()
```

This structured representation enables consistent translation into executable commands.

3.3. Reflection Agent

After the action code generated by the Action Agent is executed in the computer environment, the Reflection Agent evaluates the outcome. For each specific action, it compares the screenshots taken before and after execution to detect visual changes—such as whether a button was clicked or text was correctly entered—and determines whether the sub-task has been successfully completed.

The Reflection Agent classifies outcomes into three categories: **Plan Error**, **Subtask Failed or Incomplete**, and **Subtask Success**. Based on the differences between the

[2]https://github.com/microsoft/OmniParser

pre- and post-execution interface states, it autonomously identifies which category applies and generates a corresponding reasoning trace as feedback to guide the next control decision:

- If a **Plan Error** is detected, or if the Action Agent exceeds a predefined failure threshold for the current sub-task, control is returned to the Planning Agent to revise the task plan.
- If the sub-task is deemed **Failed or Incomplete**, the reflection result is passed back to the Action Agent, initiating another iteration of the action–reflection loop to continue attempting the sub-task.
- If the sub-task is identified as **Successfully Completed**, the feedback is returned to the Planning Agent to update the progress state.

The design of the Reflection Agent leverages the fine granularity and evaluability of sub-tasks to pinpoint specific failure types—such as incorrect UI element localization, wrong action execution, or flaws in the task plan—thereby simplifying error diagnosis and correction. Furthermore, feedback from the Reflection Agent enables both the Planning and Action Agents to adaptively adjust their plans or strategies based on observed execution outcomes, thereby avoiding repeated execution of the same failed actions or plans.

4. Experiments

Table 1. Success rates of our method vs. PromptAgent across 10 application domains. LO stands for LibreOffice. Success rates are calculated independently for each domain; the overall success rate is calculated from all examples combined.

Domain	**Ours**	**PromptAgent**	**Diff.**
Chrome (43)	16.1	2.3	+13.8
GIMP (19)	15.8	0.0	+15.8
LO Calc (44)	0.0	0.0	0.0
LO Impress (39)	2.6	0.0	+2.6
LO Writer (21)	0.0	0.0	0.0
Multi Apps (76)	0.0	5.3	-5.3
OS (23)	17.4	13.0	+4.4
Thunderbird (13)	23.1	0.0	+23.1
VLC (12)	8.3	8.3	0.0
VS Code (18)	16.7	0.0	+16.7
Overall (308)	**7.1**	**2.9**	**+4.2**

4.1. Dataset

We evaluate our proposed method on the widely used OSWorld benchmark, using a subset of 308 task instances. Our experiments are conducted in the Ubuntu operating environment, where interactions with the operating system are executed via pyautogui code. During evaluation, the system is restricted to using only OSWorld-provided screenshots as visual input. The selected OSWorld tasks cover 10 distinct domains, including but not limited to browser operations (e.g., Chrome), code editing (e.g., VSCode), and multi-application workflows. For evaluation, we follow the official OSWorld protocol and compute the final task success rate as the metric.

4.2. Implementation Details

We use GPT-4.1 (version 2025-04-14), accessed via Azure, as our base VLM. As a baseline, we adopt the **Prompt Agent** provided in the OSWorld codebase [3], combined with the base VLM. We compare this baseline setup with our full method, which augments the same base VLM with our proposed agentic framework.

4.3. Results

Our experimental results are summarized in Table 1. Compared to the Prompt Agent baseline—built on the same underlying VLM—our agentic framework achieves a substantial improvement in computer use tasks, more than doubling the overall success rate. Our method outperforms the baseline in 7 out of 10 application domains, with particularly strong performance in Chrome, GIMP, Thunderbird, VS Code, and OS tasks.

For the LibreOffice suite, both our method and the Prompt Agent fail to complete any task successfully. Error analysis indicates that the primary cause lies in the grounding process: the LibreOffice interface contains many small and low-visibility UI elements, which make it difficult for the grounding tool to correctly identify and label relevant components.

Our method also underperforms the baseline in the multi-application domain. We attribute this to similar grounding-related issues—specifically, the increased visual complexity caused by overlapping windows in multi-app workflows makes it challenging to generate accurate bounding boxes for interactive elements, thereby degrading overall task execution.

5. Conclusion

In this paper, we present an agentic framework that enables a VLM to perform computer use tasks without any task-specific fine-tuning. Our experiments demonstrate that the proposed method significantly improves performance across multiple application domains. In future work, we plan to integrate trained computer use agents into the framework to further enhance the grounding accuracy and action execution reliability, thereby improving overall system performance.

[3]https://github.com/xlang-ai/OSWorld/blob/main/mm_agents/agent.py

References

[1] Anthropic. Introducing the claude 3 model family. https://www.anthropic.com/news/claude-3-family, 2024. Accessed: 2025-05-03. 1, 2

[2] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023. 1

[3] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024. 2

[4] Shir Gur, Haonan Wang, Roozbeh Mottaghi, et al. Learning to ground commands to ui actions in interactive environments. *arXiv preprint arXiv:2304.03870*, 2023. 2

[5] Jinxin Hao, Tianyi Wu, et al. Agents: Self-reliant agents via self-reflection. *arXiv preprint arXiv:2402.01676*, 2024. 2

[6] Chengyou Jia, Qiushi Sun, Yifan Zhang, Yujie Wang, Xiang Deng, and Yu Su. Agentstore: Scalable integration of heterogeneous agents as specialized generalist computer assistant. *arXiv preprint arXiv:2410.18603*, 2024. 2

[7] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36:39648–39677, 2023. 1

[8] Seungwon Kim, Xi Victoria Lin, and Luke Zettlemoyer. Recursively criticize and improve: A simple yet effective method for llm agents. *arXiv preprint arXiv:2312.02890*, 2023. 2

[9] Joonhyung Lee, Xin Wang, Yujia Li, et al. Pix2act: An image-to-action visual language model for ui grounding. *arXiv preprint arXiv:2305.10989*, 2023. 2

[10] Tao Li, Gang Li, Zhiwei Deng, Bryan Wang, and Yang Li. A zero-shot language agent for computer control with structured reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11261–11274, 2023. 1

[11] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023. 1

[12] Xing Liu, Yujia Wang, Ximing Liu, et al. Miniwob++: A benchmark for ui learning with web interfaces. *arXiv preprint arXiv:1802.08802*, 2018. 1, 2

[13] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent, 2024. 3

[14] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1

[15] OpenAI. Gpt-4v(ision) system card. https://openai.com/index/gpt-4v-system-card/, 2023. Accessed: 2025-05-03. 1

[16] OpenAI. Chatgpt can now use a computer. https://openai.com/index/computer-using-agent/, 2024. Accessed: 2025-05-03. 2

[17] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024. 1

[18] Noah Shinn, Eric Wu, and Tatsunori Hashimoto. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. 2

[19] Shu Sun, Noah Shinn, et al. Adaptive planning for language agents. *arXiv preprint arXiv:2305.16096*, 2023. 2

[20] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6): 186345, 2024. 1

[21] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025. 1

[22] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024. 1, 2

[23] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024. 1

[24] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 2

[25] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. 1

[26] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, pages 61349–61385. PMLR, 2024. 1

[27] Xin Zheng, Yujia Bai, Amogh Jain, et al. Can large language models use tools reliably? *arXiv preprint arXiv:2310.12055*, 2023. 2

[28] Shuyan Zhou, Frank F Xu, Hao Zhu, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. 1, 2