CoAct-1: Computer-using Agents with Coding as Actions

Linxin Song [1], **Yutong Dai** [2], **Viraj Prabhu** [2], **Jieyu Zhang** [3], **Taiwei Shi** [1], **Li Li** [1], **Junnan Li** [2], **Silvio Savarese** [2], **Zeyuan Chen** [2], **Jieyu Zhao** [1], **Ran Xu** [2], **and Caiming Xiong** [2]

[1]University of Southern California, [2]Salesforce Research, [3]University of Washington

Autonomous agents that operate computers via Graphical User Interfaces (GUIs) often struggle with efficiency and reliability on complex, long-horizon tasks. While augmenting these agents with planners can improve task decomposition, they remain constrained by the inherent limitations of performing all actions through GUI manipulation, leading to brittleness and inefficiency. In this work, we introduce a more robust and flexible paradigm: enabling agents to use coding as an enhanced action. We present CoAct-1, a novel multi-agent system that synergistically combines GUI-based control with direct programmatic execution. CoAct-1 features an Orchestrator that dynamically delegates subtasks to either a conventional GUI Operator or a specialized Programmer agent, which can write and execute Python or Bash scripts. This hybrid approach allows the agent to bypass inefficient GUI action sequences for tasks like file management and data processing, while still leveraging visual interaction when necessary. We evaluate our system on the challenging OSWorld benchmark, where CoAct-1 achieves a new state-of-the-art success rate of 60.76%, significantly outperforming prior methods. Furthermore, our approach dramatically improves efficiency, reducing the average number of steps required to complete a task to just 10.15, compared to 15 for leading GUI agents. Our results demonstrate that integrating coding as a core action provides a more powerful, efficient, and scalable path toward generalized computer automation.

Website: https://linxins.net/coact/

1. Introduction

Recent advancements in computer-using agents have primarily focused on operating through Graphical User Interfaces (GUIs). While these GUI agents, powered by vision-language (action) models (Li et al., 2023, Deepmind, 2025a,b, OpenAI, 2024, 2025a, Qin et al., 2025, Xie et al., 2025a, Yang et al., 2025), have demonstrated the ability to perform a variety of tasks, they often struggle with long-horizon planning and interactions in environments with dense GUI elements. For example, workflows common in office productivity suites or involving direct OS-level operations—such as locating a specific table within a multi-sheet spreadsheet, filtering it based on complex criteria, copying the results, and then saving it as a new CSV file—often involve a long and intricate sequence of precise GUI manipulations. Similarly, tasks like finding all image files in a nested directory structure, resizing them to specific dimensions, and then compressing the entire directory into a single archive are brittle and inefficient when solved via GUI actions such as clicking and dragging. In these scenarios, existing agents frequently struggle with visual grounding ambiguity (e.g., distinguishing between visually similar icons or menu items) and the accumulated probability of making any single error over the long horizon. A single mis-click or misunderstood UI element can derail the entire task.

To mitigate these challenges, a prominent line of research has focused on augmenting GUI agents with dedicated high-level planners. Approaches such as GTA-1 (Yang et al., 2025) and other modular systems (Yang et al., 2024, Xu et al., 2024, Agashe et al., 2024, 2025) employ powerful language models like OpenAI o3 (OpenAI, 2025b) to decompose a user's high-level goal into a sequence of more manageable subtasks.

Corresponding author(s): Linxin Song (linxinso@usc.edu), Jieyu Zhao(jieyuz@usc.edu), Ran Xu(ran.xu@salesforce.com), Caiming Xiong(cxiong@salesforce.com)

This hierarchical decomposition can improve performance on complex, multi-step problems by providing a structured plan. However, this paradigm does not fundamentally address the inefficiency and brittleness inherent in relying exclusively on GUI-based actions for execution. The agent must still navigate menus, click buttons, and type into fields, even for operations that could be accomplished more directly and reliably through programmatic means. This leaves the system susceptible to planning uncertainty, visual perception errors, and the integration challenges between high-level planning and low-level action generation.

In this work, we argue for and instantiate a more flexible and powerful action space. We propose a hybrid approach that combines the intuitive, human-like strengths of GUI manipulation with the precision, reliability, and efficiency of direct system interaction through code. We introduce **CoAct-1** (Computer-using Agent with **Co**ding as **Act**ions), a novel multi-agent system that is architected around three specialized agents: Orchestrator, Programmer, and GUI Operator. A high-level Orchestrator serves as the central planner, decomposing the user's goal and assessing the nature of each subtask. Based on this analysis, it assigns the task to one of two distinct execution agents: a Programmer agent, which writes and executes Python or Bash scripts for backend operations like file management, data processing, or environment configuration; or a GUI Operator, a VLM-based agent that performs frontend actions like clicking buttons and navigating visual interfaces. This dynamic delegation allows CoAct-1 to strategically bypass inefficient GUI sequences in favor of robust, single-shot code execution where appropriate, while still leveraging visual interaction for tasks where it is indispensable.

Our experimental analysis provides strong evidence for the advantages of this hybrid design. On the comprehensive OSWorld benchmark, CoAct-1 establishes a new state-of-the-art, achieving an overall success rate of 60.76%. This marks a significant improvement over leading baselines like GTA-1 (53.10%). The performance gains are most pronounced in categories where programmatic control is highly advantageous. For instance, in Calc (70.21%), multi-application (47.88%), and VS Code (78.26%) tasks, the Programmer agent's ability to execute precise scripts leads to substantial gains over the strongest GUI-only methods. Beyond improving success rates, our dual-modality approach dramatically enhances operational efficiency. By replacing long, error-prone click sequences with concise code, CoAct-1 solves tasks in an average of just 10.15 steps, a stark contrast to the 15 steps required by agents like GTA-1. This efficiency underscores the potential of our approach to pave a more robust and scalable path toward generalized computer automation.

2. Related Work

Screen parsing and visual grounding A first line of work focuses on perceiving and grounding GUI elements directly from pixels, without relying on DOM or accessibility hooks. *OmniParser* learns screen-parsing primitives for pure vision–based understanding (Lu et al., 2024). On the grounding side, *SeeClick* (instruction-to-target grounding), *Aria-UI* (instruction grounding over GUIs), and *UGround* (universal GUI grounding) map language to actionable screen locations (Cheng et al., 2024, Yang et al., 2024, Gou et al., 2024). *OS-Atlas* trains a *foundation action model* to generalize across diverse interfaces (Wu et al., 2024). Dedicated grounding evaluations such as *ScreenSpot-Pro* further benchmark grounding under professional, high-resolution settings (Li et al., 2025).

Modular planner–grounder agents A second strand explicitly separates *what to do* from *where/how to act on screen*: a language planner proposes subgoals while a visual model grounds each step. Representative systems include *SeeClick* and *OS-Atlas* (Cheng et al., 2024, Wu et al., 2024). *GTA-1* strengthens this two-stage paradigm via *test-time scaling*: sampling multiple candidate actions and using an MLLM judge to select among them, improving robustness on high-resolution, cluttered UIs (Yang et al., 2025). Other related open frameworks such as *Agent-S / Agent-S2* and *AutoGen* provide reusable infrastructures for multi-agent orchestration and tool calling (Agashe et al., 2024, 2025, Song et al., 2025, Zhang et al., 2024, Wu et al., 2023, Zhang et al., 2023, 2025b).

Native end-to-end GUI agents A third thread trains *native* agents that unify perception, reasoning, and action in a single model. *UI-TARS* exemplifies this approach with a unified action space for mouse/keyboard operations across apps, eschewing hand-crafted controllers (Qin et al., 2025). *AGUVIS* pushes toward unified, pure-vision GUI agents that generalize across interfaces (Xu et al., 2024).

Hybrid agentic frameworks Beyond GUI-only interaction, several agentic systems compose tools and APIs on the fly to extend capabilities at run time. Examples include *UFO-2* (Zhang et al., 2025a), *PyVision* (Zhao et al., 2025), and *ALITA* (Qiu et al., 2025)—which, while not restricted to GUI/CUA, share the principle of dynamically constructing and invoking tools.

3. Computer-using Agent with Coding as Actions

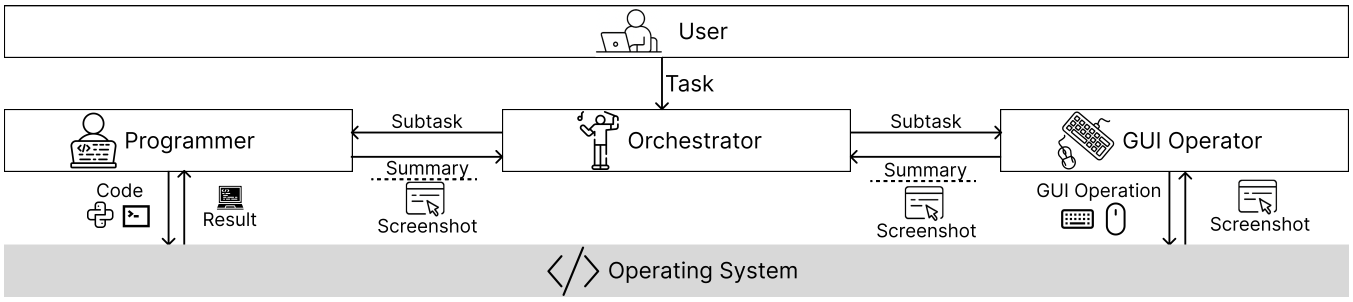


Figure 1: Multi-agent system design for our CoAct-1. This multi-agent system includes a programmer who can interact with the operating system through multi-round coding.

In this work, we introduce a new system-interactive action: coding, to replace redundant and brittle GUI actions. Specifically, we design a multi-agent system called CoAct-1 based on the GUI agent with a planner approach, introducing a new agent called programmer who can interact with the operating system by coding. An orchestrator is designed to determine whether to assign the programmer or the GUI operator to complete a subtask. The overall framework is illustrated in Figure 1.

3.1. Multi-agent System Design for Computer Use

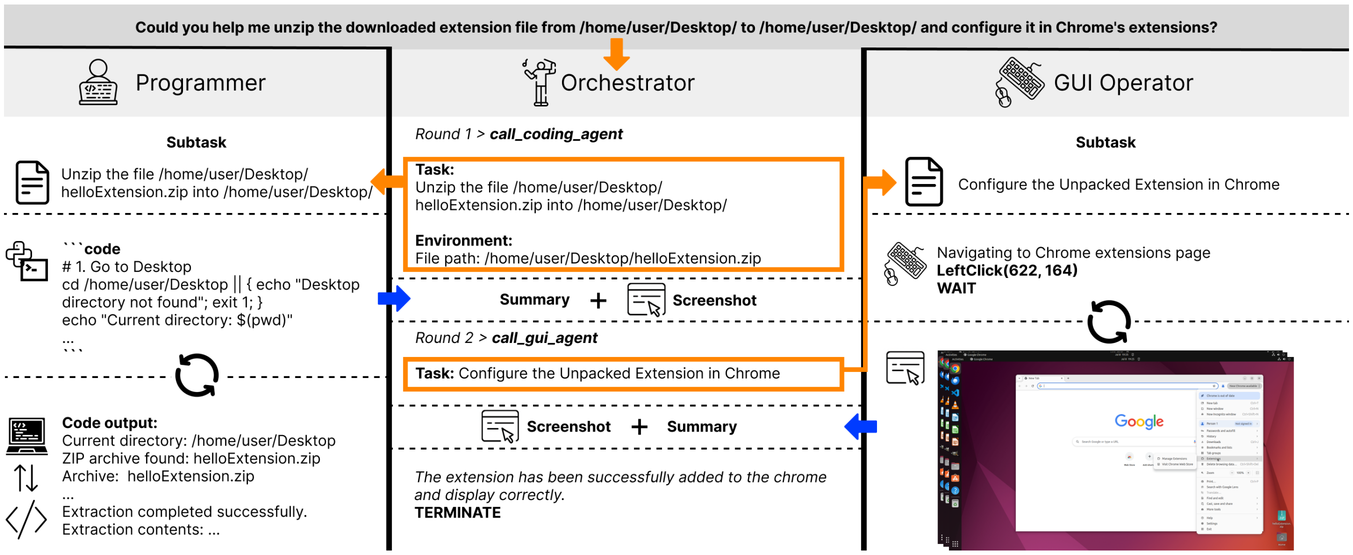
Our multi-agent system comprises three agents: the Orchestrator, Programmer, and GUI Operator, each of which will establish a conversation.

Orchestrator. The orchestrator is responsible for decomposing user tasks and dynamic planning. The orchestrator's step-wise action is based on all previous observations. Orchestrator cannot interact with the OS directly. Therefore, in each step, the orchestrator is required to assign a subtask to either a programmer or a GUI operator. After the programmer and the GUI operator complete their subtask, the operator will receive a summarization of their task-solving process and a screenshot that reflects the current system state. If the orchestrator determines that the task is complete, it will return a termination signal to end the conversation.

Programmer. Once the orchestrator assigns a subtask to the programmer, we will establish a new conversation between the coding agent and code interpreter, enabling the coding agent to reflect the generated code according to the environment feedback in multiple rounds. The coding agent is required to solve the assigned subtask by programming with Python or Bash scripts, which is the two most frequently used languages when operating the OS. When the code interpreter successfully retrieves the code from the coding agent's reply, it sends the code to the operating system (which can be a remote virtual machine). Then it returns the execution result to the conversation along with a screenshot. Notably, we use a language model to serve as a coding agent with limited ability to read and understand sequential images. Therefore, we require the orchestrator to provide sufficient environment information to the programmer, such as the opened windows, browser tabs, or file paths inferred from the user's instructions.

GUI operator. The GUI agent is a vision-language action model, which can take a sequence of images along with the text instruction as input and generate a (sequence of) GUI action(s). When the GUI operator receives a subtask from the orchestrator, it will establish a conversation between a GUI agent and a GUI action interpreter. The GUI action includes mouse movement, mouse clicks, keyboard hotkeys, and keyboard typing. The GUI interpreter is a non-language model, proxy agent responsible for interpreting the language model's actions as system operations, and returning the resulting screenshot to the GUI agent. Once the GUI agent decides to complete the task, it will output a message along with a termination signal to end the conversation. This message includes the information that the orchestrator requires the GUI agent to collect, such as a file path, and will be returned to the orchestrator along with a screenshot.

3.2. Workflow and Memory Design



Could you help me unzip the downloaded extension file from /home/user/Desktop/ to /home/user/Desktop/ and configure it in Chrome's extensions?

Unzip the file /home/user/Desktop/helloExtension.zip into /home/user/Desktop/

Task:
Unzip the file /home/user/Desktop/helloExtension.zip into /home/user/Desktop/

Environment:
File path: /home/user/Desktop/helloExtension.zip

Configure the Unpacked Extension in Chrome

~~~
```code
# 1. Go to Desktop
cd /home/user/Desktop || { echo "Desktop directory not found"; exit 1; }
echo "Current directory: $(pwd)"
...
```
~~~

Navigating to Chrome extensions page
LeftClick(622, 164)
WAIT

Task: Configure the Unpacked Extension in Chrome

```
Code output:
Current directory: /home/user/Desktop
ZIP archive found: helloExtension.zip
Archive:  helloExtension.zip
...
Extraction completed successfully.
Extraction contents: ...
```

The extension has been successfully added to the chrome and display correctly.
TERMINATE

Figure 2: Illustration of CoAct-1 workflow. Given a user task, Orchestrator can choose either to call Programmer or GUI Operator to solve a subtask. A programmer can interact with the OS by coding, and a GUI Operator can interact with the OS by performing GUI operations.

For each agent, we use the instance conversation history as their memory. As illustrated in Figure 2, when the Programmer completes an assigned subtask, a language model will summarize the conversation between the coding agent and code interpreter, returning a summary and a screenshot to the Orchestrator as part of the Orchestrator's memory. On the other hand, the GUI Operator will return a message that includes the required information from the Orchestrator, which is also stored as part of the Orchestrator's memory. The Programmer, Orchestrator, and GUI Operator do not share their conversation history. Once the subtask is completed, the system will clean the instance conversation history of the Programmer and the GUI Operator, allowing them to focus solely on the current assigned subtask.

4. Experiments

4.1. Benchmark Dataset

We test the CoAct-1 and other baselines on the OSWorld benchmark (Xie et al., 2024). OSWorld is a scalable real-computer testbed that exposes an OS like Linux to an agent through both pixel streams and an OS shell interface. The benchmark currently comprises 369 tasks that span common productivity tools, IDEs, browsers, file managers, and multi-application workflows, thereby stressing both vision–language grounding and long-horizon planning in heterogeneous GUI environments. Each task in OSWorld ships with (i) a deterministic VM snapshot capturing the initial desktop state, (ii) a natural-language goal that mirrors how end-users phrase requests (e.g., “resize the image to 512×512 and export as PNG”), and (iii) a rule-based evaluator. Tasks cover atomic operations (single-app editing, file I/O) as well as cross-app pipelines, such as “export this spreadsheet, compress it, then e-mail the archive,” providing a realistic spectrum of difficulty.

4.2. Baselines

We compare CoAct-1 with OpenAI CUA 4o (OpenAI, 2025a), GTA-1 (Yang et al., 2025), UI-TARS (Qin et al., 2025), and other baselines listed in Table 1. These baselines represent the forefront of GUI-based task automation, each with a unique approach to interacting with and controlling the computer.

- **OpenAI CUA 4o (SOTA of accessible proprietary model).** This agent uses vision and reasoning to interact with graphical user interfaces, controlling the mouse and keyboard to perform tasks. It is the technology behind services like Operator and ChatGPT agent.
- **GTA-1 (SOTA of grounder + planner)** The GUI Test-time Scaling Agent (GTA-1) is a GUI agent that addresses the challenges of planning ambiguity and action grounding in high-resolution interfaces. To improve accuracy, this agent employs a "test-time scaling" strategy, where it generates multiple possible actions and utilizes a Judge model to select the optimal one. It also leverages the GRPO (Guo et al., 2025) to train a powerful grounder.
- **UI-TARS (SOTA of opensourced unified GUI agent).** This is a self-contained agent that learns from a large dataset of GUI screenshots and computer-using trajectories. It utilizes a unified action model to execute human-like mouse and keyboard operations across various platforms, eliminating the need for external models.

4.3. Implementation Details

Environment We test the CoAct-1 on Linux with an extended RESTful server from OSWorld. Specifically, we implement a remote code interpreter that can take long Python and Bash scripts as input and return the execution result back to the sender. On the other hand, for each task, OSWorld will establish an initial state, such as opening a set of apps or specific websites, or downloading the specified files to a specified location, etc. After the initial state is ready, we will take a screenshot as the initial input along with a user task to CoAct-1 and baselines.

CoAct-1 settings We implement CoAct-1 using AG2 (Wu et al., 2023). We tried different backbones for the Programmer and Orchestrator, including OpenAI o3 (OpenAI, 2025b) and o4-mini (OpenAI, 2025b). For the GUI Operator, we use computer-use-preview, a vision-language action model finetuned by OpenAI for computer use, as the backbone model. We use the o4-mini as the summarizer for summarizing the conversation history between the Programmer and the Orchestrator. We set the max round I for Programmer

Table 1: Comparison of the state-of-the-art methods on the OSWorld (Xie et al., 2024) benchmark. We split the results by steps and show the approach type in the second column. A specialized model means that the model is trained specifically for computer use. We report the success rate (%) as the evaluation metric in the third column. We only include the verified results from `https://os-world.github.io/`.

Agent Model	Approach Type	Success Rate
15 steps		
o3 (OpenAI, 2025b)	General Model	9.10
UI-TARS-72B-DPO (Qin et al., 2025)	Specialized Model	24.00
UI-TARS-1.5-7B (Qin et al., 2025)	Specialized Model	25.70
OpenAI CUA 4o (OpenAI, 2025b)	Specialized Model	26.00
Jedi-7B w/ o3 (Xie et al., 2025b)	Agentic Framework	26.80
Claude 3.7 Sonnet (Anthropic, 2025)	General Model	27.10
Claude 4 Sonnet (Anthropic, 2025)	General Model	31.20
Agent S2 w/ Gemini-2.5-Pro (Agashe et al., 2025)	Agentic Framework	34.64
Agent S2.5 w/ o3 (Agashe et al., 2025)	Agentic Framework	39.00
CoAct-1 w/ o3 & o4mini & OpenAI CUA 4o	Agentic Framework	**39.81**
50 steps		
o3 (OpenAI, 2025b)	General Model	17.17
UI-TARS-72B-DPO (Qin et al., 2025)	Specialized Model	25.80
UI-TARS-1.5-7B (Qin et al., 2025)	Specialized Model	29.40
OpenAI CUA 4o (OpenAI, 2025b)	Specialized Model	31.30
Claude 3.7 Sonnet (Anthropic, 2025)	General Model	35.80
Claude 4 Sonnet (Anthropic, 2025)	General Model	43.90
Agent S2 w/ Gemini-2.5-Pro (Agashe et al., 2025)	Agentic Framework	45.76
GTA-1-7B w/ o3 (Yang et al., 2025)	Agentic Framework	48.59
Jedi-7B w/ o3 (Xie et al., 2025b)	Agentic Framework	50.60
Agent S2.5 w/ o3 (Agashe et al., 2025)	Agentic Framework	54.20
CoAct-1 w/ o3 & o4mini & OpenAI CUA 4o	Agentic Framework	**56.39**
100 steps		
o3 (OpenAI, 2025b)	General Model	23.00
UI-TARS-72B-DPO (Qin et al., 2025)	Specialized Model	27.10
UI-TARS-1.5-7B (Qin et al., 2025)	Specialized Model	29.60
OpenAI CUA 4o (OpenAI, 2025b)	Specialized Model	31.40
Claude 3.7 Sonnet (Anthropic, 2025)	General Model	35.60
Claude 4 Sonnet (Anthropic, 2025)	General Model	41.40
Jedi-7B w/ o3 (Xie et al., 2025b)	Agentic Framework	51.00
GTA-1-7B w/ o3 (Yang et al., 2025)	Agentic Framework	53.10
Agent S2.5 w/ o3 (Agashe et al., 2025)	Agentic Framework	56.00
CoAct-1 w/ o3 & o4mini & OpenAI CUA 4o	Agentic Framework	**59.93**
100+ steps		
CoAct-1 w/ o3 & o4mini & OpenAI CUA 4o	Agentic Framework	**60.76**

as 20, max step K for GUI Operator as 25, and max round J for Orchestrator as 15. Therefore, the number of system interactions, i.e, the number of steps, for CoAct-1 is upper bounded by 375.

Evaluation Protocol We evaluate our method with the rule-based evaluator provided by OSWorld. Internally, every evaluator is expressed as a Boolean expression built from 134 atomic, execution-based evaluators that the authors handcrafted for the benchmark. For a given task, the benchmark composes these atoms with logical AND / OR operators, so a "pass" might require, for instance, (file exported AND MD5 matches) AND (email sent == True).

Table 2: Per-task performance comparison on OSWorld tasks with 100 steps budget. The number beside each subtask is the total number of that subtask. It is obvious that CoAct-1 outperforms the previous SOTA methods in almost all subtasks and achieves the best overall average performance. Subtasks with the most performance gain are Multi-apps, Thunderbird, VLC, and OS, demonstrating the effectiveness of integrating a Programmer.

Methods (100 steps)	Chrome (46)	Calc (47)	Impress (34)	Writer (23)	GIMP (25)	VSCode (23)	Multi Apps (101)	Thunderbird (15)	OS (24)	VLC (17)	Avg.
OpenAI CUA 4o	34.56	12.76	23.19	38.69	65.38	56.52	8.91	46.66	45.83	30.58	31.40
UI-TARS-1.5 (7b)	41.30	19.14	36.17	34.78	42.31	52.17	7.92	53.33	45.83	11.76	29.60
GTA-1	**54.26**	59.57	46.72	60.73	61.53	60.86	38.34	**86.66**	62.50	53.29	53.10
CoAct-1	52.09	**70.21**	**50.27**	**73.91**	**65.38**	**78.26**	**47.88**	66.67	**75.00**	**66.07**	**59.93**

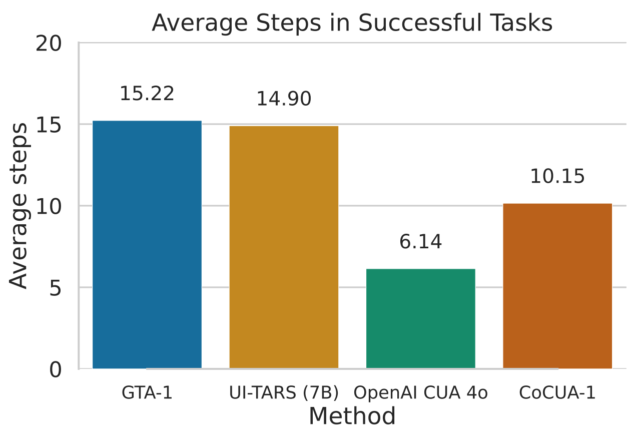
4.4. Results

Table 1 presents a comprehensive comparison of CoAct-1 with state-of-the-art models on the OSWorld benchmark, demonstrating that our agent establishes a new standard for performance. CoAct-1 achieves a top success rate of 60.76% in the 100+ step category. The results show that CoAct-1 consistently outperforms other leading methods across various maximum step allowances. For instance, at a 100-step budget, CoAct-1 reaches a 59.93% success rate, which is a significant improvement over other agentic frameworks like Agent S2.5 w/ o3 (56.00%) and GTA-1-7B w/ o3 (53.10%). This robust performance is also evident at the 50-step limit, where CoAct-1 achieves 56.39%, surpassing all other agents.

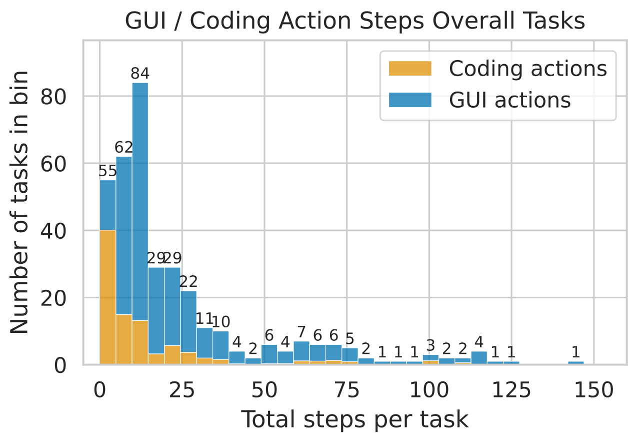
On the other hand, as shown in Table 2, CoAct-1 achieves the highest overall average success rate of 59.93%. This represents a significant margin of improvement over other leading agents, including GTA-1 (53.10%), OpenAI CUA 4o (31.40%), and UI-TARS-1.5 (7b) (29.60%). The advantages of our hybrid approach are most evident in task categories that benefit from programmatic control. In the "Multi Apps" category, CoAct-1's performance of 47.88% substantially exceeds that of GTA-1 (38.34%). Similarly, major performance gains are observed in application-specific and OS-level tasks. For VLC, CoAct-1 scores 71.96% compared to GTA-1's 53.29% ; and for general OS tasks, CoAct-1 reaches 75.00%. CoAct-1 also demonstrates superior performance in other applications like LibreOffice Calc (70.21%) and Writer (73.91%). These results underscore the effectiveness of integrating a Programmer agent, which enables CoAct-1 to handle a more diverse and complex range of tasks by augmenting GUI manipulation with the precision of direct coding actions.

4.5. Discussion

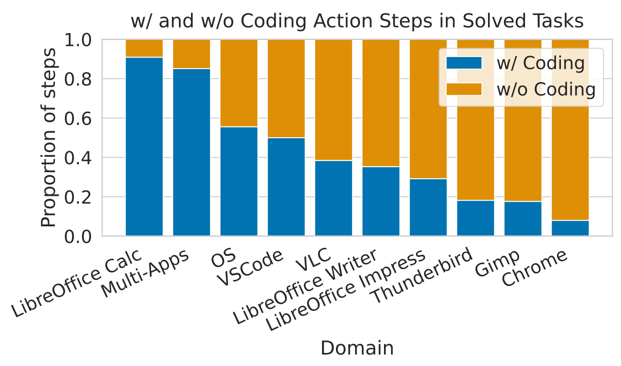
Efficiency Analysis The analysis of CoAct-1's operational efficiency, illustrated in Figure 3, reveals that our hybrid approach is substantially more efficient than leading GUI-only agents. This efficiency is a key factor in its improved success rate. As shown in Figure 3a, CoAct-1 solves tasks with an average of 10.15 steps. This represents a significant improvement over other high-performing agents like GTA-1, which requires 15.22 steps, and UI-TARS, which needs 14.90 steps on average. While OpenAI CUA 4o averages fewer steps (6.14) , its overall success rate is much lower compared to CoAct-1's (31.40% v.s. 59.93% on 100 steps). This indicates that CoAct-1's efficiency is coupled with greater effectiveness. The source of this efficiency lies in the strategic use of coding actions. Figure 3b supports this by showing that coding actions help keep the total steps per task relatively low. This efficiency is crucial for robust performance. Figure 3c shows that coding is particularly beneficial in complex domains like "LibreOffice Calc", "Multi-apps" and direct OS interactions, where a large proportion of tasks are solved with code. A single script can replace a long and error-prone sequence of GUI clicks, streamlining the workflow. Figure 3d illustrates a clear trend: tasks that require more actions are more likely to fail. By reducing the total number of steps, the hybrid approach not only accelerates task completion but also minimizes the opportunities for error. The ability to dynamically



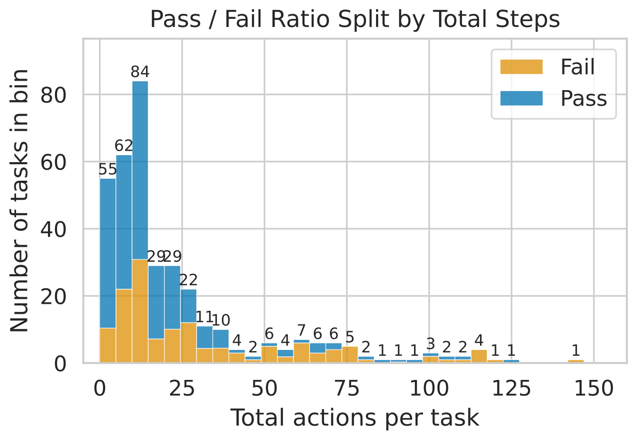
(a) Average steps for a successful task. CoAct-1 can solve tasks with lower average steps than previous SOTA agentic frameworks. On the other hand, CoAct-1 higher steps than OpenAI CUA 4o but also higher accuracy.



(b) GUI/code ratio split by total action of tasks. Coding helps reduce the total action steps.



(c) This chart illustrates the domains where coding actions are most frequently applied.



(d) Error rate distribution split by total action of tasks. The error rate is positive related to total actions.

Figure 3: CoAct-1 Efficiency and Step Modality Analysis.

select the most appropriate action—either a direct coding command or a GUI interaction—is fundamental to the enhanced efficiency and reliability of CoAct-1.

CoAct-1 with different backbone We investigated the impact of different backbone models on the performance of CoAct-1, with results presented in Table 3. The study reveals that the choice of models for the Orchestrator, Programmer, and GUI Operator significantly influences overall effectiveness. When a uniform backbone, o4-mini, was used for both Orchestrator and Programmer, the system achieved a performance score of 47.41%. Performance improved to 53.12% when the o3 model serving as both the Orchestrator and Programmer. The highest performance of 60.76% was achieved by retaining OpenAI CUA 4o for the GUI Operator and o3 for the Orchestrator, while upgrading the Programmer to the o4-mini model. This highlights that the system benefits from a powerful, vision-centric model for GUI operations and that enhancing the capability of the Programmer agent yields significant performance gains.

Table 3: Performance of CoAct-1 with different backbone model for each participant agent. Powerful Orchestrator significantly help improve the performance on OSWorld.

	GUI Operator	Orchestrator	Programmer	Performance
CoAct-1	OpenAI CUA 4o	o4-mini	o4-mini	47.41
		o3	o3	53.12
		o3	o4-mini	60.76

4.6. Case Study

To better understand the capabilities and limitations of CoAct-1, this section analyzes failure cases observed during evaluation. Generally, errors in task completion arise from two primary challenges: high-level and ambiguous queries.

High-level query A high-level query is one where the user's instruction does not directly map to a sequence of actions. Instead, it requires the agent first to infer the user's underlying intent and the broader context before it can devise a solution. For instance, one task in the VSCode domain instructed the agent: "Please help me modify the setting of VSCode to keep my cursor focused on the debug console when debugging in VSCode, instead of automatically focusing back on the Editor." In this scenario, the Orchestrator delegated the task to the Programmer. The Programmer attempted to find the relevant setting by searching for keywords like "debug" and "console". However, it failed to make the conceptual leap that the debugging process relates to "breakpoints." Consequently, it overlooked the correct setting, "focusEditorOnBrake," leading to the task's failure. This case highlights a limitation in the agent's ability to reason about concepts that are not explicitly mentioned in the query.

Ambiguous query An ambiguous query is a user request that is vague or omits critical information necessary for successful task completion. Resolving ambiguity often requires the agent to correctly infer the user's intent, which can also involve safety considerations. An example of this occurred in a VSCode task with the instruction: "Please help me modify VSCode setting to hide all "__pycache__" folders in the explorer view." The Orchestrator assigned this subtask to the Programmer. The Programmer successfully identified the need to modify a settings file but incorrectly altered the workspace-specific settings instead of the global user settings. This misinterpretation of the query's scope resulted in the task failing. This illustrates the challenge the agent faces in disambiguating the user's intent when multiple valid interpretations exist.

5. Conclusions

In this work, we introduced CoAct-1, a novel multi-agent system designed to address the inherent inefficiency and brittleness of agents that rely exclusively on GUI manipulation. Our multi-agent system features an Orchestrator that dynamically delegates subtasks to a GUI Operator or a Programmer. Our extensive evaluation on the OSWorld benchmark confirms the effectiveness of this approach. CoAct-1 achieves a new state-of-the-art success rate of 60.76%, significantly outperforming previous leading methods. The performance gains were particularly pronounced in categories involving OS-level interactions, multi-application workflows, and other tasks where the Programmer agent could leverage direct programmatic execution.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

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

Anthropic. Claude opus 4 & claude sonnet 4 system card. System card, Anthropic, May 2025. URL `https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf`. Accessed 4 Aug 2025.

Anthropic. Claude 3.7 sonnet and claude code. Technical report, Anthropic, 2025. URL `https://www.anthropic.com/news/claude-3-7-sonnet`. System Card.

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

Deepmind. Introducing gemini 2.0: our new ai model for the agentic era. Technical report, Deepmind, 2025a. URL `https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/#project-astra`.

Deepmind. Gemini 2.5: Our most intelligent ai model. Technical report, Deepmind, 2025b. URL `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/`.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024. URL `https://arxiv.org/abs/2410.05243`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*, 2025.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*, 2024.

OpenAI. Gpt-4o system card, 2024.

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

OpenAI. Openai o3 and o4-mini system card. Technical report, OpenAI, 2025b. URL `https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf`. System Card.

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

Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, Xing Zhou, Dongrui Liu, Ling Yang, Yue Wu, Kaixuan Huang, Shilong Liu, Hongru Wang, and Mengdi Wang. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025. URL https://arxiv.org/abs/2505.20286.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. Adaptive in-conversation team building for language model agents, 2025. URL https://arxiv.org/abs/2405.19425.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

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. Scaling computer-use grounding via user interface decomposition and synthesis, 2025a. URL https://arxiv.org/abs/2505.13227.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*, 2025b.

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.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, Ran Xu, Liyuan Pan, Caiming Xiong, and Junnan Li. Gta1: Gui test-time scaling agent, 2025. URL https://arxiv.org/abs/2507.05791.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. Aria-ui: Visual grounding for gui instructions. *arXiv preprint arXiv:2412.16256*, 2024.

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

Jieyu Zhang, Ranjay Krishna, Ahmed H. Awadallah, and Chi Wang. Ecoassistant: Using llm assistant more affordably and accurately, 2023. URL https://arxiv.org/abs/2310.03046.

Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights, 2024. URL https://arxiv.org/abs/2402.11359.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems, 2025b. URL https://arxiv.org/abs/2505.00212.

Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Ming Li, Qilong Wu, Kaipeng Zhang, and Chen Wei. Pyvision: Agentic vision with dynamic tooling, 2025. URL https://arxiv.org/abs/2507.07998.