BROWSER USE AND COMPUTER USE

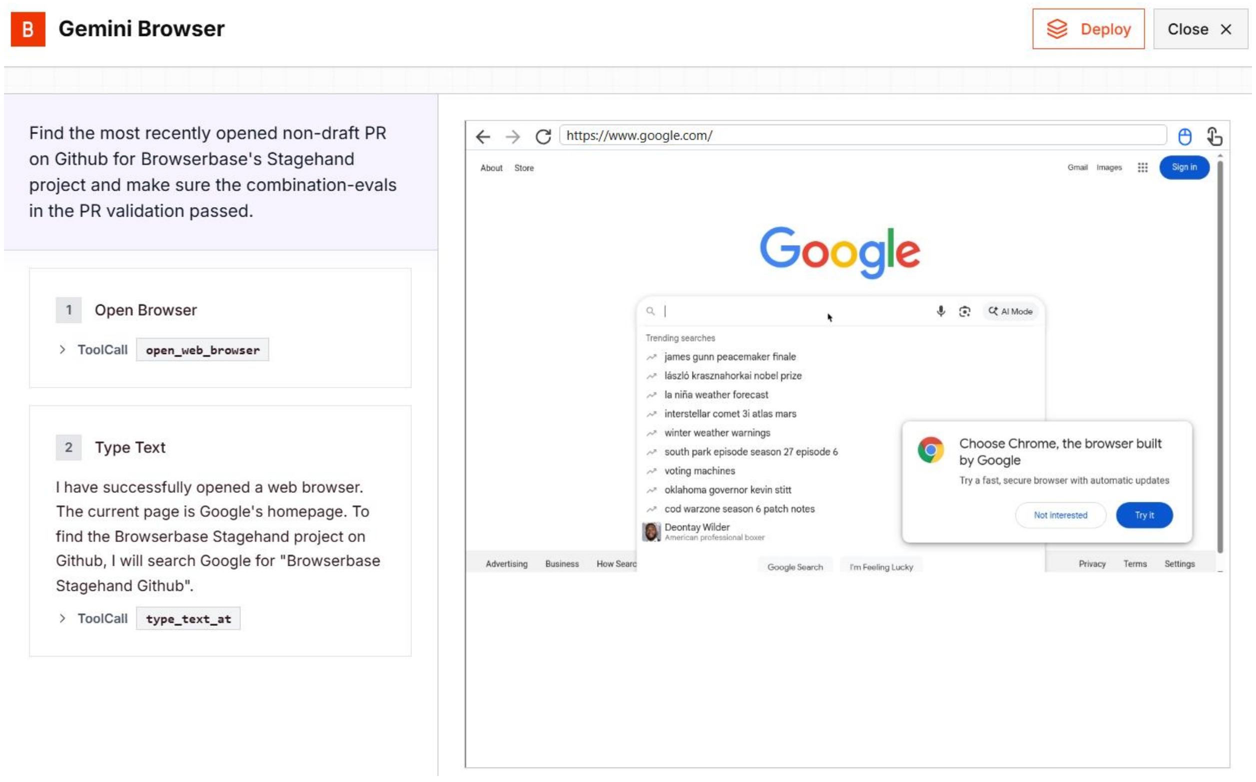
1. Introduction

In recent years, the proliferation of intelligent agents capable of interacting with web browsers and operating systems has captured substantial attention both in industry and academia. The emergence of browser use reflects a new wave of web automation that promises to reshape human–web interaction paradigms. Alongside this, computer use agents are being elevated from script-based tools to sophisticated systems leveraging large language models and multimodal reasoning to execute complex, multi-step workflows. These developments are not merely incremental: they hint at a shift toward agents that act as intermediaries in digital environments, rather than passive interfaces. The trend has garnered significant investment and strategic focus, as major technology firms increasingly embed agentic capabilities into mainstream products and service ecosystems. At the same time, early deployments and demonstrations underscore persistent challenges in robustness, usability, and security.

2. Browser use and Computer use

2.1. Definitions and Use cases

Browser use agents, often referred to as web agents, are intelligent systems designed to automate interactions with web browsers. Their primary role is to carry out repetitive or structured tasks that would otherwise require direct human input. Typical applications include logging into websites, completing online forms, extracting or scraping information, executing online purchases, and conducting automated user interface testing. By performing these activities efficiently and reliably, browser use agents have become an essential tool for streamlining web-based workflows and enabling large-scale automation in both research and industry settings.


B
Gemini Browser
Deploy
Close ×
https://www.google.com/
About
Store
Gmail
Images
Sign in
Google
AI Mode
Trending searches
james gunn peacemaker finale
lászló krasznahorkai nobel prize
la niña weather forecast
interstellar comet 3i atlas mars
winter weather warnings
south park episode season 27 episode 6
voting machines
oklahoma governor kevin stitt
cod warzone season 6 patch notes
Deontay Wilder
American professional boxer
Choose Chrome, the browser built by Google
Try a fast, secure browser with automatic updates
Not interested
Try it
Advertising
Business
How Searc
Google Search
I'm Feeling Lucky
Privacy
Terms
Settings

Find the most recently opened non-draft PR on Github for Browserbase's Stagehand project and make sure the combination-evals in the PR validation passed.

1 Open Browser

› ToolCall `open_web_browser`

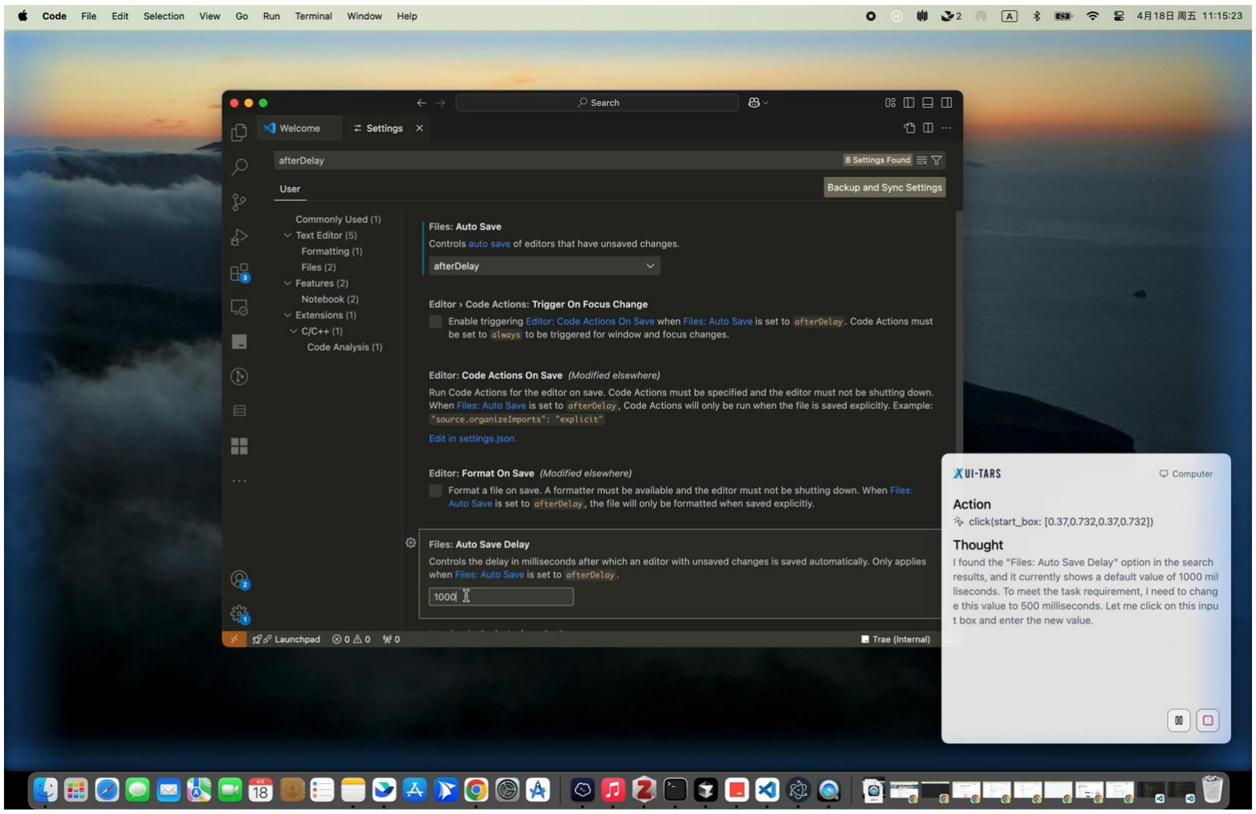
2 Type Text

I have successfully opened a web browser. The current page is Google's homepage. To find the Browserbase Stagehand project on Github, I will search Google for "Browserbase Stagehand Github".

› ToolCall `type_text_at`

Figure 1. Illustration of a browser use agent, Gemini Browser, performing tasks on the Github website within the browser environment

Computer use agents, or more broadly operating system (OS) agents, are intelligent agents capable of controlling and interacting with the entire computer environment rather than being confined to web browsers. In contrast to browser use agents, which operate within the boundaries of a web interface, computer use agents can manage system-wide operations, encompassing multiple applications and processes simultaneously. Their functionality may subsume browser use capabilities as a subset of their broader control. Typical use cases include cross-application automation, such as coordinating tasks across Word, Excel, Visual Studio Code, Photoshop, and file management systems or operating seamlessly within virtual machines. This versatility allows computer-use agents to serve as general-purpose automation frameworks, bridging diverse software ecosystems under a unified agentic interface.



Files: Auto Save
Controls auto save of editors that have unsaved changes.
afterDelay

Editor › Code Actions: Trigger On Focus Change
Enable triggering Editor: Code Actions On Save when Files: Auto Save is set to `afterDelay`. Code Actions must be set to `always` to be triggered for window and focus changes.

Editor: Code Actions On Save *(Modified elsewhere)*
Run Code Actions for the editor on save. Code Actions must be specified and the editor must not be shutting down. When Files: Auto Save is set to `afterDelay`, Code Actions will only be run when the file is saved explicitly. Example: `"source.organizeImports": "explicit"`
Edit in settings.json

Editor: Format On Save *(Modified elsewhere)*
Format a file on save. A formatter must be available and the editor must not be shutting down. When Files: Auto Save is set to `afterDelay`, the file will only be formatted when saved explicitly.

Files: Auto Save Delay
Controls the delay in milliseconds after which an editor with unsaved changes is saved automatically. Only applies when Files: Auto Save is set to `afterDelay`.
1000

UI-TARS

Action
click(start_box: [0.37,0.732,0.37,0.732])

Thought
I found the "Files: Auto Save Delay" option in the search results, and it currently shows a default value of 1000 mil liseconds. To meet the task requirement, I need to chang e this value to 500 milliseconds. Let me click on this inpu t box and enter the new value.

Figure 2. Illustration of a computer use agent, UI-TARS Desktop, performing a task using the Visual Studio Code application within the computer environment

2.2. Methods

Most existing researchs conceptualize the research landscape through three main perspectives: the *environment perspective*, the *interaction perspective*, and the *agent perspective*.

Environment Perspective: This perspective discusses the properties of computer and web environments, identifies observation and action types that are shared across different domains.

Interaction Perspective: This perspective examines the observation spaces O and action spaces A through which agents interact with their environments. It discusses methods like observation simplification and action grounding, which simplify the task.

Agent Perspective: This perspective focuses on the core components of browser use and computer use agents. It dissects how an agent acts (policy) and how it learns to act (learning strategy), including the use of planning and demonstrations.

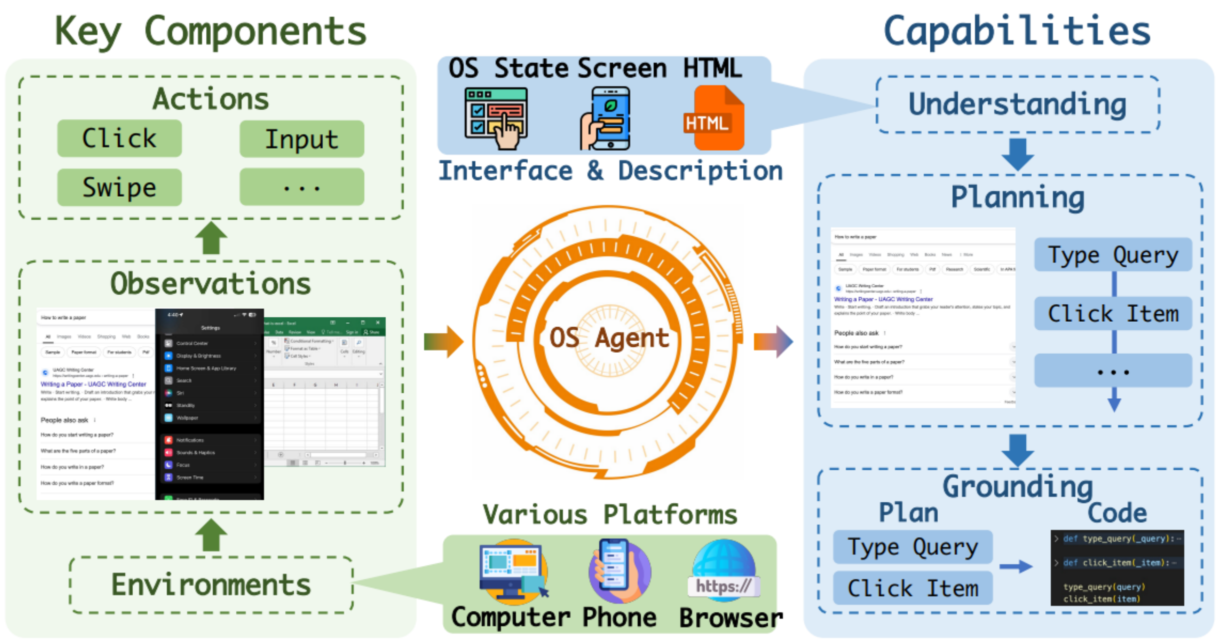


Figure 3. Simplified illustration of the Perspectives in the paper OS Agents: A Survey on MLLM-based Agents for Computer, Phone and Browser Use

2.2.1. Interaction Perspective

2.2.1.1. Observation space

The observation space encompasses the information agents can access about the system's state and user activities. Observation includes capturing information from the environment, such as screen images, or textual data, such as the description of the screen and the HTML code in web-based contexts.

Textual representation: Early works are limited by the fact that LLMs could only process textual input. Therefore, they mainly rely on using tools to convert environments into text descriptions, often represented in a structured format, such as HTML, DOM, or accessibility tree. Textual representations can reveal information visually hidden in images (items in a collapsed menu), possess an inherent hierarchical structure, and include explicit semantic information. However, they can suffer from reduced information density (verboseness), structural inconsistency across applications and websites, omission of critical visual layout and spatial relationships. Some screen components (like embedded plugins) may entirely lack textual representation. Some text-based agents have difficulty dealing with complex modern sites.

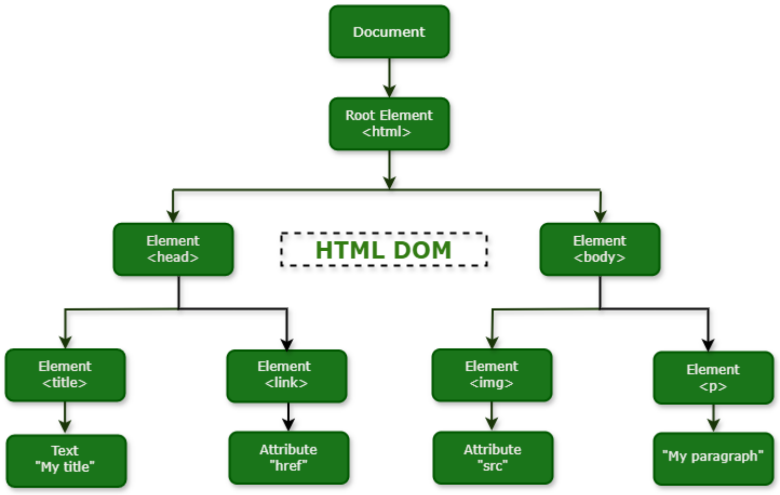


Figure 4. Document Object Model (DOM)

GUI screenshot representation: Despite the remarkable success of text-based agents, leveraging the textual metadata of the environment usually fails to align closely with human cognitive processes since the GUI are inherently visual. Additionally, due to drawbacks of using textual representations described above and performance using text-based agents drop significantly on realistic benchmarks like Mind2Web (where HTML is often more verbose and inconsistent), research is increasingly treating GUI screenshots as the perception input for agents. But due to practical concerns, screenshots are often simplified by downsampling or cropping.

Indirect: Indirect observations that do not describe the current screen but information of the computer state, by accessing network traffic with website or stored files with computer.

Multi-modal representation: To enhance agents' understanding and grounding ability without fine-tuning visual encoders, existing research focuses on using prompting techniques to describe GUI screenshots. These descriptions can generally be categorized into three types: (1) Visual description. Most research uses Set-of-Mark (SoM) prompting to enhance agents' visual grounding ability. (2) Semantic description. Some studies improve agents' understanding and grounding ability by adding descriptions of these interactive elements. (3) Dual description. Dual description combines both visual and semantic information to improve agents' understanding and grounding of the visual environment.

2.2.1.2. Action space

The action space can be categorized into three main types: Input operations, Navigation operations and Extended operations.

Input operations: encompass interactions via mouse/touch and keyboard.

Navigation operations: enable agents to traverse targeted platforms and acquire sufficient information for subsequent actions. While browser use agents can only perform web-specific navigation across websites inside the browser, computer use agents can navigate directly through the OS.

Extended operations: provide additional capabilities beyond standard interface interactions, enabling more flexible and powerful agent behaviors. These operations primarily include (1) code execution capabilities that allow agents to dynamically extend their action space beyond predefined operations, enabling flexible and customizable control through direct script execution and command interpretation, and (2) API integration features that expand agents' capabilities by accessing external tools and information resources, facilitating interactions with third-party services and specialized functionalities. Most of these operations can only be done with computer use agents, although some browser use agents can ultilize API tools with tool-based interacting methods in addition to web browsing-based features.

2.2.2. Agent Perspective

2.2.2.1. Main approachs:

There are three main appoachs: General model, Specialized model and Agentic framework.

General model: is a pre-trained foundation model (LLM/VLM) with broad, general-purpose capabilities. Browser or computer use is one capability that can be elicited via prompting; the model itself can still perform other tasks such as dialogue and code generation. Agents rely on the model's broad knowledge and in-context learning, typically utilizing history-based policies.

Specialized model: is trained specifically to serve as a browser or computer use agent; other capabilities are out of scope and are not emphasized in the corresponding reports. It uses predefined output possibilities and learns via environment learning techniques like Reinforcement Learning (RL). Agents using this model type commonly use state-based policies.

Agentic framework: organize one or more General and Specialized models into a structured workflow – commonly, a GPT-family model acts as the planner while a proprietary or task-specific model serves as the grounder. This framework contains four core components:

- **Perception**: This component collects and analyzes information from the environment, extracts relevant data for planning, action, and memory optimization. It is categorized by input modality.
- **Planning**: Planning develops a sequence of actions to achieve a goal, breaking complex tasks into manageable sub-tasks. Global Planning: generates a plan once (often using Chain-of-Thought (CoT) prompts) and executes it without adjustments based on environmental changes. Iterative Planning: continuously iterate and adjust their plans based on historical actions or changes in the environment, utilizing methods like ReAct (integrating reasoning with action outcomes).
- **Memory**: The Memory module saves useful information for task performance, environment adaptation, and optimization. Memory Sources: Internal Memory (stores information during task completion, such as action history and screenshots) and External Memory (provides long-term knowledge support through knowledge bases, documents, and online tools). Memory Optimization: Key strategies include Management (processing and abstracting information, consolidating content), Growth Experience (revisiting steps to analyze successes/failures and optimize future performance, sometimes returning to a previous state), and Experience Retrieval (retrieving similar past experiences from long-term memory).
- **Action**: The Action space defines the interfaces for agents to interact with different platforms.

In terms of Agentic Framework, CoAct-1 hase been presented with a new robust and flexible paradigm by enabling agents to use coding as an enhanced action, synergistically combining conventional GUI-based control with direct programmatic execution. Existing computer-using agents, which rely primarily on GUI, often struggle with efficiency and reliability when facing complex, long-horizon tasks. Workflows like complex data processing or file management that require intricate sequences of precise GUI manipulations are susceptible to failure due to visual grounding ambiguity and the high probability of error accumulation over a long action horizon. To solve this problem, CoAct-1 is designed as a novel multi-agent system that with its architecture is 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.

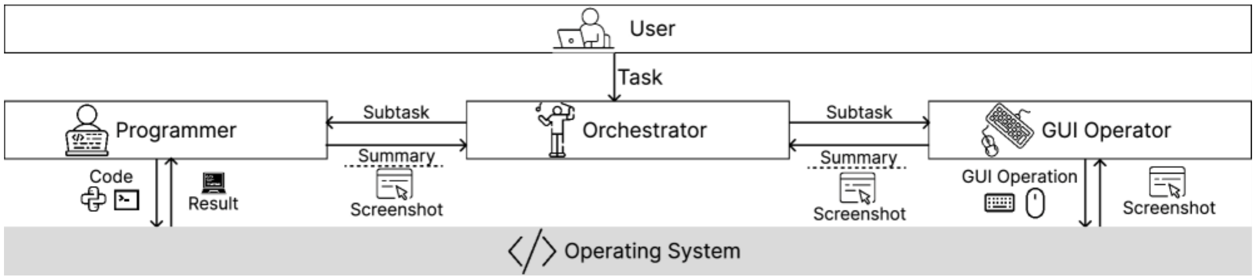


Figure 5. System design of CoAct-1

Recently, Agent S3 introduced a novel wide scaling pradigm for computer use agents, showing that generating multiple trajectories in parallel and selecting among them substantially improves robustness and task success rates. To realize this, the authors proposed Behavior Best-of-N (bBoN), a framework that transforms dense trajectories into compact behavior narratives and leverages them for principled trajectory selection. They also improve the baseline of previous version, ultizing Coding Agent (like CoAct-1) and Flat Policy (remove hierarchical planning in favor of this new policy that can replan at any time). However, this method still needs to be enhanced more to adapt to real-world applications.

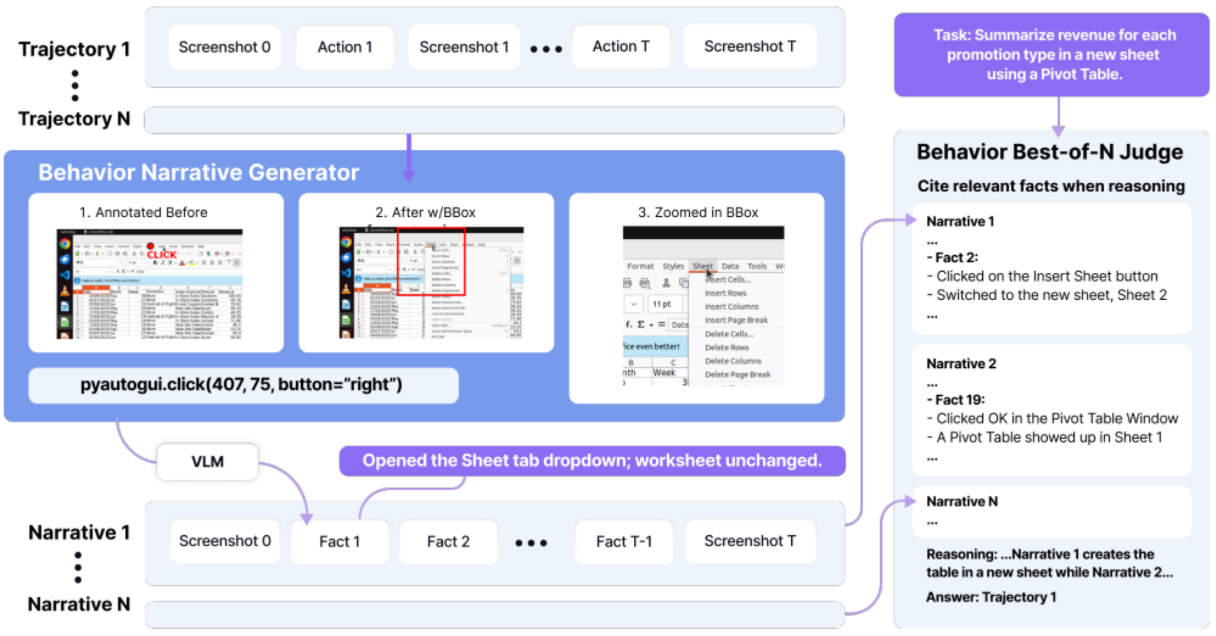

Trajectory 1
Trajectory N
Screenshot 0
Action 1
Screenshot 1
Action T
Screenshot T
Task: Summarize revenue for each promotion type in a new sheet using a Pivot Table.
Behavior Narrative Generator
1. Annotated Before
2. After w/BBox
3. Zoomed in BBox
pyautogui.click(407, 75, button="right")
VLM
Opened the Sheet tab dropdown; worksheet unchanged.
Narrative 1
Narrative N
Fact 1
Fact 2
Fact T-1
Behavior Best-of-N Judge
Cite relevant facts when reasoning
Narrative 1
- Fact 2:
- Clicked on the Insert Sheet button
- Switched to the new sheet, Sheet 2
Narrative 2
- Fact 19:
- Clicked OK in the Pivot Table Window
- A Pivot Table showed up in Sheet 1
Narrative N
Reasoning: ...Narrative 1 creates the table in a new sheet while Narrative 2...
Answer: Trajectory 1

Figure 6. Behavior Best-of-N generates multiple rollouts of screenshots and actions, which are converted into concise behavior narratives describing the changes between before and after states. The judge then compares these narratives to select the best overall trajectory.

2.2.2.2. Policy – How to act:

A policy is the decision-making component of an agent, determining how it acts in a given situation. In most surveys and research, three main types of policies are identified: memoryless, history-based, and state-based.

Memoryless policy predicts the next action solely from the current observation without considering any past information. While this approach simplifies model architecture, it is generally inadequate for complex computer control tasks that depend on remembering previous actions or observations, such as tracking items already added to a shopping cart.

History-based policy is developed to overcome this limitation**,** maintains a sequence of past observations and actions, known as history, which the agent uses to inform its next move. Because foundation models have limited context lengths, simplified histories are often employed, keeping only past actions, recent observations, or compressed representations of earlier inputs.

State-based policy summarizes past information into an internal state that evolves over time through a state-update function. This internal state helps the agent retain relevant context while predicting future actions and is often implemented using recurrent neural networks or similar architectures.

Mixed policies combine both strategies, incorporating elements of history-based inputs and external state representations to guide decisions and update the agent's internal understanding of its environment.

2.2.2.3. Learning trategy – How to learn to act:

An agent's learning strategy defines how it learns to act and can generally be divided into three steps: general pre-training, environment learning, and episodic improvement.

a) General Pre-training:

In the general pre-training phase, foundation agents rely on large foundation models with extensive knowledge and in-context learning abilities. Such models can perform tasks in computer environments (like web navigation or form filling) without explicit fine-tuning, adapting only through prompts. By contrast, specialized agents may be trained from scratch or initialized from pre-trained backbones such as vision or text encoders, which require fine-tuning for specific tasks. The choice of foundation or backbone model depends on the observation and action spaces. For example, GPT-4 can function as a multimodal agent, coding-oriented models can generate executable code, and vision or text backbones can be fine-tuned for perception-based tasks. Some models are even pre-trained on specific data types, such as HTML, to enhance their domain expertise.

b) Environment training:

The environment learning stage focuses on teaching the agent how to act through experiences within different environments. Three primary strategies are used: **reinforcement learning (RL)**, **behavioral cloning (BC)**, and **long-term memory utilization**.

In **Reinforcement learning**, the agent learns via trial and error to maximize rewards in simulated, controlled environments. However, because rewards in computer tasks are often sparse, methods like human-labeled demonstrations, reward shaping, or curriculum learning are used to guide exploration and accelerate progress. RL's main strength lies in autonomous exploration but its dependence on predefined reward structures limits its applicability to real-world computer control.

In contrast, **Behavioral cloning** teaches an agent to imitate human behavior through supervised learning on demonstration data consisting of recorded observations and actions. BC can serve as an initial phase before RL or as a complete learning method for simpler tasks. It does not require live interaction with the environment, making it practical for uncontrolled or real-world systems. BC strategies differ in model fine-tuning scope, some update the entire network, while others adjust specific modules or apply efficient low-rank adaptations.

Long-Term Memory plays an important role in enhancing an agent's ability to learn and adapt within computer environments. Foundation models, with their strong few-shot and in-context learning abilities, can store and reuse experiences to improve action prediction without explicit fine-tuning. Through in-context learning, agents autonomously collect and retrieve relevant experiences, which are typically represented in two forms: environment transitions (triples of pre-action observation, action, and post-action observation, capturing how actions change the environment) and task demonstrations (tuples recording successful instruction and action sequences for solving specific tasks).

c) Episodic training:

Episodic improvement refers to an agent's short-term ability to enhance its performance within a single interaction episode by reasoning, adapting, or planning, without retaining knowledge afterward. This process effectively exchanges additional computation at test time for better execution of the current task. General foundation model agents commonly achieve episodic improvement through in-context learning, using techniques such as instruction tuning and few-shot learning, while specialized agents rarely apply it directly.

In-Context Learning (Instruction Tuning): The agent improves performance through prompt design, where humans or other models craft textual or visual instructions to adapt the foundation model to specific environments.

In-Context Learning (Demonstrations): The agent learns from example trajectories (either human-crafted, retrieved, generated, or self-collected) to guide actions, often simplified and sometimes including reasoning steps for better decision-making.

Planning: The agent enhances task execution by reasoning ahead, either implicitly or through explicit textual plans, decomposing goals into sub-tasks or simulating future outcomes to decide optimal actions.

3. Evaluation and Benchmarks

3.1. Evaluation

During evaluation, agents receive task instructions and environment inputs, then perform continuous actions until the task is completed. Their observations, actions, and environmental data are collected to compute performance metrics. These metrics can be divided into **task-level**, **step-level** and **other**.

Task-level metrics evaluate how effectively an agent completes a full task. The most common is task success rate, which measures the percentage of fully completed tasks, determined automatically in controlled environments or by human judgment in online settings (more accurate measure of true capability but is costly and lacks consistent reproducibility). Other metrics include task progress, estimating partial completion, and average reward, capturing mean rewards in controlled setups.

Step-level metrics assess how accurately the agent predicts individual actions within tasks. The main metric is step success rate (also called partial match or action accuracy), with results averaged using either macro averaging (averaging within each trajectory before across tasks) or micro averaging, which treats all steps equally. Additional measures such as action F1, recall, or element accuracy provide finer action-level evaluation.

Other metrics capture broader performance aspects, such as efficiency (such as number of API calls) and safeguard rate, which measures how reliably the agent detects and confirms sensitive or risky actions before execution.

3.2. Dataset

3.2.1. Datasets

With computer use agents, two main dataset types are identified: **controlled environments** and **offline datasets**.

Controlled environments are simulated settings where agents can freely act without real consequences, allowing for reinforcement learning, safe experience collection, and action planning during inference. They are expensive to build but enable full exploration of the environment.

Offline datasets, in contrast, consist of recorded human demonstrations of computer tasks. Agents learn from these pre-collected trajectories without directly interacting with the environment, making training safe but limited, since not all possible interactions are captured and only one valid trajectory per task is shown. Both dataset types play complementary roles in developing and evaluating computer control agents.

In addition to two above types of datasets, browser use agents can also be validated by **Open-world** datasets. Open-world datasets are collected or utilized directly from real, dynamic websites where content and structure continuously evolve. Unlike simulated or static settings, these datasets expose agents to live web environments, requiring them to handle variations in page layout, DOM structure, language, and interface design. As a result, they provide a more realistic measure of an agent's adaptability and generalization ability in real-world contexts. However, they also pose greater challenges for evaluation, reproducibility, and safety, necessitating secure sandboxing mechanisms and stricter testing protocols to prevent unintended interactions with real systems.

3.2.2. Benchmarks

To comprehensively evaluate the performance and capabilities of browser and computer use agents, researchers have developed a variety of benchmarks. These benchmarks construct various environments, based on different platforms and settings, and cover a wide range of tasks.

Computer platforms are complex due to the diversity of operating systems and applications. Efficient computer benchmarks need to handle the wide variety and complexity of real-world computing environments, which span different operating systems, interfaces, and applications. Some popular benchmarks: OSWorld, OS-Copilot, Window agent arena…

Browser platforms are essential interfaces to access online resources. Webpages are open and built with HTML, CSS, and JavaScript, making them easy to inspect and modify in real-time. Some popular benchmarks: WebArena/VisualWebArena, WebLINX, Webshop, Mind2Web…

4. Limitations

4.1. Technical

Although both browser and computer use agents have achieved impressive performance on many benchmarks (such as CoAct-1 on OSWorld and Browser Use on WebVoyager), they still struggle with **high-level or ambiguous queries**. These queries often require deeper contextual understanding, multi-step reasoning, or clarification of vague user intentions, capabilities that remain limited in current agent architectures. As a result, even state-of-the-art systems may produce incorrect actions, incomplete plans, or irrelevant responses when faced with open-ended or underspecified instructions.

Environmental adaptation also remains an open problem. In real business or software environments, agents must be customized either through environment learning (which is data- and cost-intensive) or prompt-based adaptation (which provides quick results but limited improvement). Although foundation agents achieve strong out-of-the-box performance, they often fail to generalize across diverse user setups such as varying screen resolutions, multi-monitor systems, or different websites.

Furthermore, production environments are **non-stationary**, as applications continuously update their interfaces and behavior. A production-ready agent must therefore adapt autonomously or receive regular developer updates.

Speed, cost, and availability are also major practical considerations. High-performing agents consume significant computational and monetary resources, especially when relying on large foundation models through APIs. Additionally, dependency on cloud APIs introduces latency and availability issues tied to internet stability and external service uptime.

Many agents today still perform insufficiently in providing **personalized** experience to users and **self-evolving** over user interactions. The challenge of expanding memory to multi-modal forms and ensuring the efficient management and retrieval of this complex memory.

With browser use agents, a key limitation lies in their dependence on sandboxed environments, which restrict interaction only in the browsers. On the other hand, although computer use agents have a wider range of action, they mostly take up the user interface like mouse.

4.2. Safety and Privacy risks

Privacy poses serious concerns. Many state-of-the-art LLMs (such as GPT-4V) operate only via remote APIs, meaning user screenshots or sensitive data might be transmitted over the internet. This creates risks of unintended data exposure, particularly when autonomous agents access confidential documents without explicit user consent. Unlike conventional applications where users control data sharing, autonomous agents blur these boundaries, demanding stricter privacy safeguards and compliance mechanisms in real-world deployment.

Some studies reveal multiple attack vectors, including **Web Indirect Prompt Injection (WIPI)**, adversarial images, and **environmental injection attacks** that can manipulate agents or steal information. Additional threats such as backdoor attacks, adversarial pop-ups, and LLM jailbreaks in browser contexts further expose vulnerabilities.

While general LLM security frameworks exist, defense mechanisms tailored to browser and computer use agents remain underdeveloped, emphasizing the need for specialized protection against injection and backdoor threats.

Moreover, these agents can potentially be exploited to **automate malicious activities**, posing serious security threats. For instance, attackers could manipulate them to send phishing emails, spread malware, scrape private data from websites, or manipulate online transactions without human oversight. Because such agents can autonomously navigate applications and execute commands, even a small vulnerability (like a prompt injection or compromised webpage) could allow them to perform harmful actions at scale, making security safeguards and behavioral monitoring essential for their safe deployment.

5. Future directions

Future research browser and computer use agents are increasingly oriented toward building **trustworthy, high-performing systems** that can operate reliably in **real-world application environments**. Current benchmarks show that agents based on Agentic Frameworks often achieve superior results, demonstrating stronger reasoning and adaptability. In the computer use domain, integrating coding as a core action has proven to be a powerful, efficient, and scalable pathway toward generalized computer automation, enabling agents to generate and execute code dynamically to accomplish complex tasks. Moving forward, research should focus on optimizing the backbone models that power specialized agents (enhancing their robustness, adaptability, and efficiency) for planning components of the system to bridge the gap between controlled experimentation and practical deployment.

Another key future direction is to overcome the hurdles related to **memory expansion and management**, allowing agents to evolve sophisticated self-evolution mechanisms. Success in this area will enable agents to provide more personalized, dynamic, and context-aware assistance by learning and adapting continuously to the user's needs and environment.

Future research should also prioritize the development of **robust defense mechanisms** specifically tailored to the unique vulnerabilities of these agents, such as injection attacks and backdoor exploits. This is necessary because defenses developed for general LLM agents are still nascent or insufficient for browser and computer use agents.

6. Current research

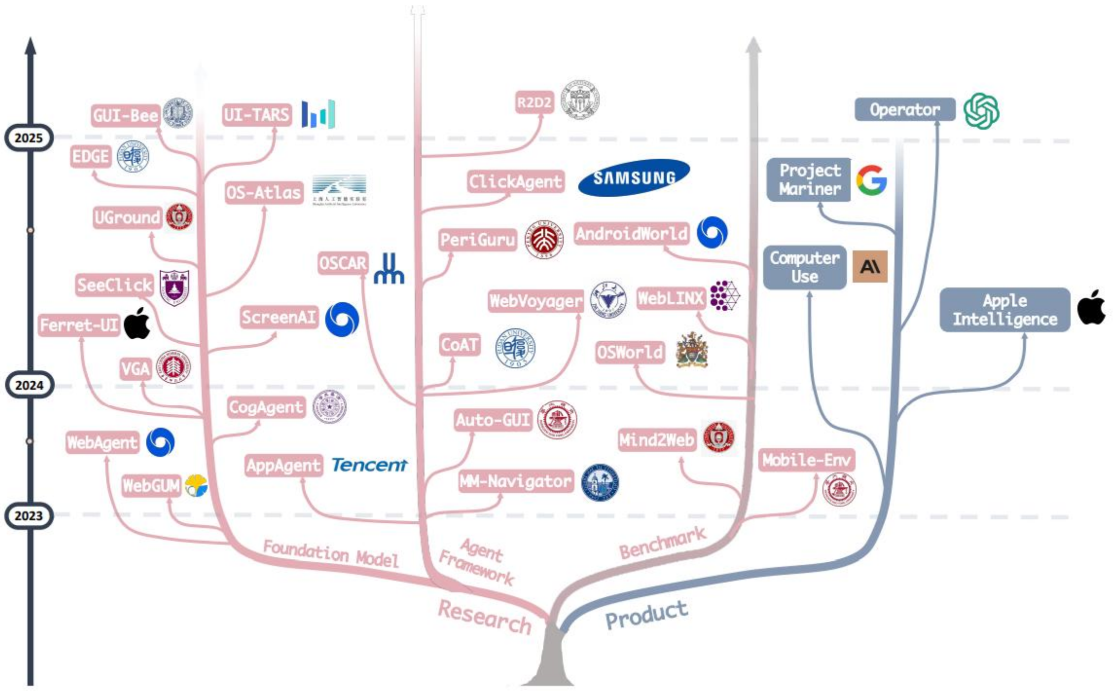


Figure 7. Academic research and commercial products of agents in recent years, illustration in paper OS Agents: A Survey on MLLM-based Agents for Computer, Phone and Browser Use

For the observation space, early agents performed best with textual representations, which worked well on structured benchmarks like MiniWoB++ because clean HTML provided consistent and concise information. However, these approaches failed to generalize to realistic websites such as Mind2Web, where noisy and inconsistent HTML caused performance to drop below 10%. In contrast, newer visual foundation models using image-based representations (like GPT-4 agents) achieved far higher success rates, as visual layouts naturally encode consistent structure through design principles. Consequently, modern high-performing agents now rely on image or multimodal screen representations, which better capture real-world web and computer environments.

Most high-performing browser use agents on benchmarks such as WebArena or Mind2Web are based on proprietary multimodal models like OpenAI's GPT or Google's Gemini, which often ultilize the Set of Mark (SoM) prompting. For example, Gemini 2.5 Computer Use (rank 1 on Online-Mind2Web) is a specialized model and not open source, limiting accessibility for research and implementation. However, on the Voyager benchmark, the top-performing agents (Magnitude and Browser Use) are open source, providing accessible codebases and architectures for practical development. Together, the open-source framework of these agents and the multimodal with SoM design trend seen in proprietary models form a strong foundation for future research and implementation in browser use agents.

For computer use applications, adopting an Agentic framework integrated with a Coding Agent currently represents the most promising direction. This approach has demonstrated strong performance on benchmarks such as OSWorld, showcasing its ability to handle diverse and challenging tasks efficiently. Moreover, it offers practical advantages in deployment, as most of these frameworks and models are open source, making them easier to implement, customize and extend for real-world experimentation and development.