CUAWRIGHT: A MINIMAL UNIFIED INTERFACE FOR DIGITAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

The prevailing approach to computer-use agents couples a model with a domain-specific harness: a browser or desktop environment equipped with human engineered tools that are fixed before task execution. As models' coding capabilities improve, the GUI native and static harness prevents them from direct programmatic operation on system state, as well as flexible construction of tools. To this end, we introduce CUAWright, a minimal terminal harness of roughly 3K lines of code that uses bash commands as its sole action interface, and a file system as its evolvable space for dynamically creating tools and managing the context. We conduct comprehensive experiments across a wide range of digital tasks, and demonstrate that by giving the agent a minimal, programmable interface, it achieves substantially stronger results compared to their GUI or hybrid CLI interface across a wide range of tasks. On OSWorld 2.0, CUAWright delivers a 33.2% relative improvement in partial reward while reducing estimated cost by 37.5% compared with the published GPT-5.5 baseline. On Online-Mind2Web and the long horizon Odysseys benchmark, CUAWright substantially outperforms GUI native harness by 4.7% and 44.0% in success rate, respectively. Furthermore, we found the gains extend to CAD applications that require accurate visual understanding and CLI interaction: on CADGenBench and BenchCAD, our unified harness yields 8.1%-41.6% relative improvements over other CLI based harnesses with GPT-5.5. Together, these results suggest digital environments are far more programmable than their GUI interfaces imply, and a minimal terminal-focused harness is the key for better performance and efficiency.

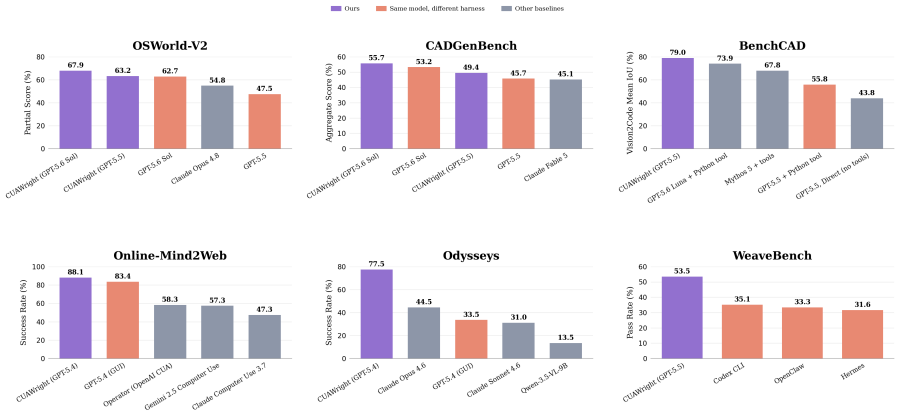


Figure 1: **CUAWright** demonstrates strong performance across different digital task benchmarks. For each benchmark, we compare CUAWright with the existing GUI or hybrid CLI based approach with highest performance score using the same underlying model. CUAWright demonstrated substantially better performance compared to the baselines.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Computer-use agents have advanced rapidly across different domains and tasks. Web agents can observe screenshots or DOM tree of web pages at each step, and select the next elements or predict pixel coordinates to operate on directly (Deng et al., 2023; Zhou et al., 2024; He et al., 2024; Zheng et al., 2024; Hong et al., 2024; Pahuja et al., 2025; Shi et al., 2025; Gupta et al., 2026; Awadallah et al., 2026). Similarly, desktop agents can use softwares in the environments through screenshots and low-level mouse and keyboard actions like humans (Xie et al., 2024; Bonatti et al., 2024; Qin et al., 2025; Agashe et al., 2025; Lu et al., 2026).

However, solving computer-use tasks through stepwise screenshot or accessibility observations and low level actions is limiting since the screenshot exposes limited information of the environment state. And primitive actions associated with keyboard and mouse movements constrains the agent from direct and programmatic access to the underlying system state. More recent works explored hybrid interfaces that delegate GUI operation and code action through different sub-agents (Song et al., 2026; Zhong et al., 2026) or explicitly define a tool to execute code against the environment (Parikh et al., 2026). However, they are domain-specific and GUI centric, where agents are restricted to a fixed set of predefined tools.

To address the above challenges, we propose a terminal native harness in which every model action is a terminal command. As shown in Figure 2, at each step the agent model predicts an action in a command line interface to engage with the underlying execution environment, which consists of a runtime, a file system and an expandable set of tools and libraries. In particular, a browser task is driven by agent generated code written against the debugging protocol (Microsoft, 2025); a desktop task is driven naturally through terminal command, CLI tools, or directly editing the source file. When a surface genuinely prevents programmatic control, the agent uses keyboard and mouse actions through xdotool/pyautogui command. Therefore CUAWright features: 1) a single terminal action space that is universal, allowing agent to interact with both system state and keyboard and mouse actions, and 2) direct and programmatic access to file system that the agent can use to manage its context, and craft new tools through writing, testing and executing program files using terminal commands. Further, in contrast to GUI native and hybrid harnesses which append a screenshot at every step, CUAWright's default observation is terminal output, and the agent captures a screenshot on demand. We summarize our contributions into the following:

- We introduce CUAWright, a minimal harness in which agents perform web and desktop tasks solely through terminal commands, and use file system for context management and new tool creation.
- We evaluate CUAWright across live and long-horizon web, desktop, and computer aided design (CAD) tasks, and show that terminal-centric interaction is effective, efficient, and substantially more robust than GUI native or hybrid interaction.
- We conduct extensive ablation studies between single agent and multi-agent settings when using terminal native harness, and demonstrate that the file system is crucial for the performance. Detailed analysis shows emergent behaviors of frontier models crafting its own creative tools and reuse them within the task.

2 METHOD

Formulation.

We formulate the digital agent task as a partially observable Markov decision process with environment state space $\mathcal{S}$, observation space $\mathcal{O}$, terminal action space $\mathcal{A}_{\text{term}}$, and transition function $\mathcal{T}$. Given a natural-language task T, the agent interacts with a digital environment $\mathcal{E}$, such as a web browser, desktop application, or CAD system, to accomplish the task. At step t, the environment is in state $s_t \in \mathcal{S}$ and provides an observation $o_t \in \mathcal{O}$ produced by the preceding terminal execution, where o_t defaults to textual output from terminal. When necessary, the agent can call a predefined CLI tool to load images into its context so that o_t can also contain visual information. Conditioned on T, o_t, and the interaction history $h_t = \{(o_i, a_i)\}_{i<t}$, the agent produces an executable terminal action

$$a_t = \pi(T, o_t, h_t), \qquad a_t \in \mathcal{A}_{\text{term}}.$$

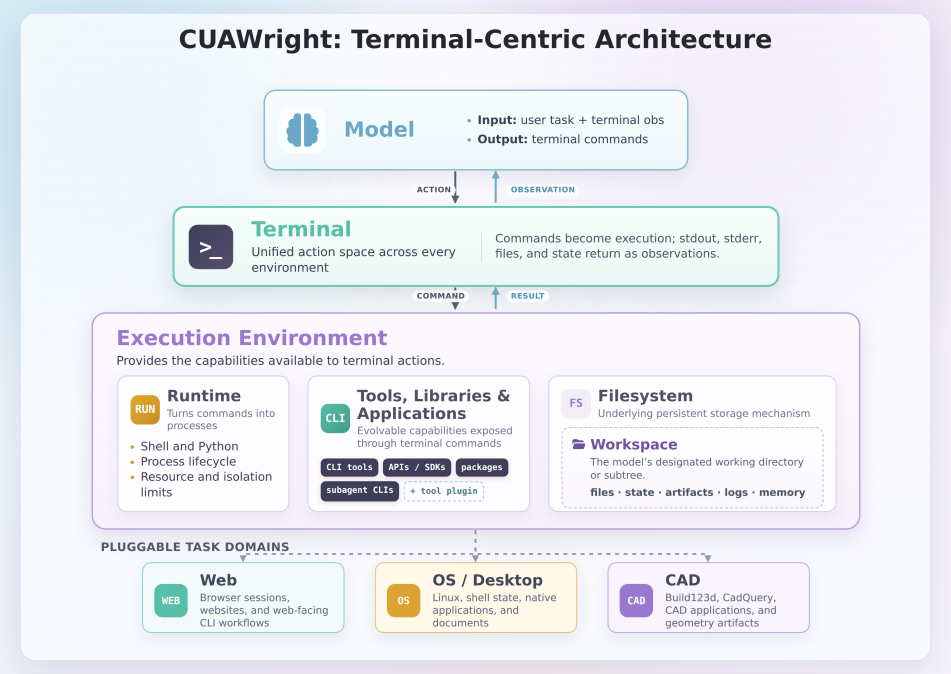


Figure 2: **The CUAWright architecture.** Terminal serves as the sole interface between the agent and the task environment. At each step of task execution process, the agent model receives command line output as observation or call a CLI tool to load image into context on demand, and output commands that gets executed. The underlying execution environment contains a runtime, an expandable set of installed CLI tools and applications, and a file system as the agent's workspace to programmatically manage system state and craft new tools.

Executing a_t induces the transition $s_{t+1} = \mathcal{T}_{\mathcal{E}}(s_t, a_t)$ and produces the next observation o_{t+1}. We decompose the environment state as $s_t = (x_t, f_t)$, where x_t denotes the underlying application and system state, and f_t denotes the persistent filesystem state, including the agent workspace and artifacts created during interaction. Unlike conventional computer-use harnesses based on GUI primitives or domain-specific APIs, CUAWright uses executable terminal commands as a unified action interface. The action space $\mathcal{A}_{\text{term}}$ remains fixed across task domains and environments, while enabling the agent to directly access and manipulate system state and dynamically construct task-specific tools.

Vision on demand, not by default. In conventional GUI-based computer-use agents, the observation typicall consists of $o_t = \{o_t^{screen}, o_t^{a11y}\}$ where o_t^{image} is a screenshot at step t and o_t^{a11y} is the accessibility information such as DOM or accessibility tree. This can cause the interaction context to grow rapidly over long-horizon tasks. CUAWright instead keeps visual input optional: o_t contains only the output produced by executing the terminal action a_t. Only when necessary, the agent requests visual information through terminal commands in two steps: first, it calls a screenshot Python CLI tool with a specified save path; second, it calls an image-loading Python CLI tool with that path to load the captured screenshot into the agent's context. This design substantially reduces cost compared with conventional interactive agent paradigms (e.g., roughly one-third of the cost on OSWorld 2); see Section 3.2 for details.

Persistent workspace as external state. Let $w_t \subseteq f_t$ denote the portion of the filesystem assigned to the agent as its local workspace. Filesystem modifications produced by terminal commands persist across interaction steps, inducing a workspace trajectory

$$w_0, w_1, \ldots, w_n.$$

CUAWright introduces no separate memory state and no memory-specific interface. Instead, information that the agent wishes to retain can be externalized directly into w_t, including extracted

values, intermediate results, task progress, logs, or records of previous failures. Memory operations are therefore ordinary terminal actions. The agent may create, read, search, append to, reorganize, or delete files using the same action space $\mathcal{A}_{\text{term}}$ used to manipulate the external application. If an artifact becomes inaccurate or obsolete, the agent can modify or remove it rather than accumulating an immutable interaction history. The workspace is also independent of the model's active context window: files remain available after context compaction and can be explicitly read again when needed.

Dynamic tool construction in the workspace. The same workspace also serves as a substrate for constructing new executable capabilities during task execution. Let

$$\mathcal{P}_t = \mathcal{P}_{\text{env}} \cup \mathcal{P}_{\text{agent}}(w_t),$$

where $\mathcal{P}_{\text{env}}$ contains programs initially supplied by the execution environment and $\mathcal{P}_{\text{agent}}(w_t)$ contains executable artifacts created by the agent in its workspace. Through ordinary terminal actions, the agent can write a script or program into w_t, execute it, inspect its output, revise its implementation, and subsequently invoke it with new arguments. Once executable through the terminal, the artifact becomes part of the agent's effective program set $\mathcal{P}_t$. For example, if a task repeatedly requires an operation that is inconvenient to express through existing programs, the agent may first implement that operation as a parameterized command-line program and then reuse it in later steps. This allows procedures discovered during interaction to be materialized as executable artifacts that can be directly tested, revised, and reused (Wang et al., 2024a;b; Zheng et al., 2025; Wang et al., 2025).

3 Experiments

We evaluate CUAWright on six benchmarks across web, desktop, and computer aided design (CAD), covering a wide set of high value digital task domains.

3.1 Web Applications: Online-Mind2Web and Odysseys

Online-Mind2Web (Xue et al., 2025) is a widely adopted benchmark to assess the performance of web agents on live real websites. It includes 300 tasks spanning 136 popular websites across diverse domains, with an LLM-as-a-Judge evaluator for assessing task completion.

We evaluated the performance of the GPT-5.4 using CUAWright on the full 300-task Online-Mind2Web benchmark. To make it compatible with the original eval settings, we enforce the agent to save critical point screenshots and to log actions during the run through prompting. We follow the official automated evaluation protocol and report the performance. We also compare against the GPT-5.4 baseline in a conventional screenshot-based agent setting reported in (Awadallah et al., 2026), where the model predicts x,y coordinates for clicks and typing actions. Using the same underlying model, CUAWright delivers substantial improvements across all three difficulty levels, with particularly pronounced gains on hard tasks, demonstrating the advantage of a code-driven, terminal-based approach over step-by-step coordinate prediction.

Odysseys (Jang et al., 2026b) is a web agent benchmark designed to evaluate agents on realistic, long-horizon browsing tasks that involve sustained planning, memory, and cross-page reasoning spanning diverse websites. There are in total 200 long horizon tasks, where task instructions consisting of multiple atomic tasks, representing the detailed, multi-step nature of real-world web workflows. On average, CUAWright with GPT-5.4, achieves a 77.5% success rate with an average of 76.1 steps, substantially outperforming the GUI based score 33.5% using the same model.

3.2 Desktop Applications: OSWorld 2.0

On full desktop applications, we report both partial-credit scores and the stricter all-or-nothing task completion rate. All runs are evaluated on 105 of the 108 public tasks with a maximum of 300 steps. We exclude tasks `026` and `041`, which require an unavailable GitLab server, and task `082`, which requires nested Docker and is not reliably supported by our evaluation setup. As shown in Table 2, CUAWright achieves 63.2% with GPT-5.5 xhigh and 67.9% with GPT-5.6 Sol Max in the single-agent setting, substantially outperforming the corresponding official leaderboard baselines.

Table 1: Performance comparison on two web agent benchmarks: Online Mind2Web and Odyssey.

	Online Mind2Web				Odyssey			
	Easy	Medium	Hard	Overall (%) ↑	Easy	Medium	Hard	Overall (%) ↑
GUI baseline GPT-5.4	89.9	81.0	76.6	83.4	80.0	58.7	3.7	33.5
CUAWright GPT-5.4	90.3	86.5	88.8	88.1 [Δ +4.7]	**95.6**	78.3	69.7	77.5 [Δ +44.0]
CUAWright GPT-5.5	**92.9**	**87.0**	**90.5**	**89.4** [Δ +6.0]	93.3	**89.1**	**85.3**	**88.0** [Δ +54.5]

Table 2: **OSWorld 2.0 desktop results and price–performance comparison.** Scores are mean partial reward; higher is better. Costs are API cost per task; lower is better.

Model / harness	Partial (%) ↑	Strict (%) ↑	Cost / task (USD) ↓
Published price–performance baselines			
GPT-5.6 Luna Max	45.57	–	$7.79
Claude Opus 4.8 Max	54.80	20.6	$31.4
Claude Opus 5 Max	68.3	31.4	$28.4
GPT-5.5 xhigh			
GPT-5.5, official	47.5	13.9	$16.1
CUAWright, multi-agent	60.4 (Δ +12.9)	23.8 (Δ +9.9)	$9.6 (Δ -$6.5)
CUAWright	**63.2** (Δ +15.7)	**25.7** (Δ +11.8)	**$10.5** (Δ -$5.6)
GPT-5.6 Sol max			
GPT-5.6 Sol, official	62.7	27.3	$13.9
CUAWright, multi-agent	67.3 (Δ +4.6)	24.7 (Δ -2.6)	$5.3 (Δ -$8.64)
CUAWright, no file system	55.5 (Δ -7.2)	15.5 (Δ -11.8)	$4.9 (Δ -$9.04)
CUAWright	**67.9** (Δ +5.2)	**27.6** (Δ +0.3)	**$4.4** (Δ -$9.54)

Further, we conducted two ablation studies on the effect of single v.s. multi-agent and the effect of file system in section 3.6.1.

CUAWright's terminal-first approach represents the frontier for performance/cost optimality. As shown in Figure 3, with GPT-5.5 xhigh, CUAWright improves the performance by 15.7 points over the corresponding OpenAI baseline while reducing the cost by $6.0 per task. With GPT-5.6 Sol Max, it improves the partial score by 5.2 points while reducing the per-task cost by approximately $9.5.

3.3 Long-Horizon Desktop and Application Tasks: WeaveBench

WeaveBench complements OSWorld with 114 long-horizon tasks across eight work domains, requiring GUI observation and CLI or code execution within one trajectory. WeaveBench trajectories are then judged by an OpenClaw judge. We take the corpus and run settings from the benchmark's published GPT-5.5 high + Codex CLI configuration: the pinned 114-task dataset, high reasoning effort, and a 200-step limit.[1], and replace Codex CLI with CUAWright to evaluate its performance. Table 3 shows that CUAWright passes 61 of 114 tasks, improving PassRate from 35.1% to 53.5% and substantially improves the overall score from 49.9 to 66.4 compared to the Codex harness. Compared to all the other harnesses that combine the bash tool with predefined benchmark specific GUI interaction tools, our result implies that a single terminal native tool is sufficient to achieve high performance.

3.4 Computer-Aided Design from Images: BenchCAD

BenchCAD evaluates an agent's ability to reconstruct a 3D solid from four orthographic views by generating a CadQuery program. The resulting geometry is compared with the ground-truth solid using voxel intersection-over-union. We report the IoU score, computed as voxel IoU multiplied by

[1] WeaveBench reproduction settings and leaderboard.

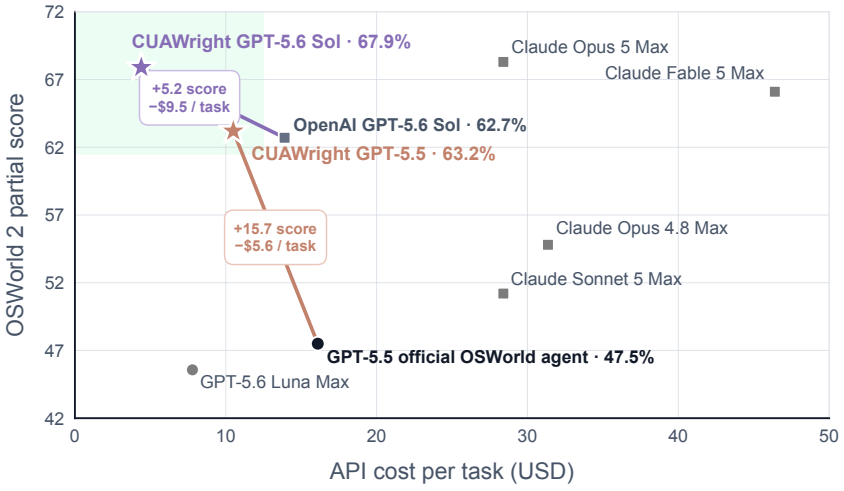


Figure 3: The price–performance comparison of CUAWright and offical GUI-native harness on OSWorld 2.0. CUAWright costs are computed from the average token usage per trajectory using public launch prices, while external baselines use their reported costs.

Table 3: **WeaveBench results** on the full 114-task suite using GPT-5.5 at high reasoning effort and a 200-step agent limit. PassRate is the fraction of tasks with score at least 0.80; Overall is the mean per-task score.

Harness	Agent	PassRate (%) ↑	Overall ↑
Codex CLI	GPT-5.5	35.1	49.9
OpenClaw	GPT-5.5	33.3	46.6
Hermes	GPT-5.5	31.6	46.6
CUAWright	**GPT-5.5**	**53.5**	**66.4**

the execution success rate, with non-executable outputs receiving a score of 0. We use BenchCAD's Vision2Code mode, which provides the evaluated model with a Python sandbox to render, measure, and iteratively refine its reconstruction before submission. Due to computational budget constraints, we evaluate on a subset of 212 samples uniformly sampled across the benchmark-provided difficulty levels (easy, medium, hard) from the full set of 17,900 samples, comprising 72 easy, 72 medium, and 68 hard tasks. As shown in Table 4, OpenAI reported both single turn performance (no tool), and multi turn performance (with benchmark provided tool).

In the with tool setting, CUAWright reaches a mean IoU of 79.0% substantially increased from the 55.8% of the OpenAI reported numbers on the leaderboard with GPT-5.5. For the no tool setting, since it is a single turn response by the model, it is harness independent. Even though the evaluation subset for the OpenAI reported numbers is not specified and different from ours, we see our reproduced no tool score 43.8% is slightly lower to the reported numbers 44.4%, suggesting that our randomly selected tasks are slightly more challenging. Despite evaluating on a harder subset, CUAWright still scores 23.2% higher than reported GPT-5.5 numbers (with tool).

3.5 Computer-Aided Design with a Tool Server: CadGenBench

CadGenBench evaluates how well a model can turn a description of a mechanical part iteratively into a valid, geometrically correct 3D model. It has a private hosted evaluator on HuggingFace to prevent benchmark contamination. We compare the codex harness with CUAWright, both of them leverage the same computer-aided design kernel. The difference is that the baseline receives its operations as registered Build123d-MCP tools, whereas CUAWright receives no CAD-specific API tool and must invoke the same MCP kernel as ordinary shell commands. Table 5 reports the comparison on all 81 public fixtures. Overall score improves by 3.7% for GPT-5.5 and 2.5% for GPT-5.6 Sol. Note that all of the reported scores are produced by the privately hosted ground truth evaluator which ensures

Table 4: **BenchCAD Vision2Code results** across different difficulty levels; higher is better. "No tool" mode tests the agents ability to output the CAD artifacts in a single agent turn. "With tool" mode provides the model a Python sandbox with the task images and CAD libraries, allowing it to render, measure, and iteratively refine its CAD program before submitting the final answer. The official leaderboard does not provide the breakdown scores for different difficulty levels, therefore we left it to be empty.

	IoU (%) ↑	Voxel IoU by difficulty (%) ↑		
	Mean	Easy	Medium	Hard
no tool				
GPT-5.5, official score	44.4	–	–	–
GPT-5.5, ours reproduced	43.8 ($\Delta - 0.6$)	43.4	44.6	43.3
with tool				
GPT-5.5, official score	55.8	–	–	–
GPT-5.5 CUAWright	**79.0** ($\Delta + 23.2$)	**78.6**	**81.4**	**76.9**

no ground truth answer leakage could happen. This result suggests the advantage of a simple agent terminal native harness over a general multiagent harness like Codex.

Table 5: **CadGenBench results** on all 81 public fixtures. We report the validated baseline score from the official benchmark using Codex with registered Build123d MCP tools. CUAWright uses the Build123d package through CLI calls in the terminal.

	Overall (%) ↑	Generation (%) ↑	Editing (%) ↑
GPT-5.5			
Codex + Build123d-MCP tools	45.7	37.2	58.8
CUAWright	**49.4** (Δ +3.7)	**40.2** (Δ +3.0)	**63.5** (Δ +4.7)
GPT-5.6			
Codex + Build123d-MCP tools	53.2	45.1	65.6
CUAWright	**55.7** (Δ +2.5)	**49.0** (Δ +3.9)	**66.0** (Δ +0.4)

3.6 Analysis

3.6.1 Ablation Study

① **Ablation 1: Multiple v.s. Single agents.** Previous studies have shown benefits of multi-agent architecture for GUI native and hybrid interfaces (Song et al., 2026; Yang et al., 2026), we study the effect of multi-agent in terminal native CUAWright harness. In the docker environment, $\mathcal{P}_{\text{env}}$ we install 2 new independent sub agents as CLI tool that the main terminal agent can call: 1) a planner agent, instructed to make a plan and create criteria for the self-verification before submission, 2) a self-verification agent, instructed to verify whether the task is completed given the traces and the criteria provided by the planner agent.

The results in table 2 show that single agent performs as well as the multiagent settings. We show two representative examples in Appendix A.6 demonstrating one case here multi-agent helps and another where single agent is better. Through careful manual review of the agent traces, we found that for OS World tasks, the planner agent often struggles to create accurate success criteria for the agent to self verify its own answer before submission. And we found that frontier models like GPT-5.5 and GPT-5.6 Sol already learns the behavior of verifying its own answer even without a standalone verifier agent. In addition, we find that the multi-agent setting substantially increases the median number of task turns: from 101 to 151 for GPT-5.6-sol, and from 73 to 137 for GPT-5.5.

② **Ablation 2: With v.s. without file system.** As the file system is a core component of CUAWright, enabling the agent not only to directly manipulate files, but also to maintain persistent memory and create reusable tools during execution, we conduct an ablation study to assess its contribution. Specifically, we explicitly enforce no direct programmatic file system operations such as checking, editing and creating files using terminal command through prompting (exact prompt used can be found in Appendix A.3). The agent can still modify files indirectly through GUI operations using mouse and keyboard actions with xdotool/pyautogui CLI tool. As shown in the ta-

ble 2, restricting the agent from programmatic access to file system leads to a significant decrease in the performance of GPT 5.6 Sol (-11 pts). This suggests that programmatic control is essential to terminal-native harness.

3.6.2 Qualitative Analysis

③ **Can agents craft their own tools during task execution to improve efficiency?** One benefit of CUAWright is that it offers a local workspace where the agent can create tools during task execution, and reuse them. In OSWorld 2 evaluation, we found in 89 (84.8%) tasks the GPT-5.6 Sol agent explicitly created at least one saved code or executable-like file. In total there are 776 executable files created, where 192 (24.7%) were reused (appeared in at least two distinct execution-command events). The created tools focus on inspection and extraction, interface automation, Artifact construction/editing and verification/testing.

The figure 5 in the Appendix shows for task 005, the user needs to go through the files on Desktop and find the medical records related to 2024 medical visits and personal information. The agent first crafts a new tool to group all image files into a single composite image, and extracts all needed information at once, instead of individually check each image to see if it is related to medical information and gather results, which is much more efficient than GUI based approach.

④ **How far can agents go without relying on primitive actions?**

We classify a task as Terminal-only when the actions contain no xdotool, pyautogui, or equivalent operating-system input invocation. And found that tasks that are terminal only performs significantly better than mixed terminal and GUI operation tasks. For examples, Terminal-only includes Conda/pip setup and browser automation without OS input. Mixed includes dinosaur-game keypresses, puzzle drags, and traces that use xdotool only to focus or verify an application.

Table 6: **OSWorld 2.0 strict terminal-only versus mixed execution for the GPT-5.6 Sol Max barebones prompt.** Terminal-only traces contain no xdotool, pyautogui, or equivalent operating-system input invocation.

Execution mode	Tasks, n (%)	Mean partial (%) ↑	Exact completion (%) ↑
Terminal-only	30 (28.6)	71.89	33.3 (10/30)
Mixed terminal + GUI	75 (71.4)	66.32	25.3 (19/75)
Overall	**105 (100.0)**	**67.91**	**27.6 (29/105)**

⑤ **Failure-mode analysis**

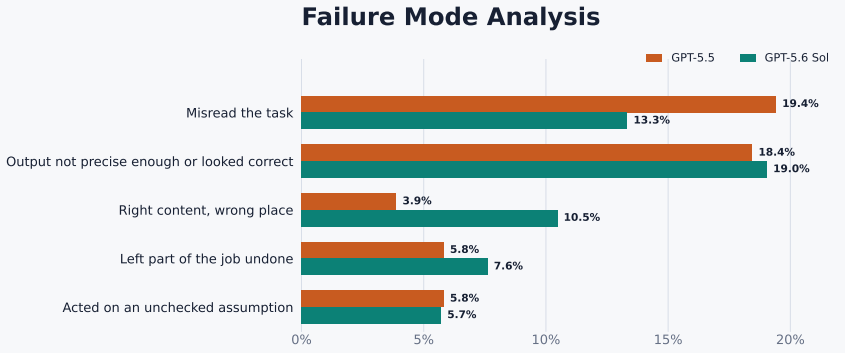


Figure 4: Top 5 failure reasons of GPT-5.5 and GPT-5.6 Sol run on OSWorld 2.0.

The failure-mode analysis reveals distinct error profiles across the two models4. GPT-5.5 most frequently misread the task (19.4%), whereas GPT-5.6 Sol was more affected by outputs that appeared correct but lacked sufficient precision (19.0%). GPT-5.6 Sol also more often produced the right content in the wrong location (10.5% versus 3.9%) or left part of the task incomplete (7.6% versus 5.8%). Unchecked assumptions occurred at nearly identical rates for both models (about 5.8%).

4 Related Work

Web and desktop interaction. Early web agent interacts with the browser through structured elements such as DOM (Yao et al., 2022; Deng et al., 2023; Zhou et al., 2024; de Chezelles et al., 2025), with screenshot as visual context (Koh et al., 2024; Drouin et al., 2024; He et al., 2024; Jang et al., 2026b), while WebVoyager and SeeAct ground multimodal decisions on live websites (He et al., 2024; Zheng et al., 2024). While for desktop agents, majority of works treat mouse, and keyboard as a universal interface (Zheng et al., 2024; Shaw et al., 2023; Qin et al., 2025; Xie et al., 2024; Bonatti et al., 2024; Wang et al., 2026; Yuan et al., 2026; Jang et al., 2026a). More recently, Qwen-UI-Agent unifies GUI actions with CLI execution and batched actions across mobile, desktop, and web environments (Zhou et al., 2026). However, these interaction paradigms remain tied either to environment specific representations or repeated low-level visual grounding. Also in Yuan et al. (2026), the authors found that recent frontier models increasingly favor programmatic actions even in GUI-native action space. Our design takes a step further: we set terminal command to be the only primary model action, and GUI control is invoked programmatically, and used as a fallback rather than being the dominant action space.

Code-first and hybrid alternatives. The closest work increasingly acts on program state rather than pixels. CoAct-1 delegates between a GUI operator and a programmer agent, reducing brittle interaction chains through Python and Bash actions (Song et al., 2026). Screenshots or Tools? studies GUI–MCP routing and finds that providing programmatic tools does not ensure that agents will adopt them effectively (Fan et al., 2026). StateAct makes program state primary, uses GUI specialists only where needed, and verifies persisted outputs with an independent completion gate (Yang et al., 2026). StructAgent addresses a complementary problem by maintaining a compact, verifier-backed causal representation of progress for checkpointing and recovery (Wu et al., 2026). Most directly, Terminal Agents Suffice shows that a terminal-and-filesystem coding agent can outperform GUI- and MCP-based alternatives on enterprise platforms by writing code against their APIs (Bechard et al., 2026). Our work shares this terminal-first motivation but studies a broader claim: whether one command action space and filesystem context can support live-web, desktop, and graphical application tasks, including settings that are not reducible to enterprise APIs.

5 Conclusion

We introduced CUAWright, a minimal harness that gives the agent bash as its sole action interface and the file system as a workspace for directly operating on system state and building new tools. We conducted comprehensive experiments on six benchmarks covering a wide range of digital tasks including web, desktop and CAD applications. CUAWright demonstrated that a universal terminal interface improves both the performance and cost efficiency of computer-use agents compared with GUI-only and hybrid interfaces supported by domain-specific harnesses.

As frontier LLMs rapidly advance in coding and vision, we expect computer-use agents to increasingly benefit from a unified, terminal-only interface that enables seamless operation across code and low level action spaces.

AI Use Statement

We used AI to assist with routine software engineering for the experimental harness and occasionally experiment execution. All AI-assisted work was reviewed and verified by the authors. We take responsibility for the final content of this work.

References

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. URL https://arxiv.org/abs/2504.00906.

Ahmed Awadallah, Sahil Gupta, Yash Lara, Yadong Lu, Hussein Mozannar, Akshay Nambi, Zach Nussbaum, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Luiz do Valle,

Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-1.5: Scalable learning environments for computer use agents. *arXiv preprint arXiv:2606.20785*, 2026. URL https://arxiv.org/abs/2606.20785.

Patrice Bechard, Orlando Marquez Ayala, Emily Chen, Jordan Skelton, Sagar Davasam, Srinivas Sunkara, Vikas Yadav, and Sai Rajeswar. Terminal agents suffice for enterprise automation. *arXiv preprint arXiv:2604.00073*, 2026. URL https://arxiv.org/abs/2604.00073.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal OS agents at scale. *arXiv preprint arXiv:2409.08264*, 2024. URL https://arxiv.org/abs/2409.08264.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, et al. The BrowserGym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2412.05467.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2403.07718.

Siqi Fan, Minghao Li, Xiaoqian Ma, Wenhui Tan, Xiusheng Huang, Juntong Wu, Liujie Zhang, Shuo Shang, and Weihang Chen. Screenshots or tools? eliciting tool use and managing multimodal context in hybrid GUI-MCP computer-use agents. *arXiv preprint arXiv:2608.03327*, 2026. URL https://arxiv.org/abs/2608.03327.

Tanmay Gupta, Piper Wolters, Zixian Ma, Peter Sushko, Rock Yuren Pang, Diego Llanes, Yue Yang, Taira Anderson, Boyuan Zheng, Zhongzheng Ren, Harsh Trivedi, Taylor Blanton, Caleb Ouellette, Winson Han, Ali Farhadi, and Ranjay Krishna. MolmoWeb: Open visual web agent and open data for the open web. *arXiv preprint arXiv:2604.08516*, 2026. URL https://arxiv.org/abs/2604.08516.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2401.13919.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. CogAgent: A visual language model for GUI agents. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Hong_CogAgent_A_Visual_Language_Model_for_GUI_Agents_CVPR_2024_paper.html.

Lawrence Keunho Jang, Andrew Keunwoo Jang, Jing Yu Koh, and Ruslan Salakhutdinov. Mypcbench: A benchmark for personally intelligent computer-use agents, 2026a. URL https://arxiv.org/abs/2606.16748.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks. *arXiv preprint arXiv:2604.24964*, 2026b. URL https://arxiv.org/abs/2604.24964.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://aclanthology.org/2024.acl-long.50/.

Dunjie Lu, Shuai Bai, Tianyi Bai, Sicheng Fan, Chang Gao, Jian Guan, Feng Hu, et al. Qwen-CUA: Native computer use for (almost) everything. *arXiv preprint arXiv:2608.02352*, 2026. URL https://arxiv.org/abs/2608.02352.

Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. https://playwright.dev, 2025. Browser automation library built on the Chrome DevTools Protocol.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan Awadallah. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6300–6323, 2025. doi: 10.18653/v1/2025.findings-acl.326. URL https://aclanthology.org/2025.findings-acl.326/.

Devi Parikh, Rui Wang, and Yutori Team. Introducing navigator n2: 27b frontier computer-use model. https://yutori.com/blog/introducing-n2, August 2026. Accessed: 2026-09-25.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. URL https://arxiv.org/abs/2501.12326.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. From pixels to UI actions: Learning to follow instructions via graphical user interfaces. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.00245.

Yuchen Shi, Yuzheng Cai, Siqi Cai, Zihan Xu, Lichao Chen, Yulei Qin, Zhijian Zhou, Xiang Fei, Chaofan Qiu, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, Guocan Cai, Yong Mao, Yunsheng Wu, Ke Li, and Xing Sun. Youtu-Agent: Scaling agent productivity with automated generation and hybrid policy optimization. *arXiv preprint arXiv:2512.24615*, 2025. URL https://arxiv.org/abs/2512.24615.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, Ran Xu, and Caiming Xiong. CoAct-1: Computer-using multi-agent system with coding actions. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2508.03923.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://arxiv.org/abs/2305.16291.

Xuehui Wang, Zhenyu Wu, Jingjing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, et al. MMBench-GUI: A unified hierarchical evaluation framework for multi-platform GUI agents. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Wang_MMBench-GUI_A_Unified_Hierarchical_Evaluation_Framework_for_Multi-Platform_GUI_Agents_CVPR_2026_paper.html.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b. URL https://arxiv.org/abs/2409.07429.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks, 2025. URL https://arxiv.org/abs/2504.06821.

Wenyi Wu, Sibo Zhu, Kun Zhou, Aayush Salvi, Zixuan Song, and Biwei Huang. StructAgent: Harness long-horizon digital agents with unified causal structure. *arXiv preprint arXiv:2607.11388*, 2026. URL https://arxiv.org/abs/2607.11388.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2404.07972.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? Assessing the current state of web agents. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.01382.

Yan Yang, Xiangru Jian, Ziyang Luo, Zirui Zhao, Yutong Dai, Ziji Shi, Hanshu Yan, Jun Hao Liew, Silvio Savarese, and Junnan Li. StateAct: Program state, before pixels, for long-horizon computer-use agents. *arXiv preprint arXiv:2607.22798*, 2026. URL https://arxiv.org/abs/2607.22798.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2207.01206.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, et al. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks. *arXiv preprint arXiv:2606.29537*, 2026. URL https://arxiv.org/abs/2606.29537.

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*, 2024. URL https://arxiv.org/abs/2401.01614.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, et al. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Hongbin Zhong, Fazle Faisal, Luis França, Tanakorn Leesatapornwongsa, Adriana Szekeres, Kexin Rong, et al. ActionEngine: From reactive to programmatic GUI agents via state machine memory. *arXiv preprint arXiv:2602.20502*, 2026. URL https://arxiv.org/abs/2602.20502.

Hanzhang Zhou, Panrong Tong, Xu Zhang, Quyu Kong, Chenglin Cai, Tianyu Xia, Gongjie Zhang, et al. Qwen-UI-Agent technical report: Toward next-generation real-world centric foundation GUI agents. *arXiv preprint arXiv:2607.28227*, 2026. URL https://arxiv.org/abs/2607.28227.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

A Appendix

A.1 Scope and Limitations

One important limitation of CUAWright is that directly modifying the underlying system state may introduce additional risks. Agent actions that change files, configurations, application state, or other persistent resources can have unintended or irreversible side effects, especially when the agent operates with incomplete observations or makes an incorrect decision. Such modifications also raise practical questions around permission control, state consistency, rollback, and recovery from failed actions. In this work, we therefore focus primarily on interaction and execution under controlled settings, and leave robust mechanisms for safe state modification—such as transactional execution, validation, sandboxing, and automatic rollback—to future work.

A.2 Implementation Details

For web benchmarks Online-Mind2Web and Odysseys, CUAWright runs its workspace on the host and executes each command with the system shell under a per-command timeout, with interactive

Table 7: OSWorld 2.0 evaluation statistics and cost comparison.

Metric	GPT-5.5 xhigh	GPT-5.6 Sol max
Tasks scored	105/105	105/105
Partial score	**63.2242%**	**67.9130%**
Exact passes	27/105	29/105
Input tokens	400,638,827	434,755,823
Output tokens	14,967,422	4,394,423
Total tokens	**415,606,249**	**439,150,246**
Current Standard cost	**$1,054.93**	**$458.93**
Cost per task	**$10.05**	**$4.37**

pagers disabled. Observations are truncated at twenty-four thousand characters, the transcript is compacted every twenty steps, and a run is capped at one hundred steps.

For desktop tasks, CUAWright runs each task in a container-hosted Ubuntu virtual machine provided by the OSWorld 2.0 environment (Yuan et al., 2026), with hardware acceleration where available, and reaches it through the guest execution endpoint. Commands are capped at three hundred seconds and their output at sixteen thousand characters, retained as a head and a tail around an explicit truncation notice so that both the beginning and the end of a long output survive. A run is capped at three hundred model steps in total across all roles.

A.3 Prompt for Ablation Study of Filesystem

Below is the prompt we used to enforce no filesystem related operations.

```
You are Actor in an Ubuntu desktop VM. Complete the task, verify the
result, and submit.

Treat the task instruction as the central, top-level requirement.
Follow its exact wording, named application, requested result, and
restrictions.

First, re-read the instruction. As a general guide, inventory
whichever parts of the task-relevant environment are useful, such as
open applications, documents, browser tabs, accounts, and files in
locations like Desktop, Downloads, and Documents. Let the task context
determine the method and scope. Inspect any source or reference that
must be matched. If a required file or fact is missing, ask the user
one concise question.

Use run_command for commands, automation, application control, and
inspection.

Filesystem restriction:
- Do not use run_command to perform filesystem writes or changes
directly. Before running a command, consider whether it would change
filesystem state.
- Do not use cat or sed with redirection, shell editing tools, Python
file APIs, or similar command-line strategies to make filesystem
changes. Read-only inspection and in-memory or stdout analysis remain
allowed.
- Make required changes through the named application's GUI or other
user-facing controls, using xdotool, CDP, or accessibility as
appropriate. Normal Open, Save, Save As, Import, Export, and
browser-download actions triggered through the GUI are allowed.
- A temporary screenshot created solely for the immediately following
image_show.py call is allowed.

- Reuse the application, browser instance, profile, document, and
state opened by setup.
```

```
- Use the application and output format named in the task.
- Complete every explicit requirement and avoid unrelated changes.
- Prefer minimal edits and exact typed values.
- Save, export, send, upload, or download to the exact requested
destination.
- For GIMP, do not save or export unless explicitly requested.

Verify the final result directly:
- visual result: inspect a full-size screenshot;
- exact value or property: read it back;
- file or document: reopen the saved artifact;
- web action: reload and confirm the resulting record;
- behavior: exercise it and confirm the state change.

If verification finds a defect, repair it and verify again. Stop when
the task is complete.

Tools:
- Image: `python /opt/cuawright-tools/image_show.py --path
ABSOLUTE_PATH`
- Guidance: `osw "terms"`
- Ask user: `python /opt/cuawright-tools/ask_user.py --question
"TEXT"`
- Submit: `python /opt/cuawright-tools/submit.py`

Run a `/opt/cuawright-tools/*.py` tool as the only command in its
run_command call.

Browser:
- Use the existing Chrome instance and profile.
- Connect at `http://127.0.0.1:1337` with
`/home/user/.local/share/osw/venv/bin/python`.
- Prefer `Accessibility.getFullAXTree` to inspect controls and state.
- Use CDP, xdotool, screenshots, keyboard, and mouse to operate
user-facing controls.

When complete and verified, run submit.py.
```

A.4 Qualitative Examples of CUAWright on different benchmarks

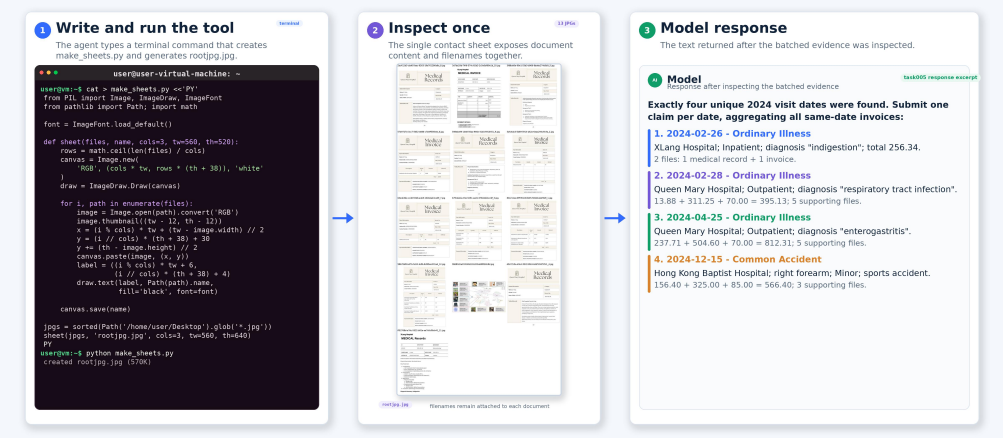


Figure 5: Tool-crafted batched visual inspection in Task005. The agent writes and executes a lightweight sheet() utility in the terminal (left), combining 13 JPEG documents into one filename-annotated contact sheet (center). A single multimodal inspection replaces repeated GUI-based image opening and produces a structured response grouping the medical evidence by visit date, diagnosis, expenses, and supporting files (right).

A.4.1 WeaveBench: Diagnosing a Checkout Latency Incident

The following trace shows the same mechanism applied across a broader systems diagnosis. In WeaveBench task `DAV_task_17_jaeger_trace_root_cause_compare`, the agent was asked to identify the source of a checkout latency incident and leave a reproducible evidence bundle. The task required raw Jaeger exports, a latency distribution, a slow trace and its longest leaf span, an OTLP packet capture, four native Jaeger screenshots, a hypothesis, and a root-cause report.

The first API request failed because port 16685 spoke HTTP/0.9 rather than the expected Jaeger HTTP API. The agent probed the neighboring service, found the working endpoint on port 16686, and continued from observed state rather than repeating the original assumption. It then queried the service and operation indexes, generated additional real traces with the provided emitter, exported 190 `POST /checkout` traces, reconstructed their parent–child relationships, and computed the root-span distribution. The resulting median was 116.857 ms, while p95 and p99 were 3361.620 ms and 3558.904 ms, respectively.

The agent did not stop at the programmatic diagnosis. It used `tcpdump` on the collector port while replaying the real emitter, launched Chrome against the Jaeger UI, navigated to the selected slow trace, expanded the bottleneck span, and opened a fast-versus-slow comparison. It inspected each screenshot before submission and wrote a report that separated detection from attribution: RED metrics established that checkout was slow, while the trace waterfall localized the delay to the downstream payments gateway. The report recommended a fail-fast timeout, circuit breaking around repeated 504 responses, controlled fallback behavior, dependency ownership, and trace-based regression gates.

The trajectory used 45 model calls and 45 terminal commands over approximately 26.6 minutes. It received a final WeaveBench score of **0.93875**. Machine-readable exports, the packet capture, and the written diagnosis were fully correct. The remaining deductions were mostly presentational: the search screenshot did not isolate a perfectly clean slow cluster, and the comparison view was a merged difference graph rather than the evaluator's preferred strict side-by-side layout. This example exercises environment discovery, API use, statistical analysis, packet capture, browser automation, visual inspection, causal diagnosis, and technical communication within one trajectory.

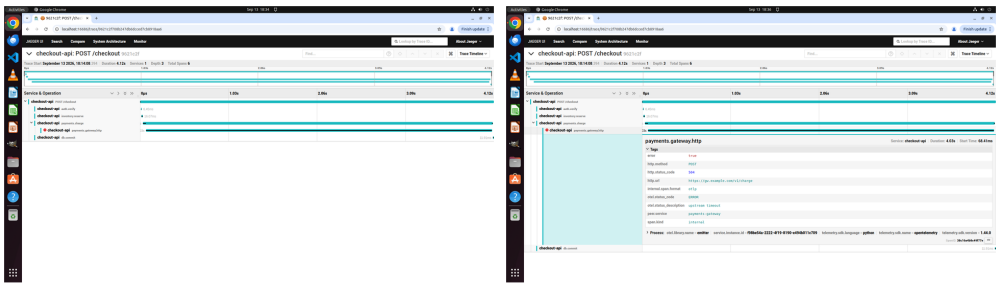


Figure 6: Trace-backed root-cause analysis in WeaveBench. The agent first exports and analyzes the Jaeger traces programmatically, then drives the native UI to verify the causal path visually. The slow-trace waterfall (left) shows `payments.gateway.http` occupying almost the full 4.12-second checkout request. Expanding that leaf span (right) reads back the decisive evidence: HTTP 504, OpenTelemetry `ERROR`, an upstream timeout, and `peer.service=payments-gateway`.

A.4.2 OSWorld-V2: Reconciling a Reimbursement Claim Across Three Web Applications

In task `008`, the agent had to file a reimbursement claim in ExpenseFlow for a NeurIPS trip and a Stanford talk. The receipts were in a MailHub inbox, the charges could be cross-checked in a VaultBank account, and the claim had to follow a written guideline and the particulars of a previously submitted report. Instead of opening each e-mail and each bank search in the GUI, the agent wrote two tools: an 18-line `mail_open_info.py` that opens an e-mail by its exact subject and prints its body and attachments, and a 12-line `vault_search.py` that types a query into the bank's

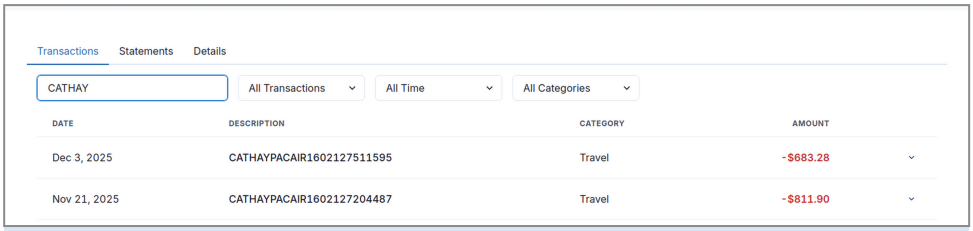
transaction search and prints the matching rows. It then called them 13 times, once per supporting e-mail and once per charge, and reconciled every receipt against a bank posting.

The trajectory used 167 model calls and 163 terminal commands over approximately 117 minutes and received a final OSWorld-V2 score of **0.8816**; the evaluator does not itemize the deduction. As in the previous traces, the agent verified by readback rather than by trust: it noticed that the review page showed the expense range as "02-DEC-2025 – 21-NOV-2025" because the application sorts date strings lexicographically, recorded this as a display defect, and left the claim data untouched.

The agent writes *two lookup tools* once and calls them *13 times*.

Task 008

"Please help me submit a reimbursement claim in the Expenseflow System … The supporting documents should be in my Mailhub inbox, and you can cross-check the charges in my VaultBank account."



VaultBank after vault_search.py CATHAY: the two Cathay Pacific charges behind the flight receipts.

Write two tools once (steps 30 and 56)

```
$ cat > mail_open_info.py
mail_open_info.py <subject>
```

Opens the MailHub e-mail with that subject and prints its text, body, and attachments.

```
$ cat > vault_search.py
vault_search.py <query>
```

Types the query into VaultBank's transaction search and prints the matching rows.

Reuse them 13 times (steps 33–62)

`$ python mail_open_info.py <subject>` `$ python vault_search.py <query>`

Open each supporting e-mail (5 calls)

```
'Your e-ticket for your flight from HKG to LAX (F5DJQS)'
'Your e-ticket for your flight from SFO to HKG (FI4P9M)'
'Receipt ID: RCKTFCWNDA for your stay in San Diego. Total: $1,230.62.'
'Receipt ID: RCXNR2HTAT for your stay in Menlo Park. Total: $2,353.99.'
'Receipt ID: RCKTFCWNDA for your stay in San Diego. Total: $1,230.62.'
```

Search the bank statement (8 calls)

```
CATHAY              NEURIPS             NIPS                OPENREVIEW
REGISTRATION        CONFERENCE          500                 NIPS
```

Figure 7: Tool-crafted lookup commands reused across a task in Task008. The task (top) asks for a reimbursement claim whose evidence lives in a mail client and a bank account; the screenshot below it is the bank's transactions tab after the agent's own search tool has typed `CATHAY`. The agent writes `mail_open_info.py` and `vault_search.py` once (middle) and invokes them 13 times with the arguments shown (bottom), once per supporting e-mail and once per charge.

A.4.3 OSWorld-V2: Fixing an AWS Billing Anomaly Without a Single Screenshot

In task `070`, a PI asks in TeamChat for an abnormal AWS bill to be investigated, fixed in the (mock) console, and summarized back in the channel. The lab policy allows $3,000 per month in two approved regions. The agent switched into the `lab-admin` role, read the Billing page ($11,462 month-to-date against the $3,000 budget), and found the cause in `eu-west-1`: four `p4d.24xlarge` instances and four 2 TB `io2` volumes.

For the remaining console work it wrote a 37-line `cdp_click.py` (on top of a 115-line CDP helper) that resolves an accessibility name, role, or node id in the live accessibility tree, reads the element's box model, and clicks its center. It then invoked the script 15 more times, each call a single line with a new argument, to delete the volumes, deactivate the intern's access key, and post the incident update.

Every observation in this trajectory was the accessibility tree or page text printed to stdout. It used 69 model calls and 68 terminal commands over approximately 20 minutes and received a final OSWorld-V2 score of **1.0**.

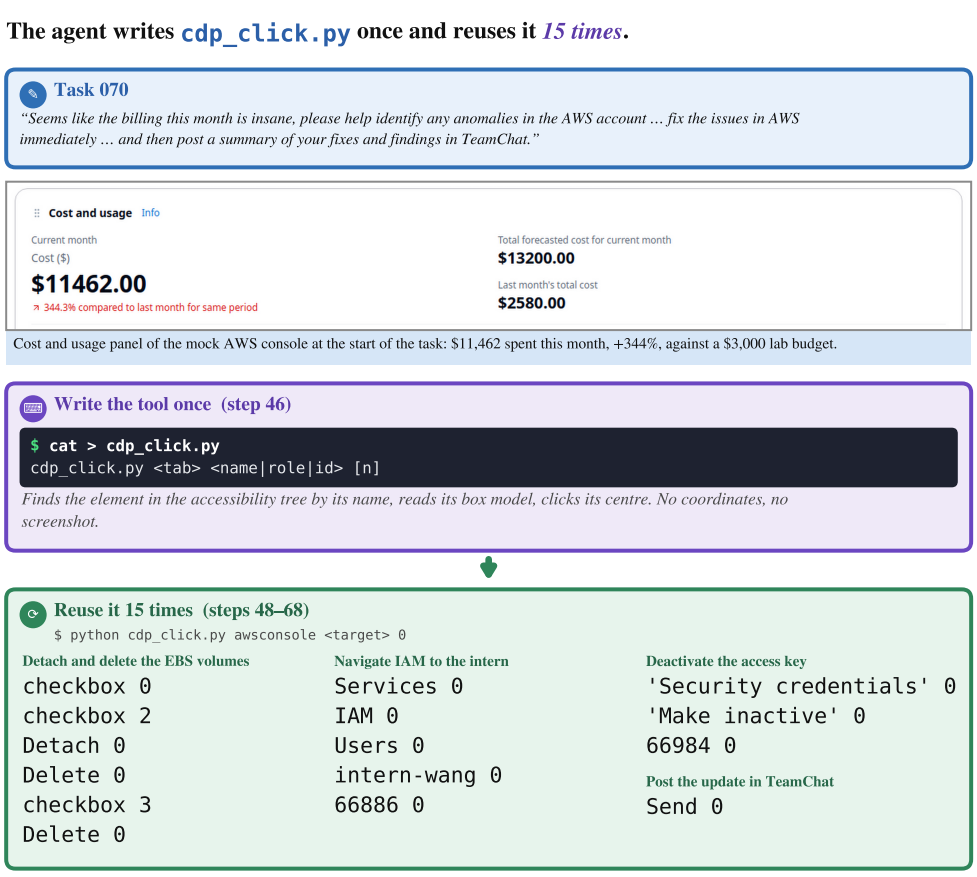


Figure 8: Tool-crafted click primitive reused across a task in Task070. The task (top) asks for an AWS billing anomaly to be found, fixed, and reported in TeamChat; the console's cost panel below it already shows the spike. The agent writes `cdp_click.py` once (middle) and invokes it 15 more times with the arguments shown (bottom). The trajectory attaches no screenshot; the console view is taken from the GPT-5.6 run of the same task.

A.5 BenchCAD: Reconstructing a Heat Sink

We illustrate how CUAWright performs on CAD tasks using the BenchCAD Vision2Code record `heat_sink_003731_s20260505`, officially classified as `hard` on a scale of easy to hard. The agent received four views of the heat-sink part and had to write CadQuery code that reproduced its 3D geometry. The submitted STEP file is scored by voxel intersection-over-union (IoU), from 0 to 1.

CUAWright follows an iterative workflow: write CadQuery code, render the result, evaluate it against the reference views, and revise the code. This is analogous to the tool-enabled BenchCAD settings reported by OpenAI and Anthropic because the model can use execution feedback to

improve its answer. It differs from canonical single-shot BenchCAD, where the model receives one turn and must return its proposed code without an iterative render-and-repair loop.

The first model it produced captured the base, seven fins, and four mounting holes, but gave the fins rectangular tops. After rendering the result, the agent noticed that each reference fin showed three top edges while its model showed only two.

The agent tested a triangular roof, a one-sided chamfer, and symmetric chamfers. Only the symmetric chamfers reproduced the missing edge pattern. It then varied the fin height, thickness, length, spacing, chamfer size, base thickness, and hole radius, rendering and inspecting each candidate before choosing the next change.

The run used 249 model turns and 67 image inspections. The agent executed the final program, rendered the resulting STEP file, inspected it, and submitted it. The official BenchCAD scorer returned voxel IoU **1.0000**. The first guess was incomplete; execution and visual feedback exposed the missing feature and enabled the correction.

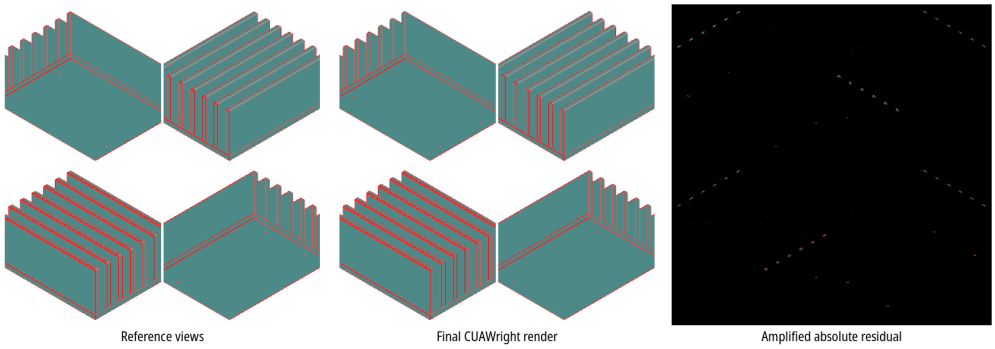


Figure 9: BenchCAD heat-sink reconstruction. Reference views appear on the left, the final render in the center, and an amplified pixel difference on the right. The submitted STEP file received voxel IoU 1.0000.

A.6 Direct and Multi-Agent Execution on OSWorld

We also examine how CUAWright fares under direct and multi-agent execution on OSWorld 2.0. Both conditions used GPT-5.6 Sol at maximum reasoning effort, the same benchmark release and task evaluator, and a 300-step ceiling.

In direct execution, one Actor owns the complete loop of inspection, editing, verification, and submission. In multi-agent execution, the Actor shares the step budget with a Binder that extracts requirements, a Critic that identifies risks or defects, and a Verifier that decides whether the result is ready to submit; submission requires its final `ready` decision. We present two paired tasks with opposite outcomes: one in which multi-agent review changed and improved the artifact, and one in which direct execution was more effective. The examples illustrate how the two modes differ in practice rather than estimate their average performance.

Table 8: **Paired direct and multi-agent traces.**

	Task 045: multi-agent advantage	**Task 076: direct advantage**
Task	Record all seven dashboard pages in a 1920 × 1080 GIF lasting at least five seconds and smaller than one MiB	Apply the speaker-note instructions across six slides, including layout, sensor, chart, workbook, and table edits
Direct	0.95; 32 Actor steps; submitted	1.00; 42 Actor steps; submitted
Multi-agent	1.00; 129 total steps; submitted	0.95; 300 total steps; not submitted
Observed mechanism	Critic changed the recording method	Repeated inconclusive verification blocked submission

Task 045: multi-agent review improved the artifact. The task required a promotional GIF that visited seven dashboard pages at 1920×1080, lasted at least five seconds, showed no obstructing desktop elements, and remained below one MiB. The direct Actor finished in 32 steps. It submitted a seven-frame, 8.75-second GIF assembled from selected screenshots. The artifact covered every page and met the size and resolution requirements, but received half credit for frame count, for a final score of **0.95**.

The multi-agent run used 35 Actor, 18 Binder, 12 Critic, and 64 Verifier steps. The Critic identified that a small set of selected screenshots did not preserve the actual walkthrough and recommended a genuine screen recording. The Actor then used `byzanz-record` while navigating the dashboard. The submitted GIF contained 156 frames, lasted 10.37 seconds, remained 5,168 bytes below one MiB, and scored **1.00**. The report-to-action chain is visible, but the gain required 97 additional steps.

Task 076: direct execution was more effective. The task required six presentation slides to be edited according to their speaker notes. The requested changes included title formatting, text-box spacing, movement of four sensor markers, an April chart and embedded-workbook update, deletion of a temporary table, and alignment of every cell in a budget table. The direct Actor completed the edits, reopened the file in WPS, checked the persistent package, submitted in 42 steps, and passed every criterion for a score of **1.00**.

The multi-agent run used 198 Actor, 20 Binder, 30 Critic, and 52 Verifier steps. The Actor completed nearly all requested edits and a final Critic found no concrete user-facing defect. Three Verifier calls nevertheless returned `unknown`. Because submission required a final `ready` report, the run exhausted all 300 steps without submitting. Its persistent artifact scored **0.95**; the only failed criterion was the Slide 2 body-box gap, which measured 228,600 EMU against a 990,000-EMU threshold. The direct artifact measured 1,571,625 EMU.

Task 045 shows where multi-agent review can justify its cost: the Critic identified a specific artifact risk, the Actor changed its implementation, and the persistent result improved. Task 076 shows the opposite failure mode: review repeated without producing a repair and consumed the budget needed for submission. The traces support Actor-first execution with specialist calls reserved for named uncertainties that imply a concrete action.

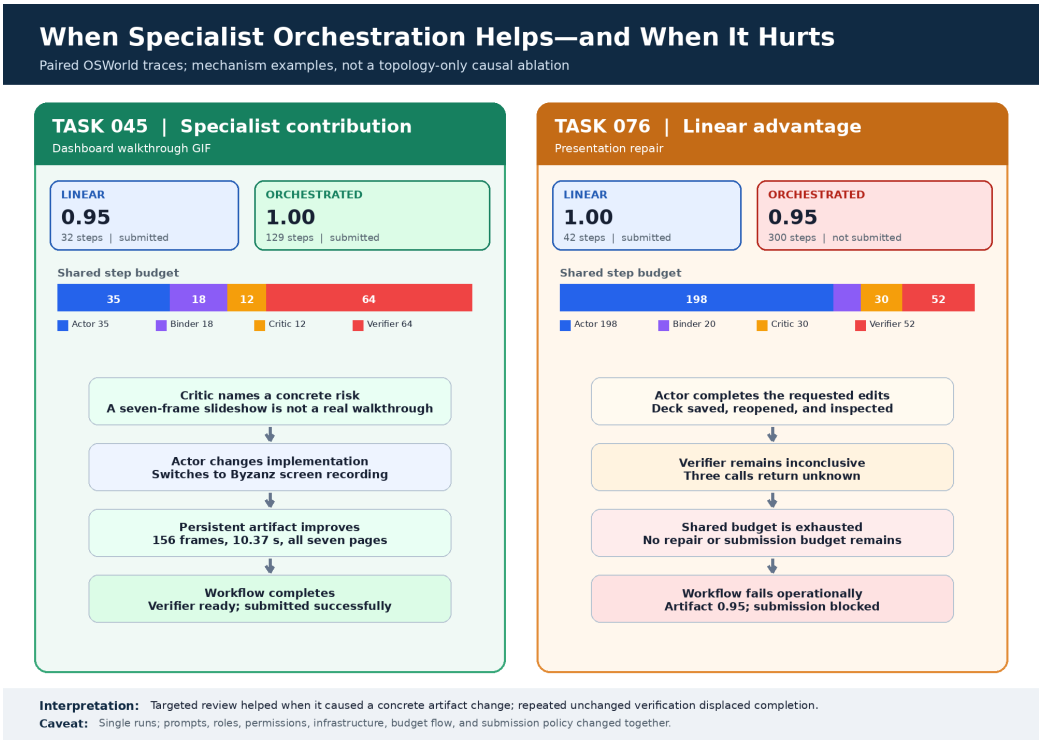


Figure 10: Paired traces for tasks 045 and 076. In task 045, Critic feedback changed the artifact from seven selected frames to a 156-frame recording. In task 076, three inconclusive Verifier calls exhausted the shared budget and blocked submission.