StateAct: Program State, before Pixels, for Long-Horizon Computer-Use Agents

Yan Yang[†] Xiangru Jian[†] Ziyang Luo[†] Zirui Zhao Yutong Dai
Ziji Shi Hanshu Yan Jun Hao Liew Silvio Savarese Junnan Li

Salesforce AI Research

Abstract

Computer-use agents are usually improved by strengthening perception: better models for reading a screenshot and choosing where to click. Yet a screenshot is only a lossy rendering of the underlying *program state*, *e.g.*, the files, application backends, and DOM that hold the task data. Different states can produce the same pixels, while code can inspect and modify that state directly. StateAct is a code-first, multi-agent harness built around this distinction. Its main agent works directly with program state by using code, while a dedicated GUI subagent handles screenshot-and-click interaction on the few subgoals that need it, just 28 of 108 tasks and 1.1% of main-agent steps. The same direct access to program state also supports verification: an independent finish gate double-checks the saved result for structural failures, *e.g.,* output that is missing, unsaved, or written to the wrong path. To stay on track over hundreds of steps, the main agent hands subgoals to fresh subagents, keeping its own context focused. On OSWorld 2.0, StateAct lifts Claude Opus 4.8 from 20.6% to 26.9% on binary success, and from 54.8% to 61.6% on partial success, at $\sim 9\times$ lower cost per task than the same model driven by screenshots alone; a code-only variant with no GUI subagent reaches only 45.9% partial, below that screenshot-based baseline's 54.8%. In general, grounding action, verification, and memory in state, what we call state-grounding, shifts the main bottleneck from perception toward reasoning: failures depend more on what the agent *thinks* than on what it *sees*.

1 Introduction

Computer-use agents operate the real desktop software people use every day, *e.g.*, file managers, spreadsheets, calendars, browsers, email. Most current work improves these agents by strengthening screen perception: better models for reading and acting on rendered pixels (Qin et al., 2025; Xu et al., 2024; Yang et al., 2025a; 2023). On *long-horizon* tasks, however, reading the screen is only part of the problem. Agents must also carry out many dependent operations and determine whether the final result is correct and saved. StateAct therefore makes *program state* (*i.e.*, the files, application backends, and DOM that hold the task data) as the main interface for agents, while retaining visual interaction through a dedicated GUI specialist. We call this *state-grounding*.

The key idea is simple: a desktop task is judged by the state it leaves behind, which a screenshot alone may not reveal. For example, a spreadsheet can display the same total whether a cell contains a formula or a literal, and relevant rows may be hidden or off-screen. Pixels alone hide these task-critical distinctions; direct state access exposes them and edits the values behind them. This gap widens with task length. A single lossy read is usually harmless, but a per-step proxy compounds: over hundreds of dependent operations, small misreads accumulate into a wrong deliverable. Worse, a screenshot offers no dependable signal that the final artifact is correctly complete, exactly the judgment a long-horizon task hinges on. StateAct therefore acts on program state, checks structural completion against the persisted artifact, and uses context management to carry task facts and plans across the long horizon.

[†]Main contributors.

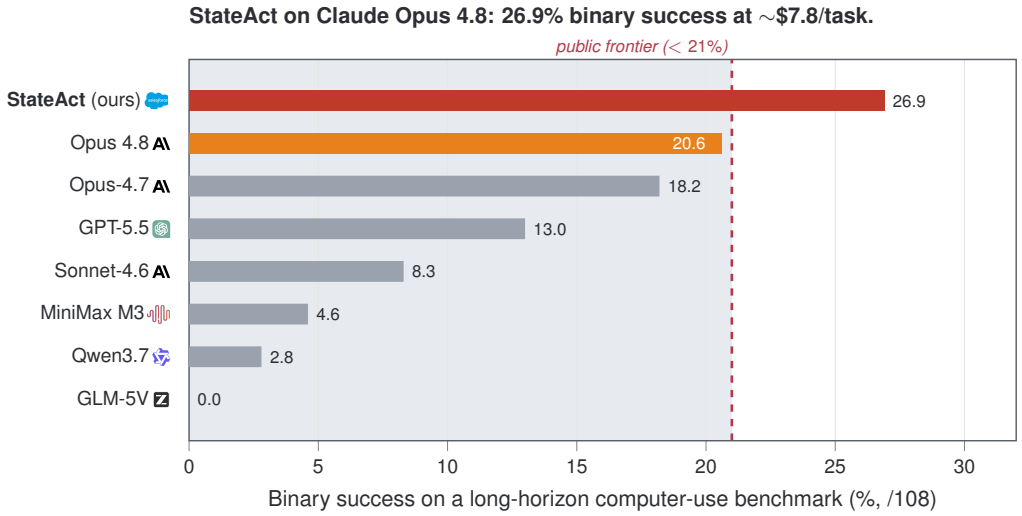

Figure 1: 26.9% binary success on Claude Opus 4.8: the only entry above the $< 21\%$ public frontier, at $\sim 9\times$ lower cost than the same backbone's reference computer-use-agent harness ($\sim$ \$7.8 *vs.* $\sim$ \$72 per task). Orange marks the Claude Opus 4.8 reference baseline.

StateAct (Figure 2) builds the full agent loop around three parts: state-grounded action, verification, and long-horizon execution. For action, the main agent reads and writes the task's real artifacts through persistent `bash`, Python, and a file editor. Code alone is not enough, since not every application exposes a comprehensive API or otherwise accessible state. Empirically, with only `bash` the same agent scores just 45.9% partial success, below the 54.8% vision baseline, so a dedicated GUI subagent supplies screen-based interaction, used on just 28 of 108 tasks and 1.1% of main-agent steps. For verification, a separate gate checks the persisted result without seeing the main agent's account of its work. For long-horizon execution, a growing context of a single agent fills with stale detail and loses task facts over hundreds of steps, so applying fresh-context delegation, compaction, and an externalized plan keeps the main agent focused.

We make three contributions. i) *Architecture*: StateAct makes the main agent observe and act primarily on program state, retains a dedicated GUI subagent for visual interaction (1.1% of main agent steps, 28 of 108 tasks), and carries the same discipline into verification and long-horizon memory (§3, §4). ii) *Empirical gain*: on the strongest backbone (Claude Opus 4.8), the harness alone lifts binary success $20.6\% \rightarrow 26.9\%$ and partial $54.8\% \rightarrow 61.6\%$ over the reference, exceeding every public entry on a system-level comparison (Figure 1 and Table 1). iii) *Diagnosis*: we identify what drives the gain and what limits it. The gain comes from *what* the agent observes (state), not from added agentic depth (offered recursion fires on only 7 of 108 tasks and never nests; §5.3). The limit is value correctness: a state-grounded verifier without a ground-truth oracle catches structural errors but still passes 68 of the 76 non-perfect tasks that reached the gate, on value errors it cannot re-derive independently, *i.e.*, dominantly reasoning errors (§6.1).

2 Related Work

Grounding-based computer-use agents. A large body of work improves the perceptual channel: native GUI action models (UI-TARS (Qin et al., 2025), Aguvis (Xu et al., 2024)), test-time selection (GTA1 (Yang et al., 2025a)), and easier-to-ground observations such as Set-of-Mark (Yang et al., 2023). These make the agent better at reading the screen; StateAct is complementary but inverts the default: it grounds in program state and treats direct screen interaction as a delegated fallback.

Code and hybrid action spaces. Acting through code rather than a UI-shaped schema is well established: CodeAct (Wang et al., 2024) makes executable code the action space, and OSWorld's native action space is Python via `pyautogui`. CoAct-1 (Song et al., 2025) pairs a programmer with a co-equal GUI operator under an orchestrator; UltraCUA (Yang et al., 2025b) and ComputerRL (Lai et al., 2025) learn hybrid API/GUI action spaces via reinforcement learning;

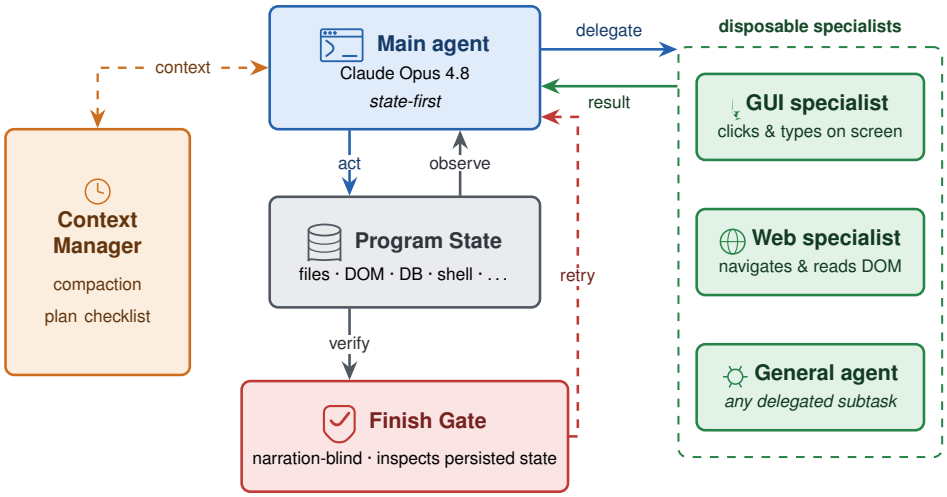

Figure 2: StateAct architecture. The main agent works directly with program state and uses fresh-context GUI and browser specialists for complementary interaction. The finish gate independently checks persisted state, while the context manager preserves the plan and task facts over hundreds of steps. The gate fires on `finish` (after at least three non-finish steps), for up to three rounds.

UFO2 (Zhang et al., 2025) fuses a unified GUI–API layer; and SWE-agent (Yang et al., 2024) argues agents deserve purpose-built interfaces. The architectural distinction is one of default routing: StateAct keeps code/state operations in the main agent and delegates direct GUI control to a subagent, while extending the same state inspection to its verification gate (§4.2).

Self-verification for agents. Grounding verification in system state rather than model narration is likewise established. OpenComputer (Wei et al., 2026) uses *hard-coded*, per-application state verifiers that align with human adjudication more closely than an LLM judge; Agentic Reward Modeling (Cui et al., 2026) probes hidden system state via proactive interaction; MCPWorld (Yan et al., 2025) verifies via backend instrumentation. Our finish gate is deliberately *weaker on value but stronger on generality*: it is a single prompt-level check that applies to all 108 tasks with *zero* per-task verifier engineering, trading OpenComputer's oracle-grade value checking for a structural-only ceiling we measure and own (§6.1). We also distinguish the finish gate from iterative self-correction. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) have the agent reflect on its *own* narrated trajectory, making self-consistency their substrate. Our finish gate is the opposite: a fresh, independent, third-party check at a single terminal boundary that *refuses* to read the agent's narration and directly inspects the persisted deliverable. This is closer to an actor–critic separation or an independent acceptance test than to reflection. This matters because narration-conditioned judges have been shown to be biased by the behavior they observe (Andrade et al., 2025). Designing narration out is the point, and §6.1 measures how far it gets us.

3 The State-Grounding Principle

A model of the two channels. Let $s \in \mathcal{S}$ be the computer program state (*e.g.*, files, application backends, DOM, tables). An agent never observes s directly; it observes it through a channel. The *pixel channel* is the render map $o_{\text{pix}} = f_{\text{render}}(s)$, and the *state channel* is a query map $o_{\text{state}} = g(s)$ (*e.g.*, a shell command, a workbook read, a DOM serialization). Two structural facts drive StateAct.

i) Rendering is lossy and non-injective. Many distinct states render to indistinguishable, or arbitrarily close, screenshots (Figure 3, a displayed total may hold a literal or a formula, be rounded, or be scrolled off-screen). Formally f_{render} is not injective, so f_{render}^{-1} does not exist and no perceptual model, however good, can in general recover s from o_{pix}. Over the substate a task touches, the state channel g is effectively invertible. For example, in Figure 3 (b), from `load_workbook(...)["B7"].value` the agent recovers the formula exactly. The loss is benign in one shot but compounds over a *long horizon*: a per-step lossy proxy accumulates state drift

```python
# read a total off the screen
screenshot() # "cell shows 1,240"
# display may omit formula,
# hidden rows, or precision
click(x=734, y=512)
# coordinate-based
type("=SUM(B2:B6)")
# depends on focus
```

(a) Pixel path – rendered, indirect

```python
# after locating artifact/cell
import openpyxl as x
wb = x.load_workbook("r.xlsx")
c = wb["Sheet1"]["B7"]
print(c.value) # stored formula
c.value = "=SUM(B2:B6)"
wb.save("r.xlsx") # persisted
```

(b) State path – targeted, queryable

Figure 3: The same subgoal via each channel. The pixel channel yields a rendered, ambiguous value at a fragile coordinate; the state channel reads and writes the exact artifact the task's deliverable is made of. StateAct uses (b) for state-addressable operations and a dedicated GUI subagent for (a) when visual interaction is required.

across hundreds of steps, and the state channel's advantage grows with task length, which is the regime this paper targets.

ii) The deliverable is *the state.* A desktop task deliverable is a change to program state, not a screenshot of it. Whether a task succeeded is determined by the program state (*e.g.*, whether the user goal is met in the saved file or application settings). Formally, let $G(s)$ denote the success predicate over state. For tasks whose deliverable depends on non-rendered content (formulas, hidden rows, off-screen data, backend state), $G(s)$ cannot be recovered from $f_{\text{render}}(s)$ alone (there is no $\tilde{G}$ with $G(s) = \tilde{G}(f_{\text{render}}(s))$ for all such s), because rendering discards that content. A GUI may still reach such content when the interface exposes it, but usually only through multiple steps, such as selecting a cell to reveal its formula or scrolling hidden rows into view; the state channel reads and writes it directly and exactly. For tasks that only require reading and summarizing information (no state change), either channel can work; the distinction matters only when the deliverable depends on content the screen does not show. For such tasks, acting and checking on $g(s)$ operates on the *actual artifact the task asks for*, whereas acting on $f_{\text{render}}(s)$ operates on a lossy image of it.

Boundary: where the state channel does not help. The model delimits the principle's scope. When a task targets a visual outcome (*e.g.*, image editing, layout, chart appearance, WYSIWYG output), the rendered screen is the relevant observation, and the state channel gives no leverage. More generally, any subgoal that can only be expressed as a rendered interaction (dragging a canvas element, dismissing a non-scriptable modal, reading a value that exists only on-screen) falls outside the state channel's reach. We accordingly distinguish three conceptual task types: *state-addressable* (*i.e.*, a code path to the target artifact exists), *hybrid* (*i.e.*, mostly state, one irreducibly visual subgoal), and *render-only* (*i.e.*, pixel/appearance judgments). The principle predicts state-grounding helps the first two and gives no advantage on the third. In our evaluation (§5), the per-capability breakdown (Table 1) is qualitatively consistent: StateAct's margins are largest on state-addressable capabilities and smallest on the most render-only ones, human-in-the-loop and multimodal editing.

4 StateAct

StateAct (Figure 2) has three components: i) a main agent that acts through code on program state, ii) an independent finish gate that verifies completion by re-reading the real artifacts, and iii) context management that sustains the run over hundreds of steps. Figure 5 shows example trajectories.

4.1 Act on state

The main agent action space is code and structured operations: persistent `bash`, a file editor, a read-only `view_image` for image *files*, a `plan` checklist, a `finish` action, and an `agent` delegation tool. No live screen actuation (*e.g.*, mouse or keyboard) is exposed to the main agent.

Table 1: Per-capability performance on OSWorld 2.0's ten capability labels, sorted by StateAct; each cell is binary / mean-partial (%). StateAct (bold) is our state-grounding harness on Claude Opus 4.8; *Opus 4.8 (ref.)* is the same backbone under the computer-use-agent (CUA) harness. For Opus-4.7, we report only its best reported batched configuration. "–" denotes not available.

Capability	Binary / mean-partial score (%)							
	StateAct (Opus 4.8)	Opus 4.8 (ref. CUA)	GPT-5.5	Opus-4.7	Sonnet-4.6	GLM-5V	MiniMax M3	Qwen3.7
Multi-item state	**27.9/66.7**	–	14.0/50.6	–	11.6/46.7	0.0/11.2	7.0/23.2	2.3/20.2
Streaming	**66.7/66.7**	–	50.0/57.8	–	0.0/4.7	0.0/1.2	0.0/6.4	0.0/0.0
Cross-source	**26.1/64.9**	–	13.0/52.4	–	10.9/45.8	0.0/11.7	6.5/24.9	6.5/26.3
Conflict disambig.	**30.8/64.5**	–	20.5/51.4	–	12.8/42.4	0.0/12.4	7.7/24.3	7.7/29.1
Visual-spatial	**31.1/61.3**	–	11.1/51.2	–	8.9/36.5	0.0/4.6	4.4/19.8	2.2/19.8
Implicit state	**30.2/60.1**	–	14.0/47.3	–	9.3/37.0	0.0/11.0	4.7/24.4	2.3/24.1
Tutorial	**13.6/55.3**	–	9.1/37.5	–	13.6/43.5	0.0/7.8	4.5/15.0	4.5/15.7
Multimodal edit	**20.0/54.2**	–	6.7/47.0	–	6.7/37.5	0.0/6.2	6.7/22.3	0.0/20.6
Dynamic env.	**30.0/53.5**	–	30.0/46.2	–	10.0/22.0	0.0/5.8	0.0/17.9	0.0/16.3
Human-in-the-loop	**0.0/43.9**	–	16.7/43.1	–	16.7/51.9	0.0/12.5	0.0/16.8	0.0/22.5
Overall (/108)	**26.9/61.6**	20.6/54.8	13.0/49.5	18.2/48.9	8.3/41.5	0.0/9.0	4.6/22.3	2.8/21.5

State discovery. The main agent acts to find where an application persists its state, drawing on two signals: the model own priors about how common desktop applications store state on disk (*e.g.*, mail stores, office-document formats, browser profiles, application databases), and active probing (*e.g.*, `find`/`ls`/`grep`/`sqlite3`) when the prior is uncertain. Discovery locates *where* state lives, never *what* the target value is.

Delegation rule. The main agent uses the dedicated `cua` (GUI) subagent when a subgoal is *irreducibly visual*: i) no file/backend/DOM path to the target could be found by probing, or ii) the effect is only expressible as a rendered interaction (dragging a canvas selection, dismissing a non-scriptable modal, or reading a value that exists only on-screen). The measured GUI-use rate is 1.1% of main agent steps (28 of 108 tasks touch the GUI subagent at least once), rising to $\sim 11\%$ of total model turns once the subagent's interior turns are included.Browser work is handled by a dedicated `web` subagent that navigates, executes JavaScript, serializes the DOM to markdown, and clicks by CSS selector, grounding it in structured state rather than a screenshot.

4.2 Verify on State

When the main agent calls `finish` (permitted only after at least three non-finish steps), an independent finish gate (Figure 4) spawns. Its context contains *only* the verbatim task instruction and machine access (`bash`, file reads, DOM queries), with editor mutations blocked. It never sees the main agent's message history, plan, or finish rationale, nor the expected values. This gives the gate four properties. i) Narration-blind: it sees only the task and the machine, not the agent's claims; unlike a Reflexion-style self-critique, it cannot be talked into agreement by the trajectory's own story. ii) State-grounded: it checks the requested result in the exact persisted artifact (*e.g.*, the calendar store, the app backend, the saved spreadsheet's cell formulas), rather than relying on the UI. iii) Anti-capture: it must independently locate the real deliverable the task names (*e.g.*, the exact file or backend the instruction asks to change) and ground its checks there; evidence found only in a side file the agent created itself, rather than in the deliverable the task names, is rejected. iv) Bounded correction: on rejection, the main agent is sent back to fix the named gap, up to three rounds. We report the gate's measured operating behavior in §6.1.

4.3 Sustain State

Long episodes (up to 200 main agent turns, plus a mean of 4.2 subagent delegations per task, each its own $\leq$50-turn loop) would overflow a naive context. Three mechanisms address this. i) Fresh-context specialists: each delegation runs in its own context window, seeded with a focused subtask and returning a concise report, so image-heavy or exploratory subwork stays out of the main agent's state model. ii) Auto-compaction: near the context limit, the oldest prefix is summarized at an assistant boundary and images are stripped, preserving state facts while reclaiming context budget.

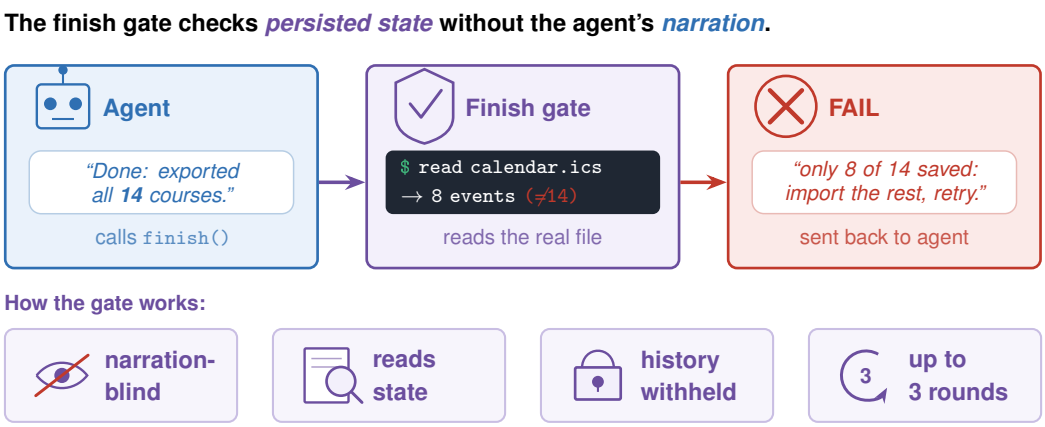

Figure 4: An independent finish gate transcript (condensed from the calendar-import failure class in §6.1). The gate checks the actual persisted store, catching that only 8 of 14 events were saved (the `Downloads` copy was complete but the store was not). This is the *structural* defect class the gate catches reliably; §6.1 reports the class it does not.

iii) Externalized plan: a task checklist persisted outside the message history is re-injected each turn, surviving compaction and anchoring multi-part tasks.

5 Experiments

5.1 Experimental setup

We evaluate StateAct on OSWorld 2.0 (Yuan et al., 2026), a standard long-horizon GUI benchmark (108 tasks). Following the benchmark's protocol, we report binary success, mean partial score (fractional credit), and cost per task (USD). All runs use Claude Opus 4.8 with adaptive thinking and a 200-turn main-agent budget per agent invocation.

Turn accounting. We distinguish model turns from tool uses. The main agent averages $\sim$ 57 model turns per episode; each subagent delegation adds $\sim$23 interior turns (capped at 50), for $\sim$ 155 model turns per task ($\sim$ 57 main agent + $\sim$ 98 subagent). We use that total for the cross-system comparison in Table 2.

5.2 Main result

On the identical backbone (Table 1), replacing the computer-use-agent harness with StateAct raises binary success by $+6.3$ points and mean partial by $+6.8$, while cutting output tokens (224K $\rightarrow$ 100K) and dollar cost (from $\sim$ \$72 (Yuan et al., 2026) to $\sim$ \$7.8 per task, a $\sim 9\times$ reduction). StateAct (26.9% / 61.6%) is the best-performing entry, exceeding the same-backbone reference.

On the cost–accuracy frontier (Figure 6), StateAct sits alone in the top-left: it exceeds the best public entry (Opus-4.7, 18.2%) by 8.7 binary points and GPT-5.5 (13.0%) by 13.9, and adds $\geq$12 partial points over both, while costing $\sim$ \$7.8 per task, below GPT-5.5 ($\sim$ \$25.5) and Opus-4.7 ($\sim$ \$33.6) and far below the same-model reference ($\sim$ \$72); among the displayed systems, only the much weaker MiniMax M3 and Qwen entries cost less. Compared to the reference computer-use-agent harness on Claude Opus 4.8, StateAct is both more accurate ($+6.3$ / $+6.8$ points higher) and $\sim 9\times$ cheaper, showing that state-grounding improves quality and cost simultaneously.

5.3 Ablation

Component sensitivity (Table 3a). We remove one component at a time from full StateAct: $-$act re-introduces the computer tool (GUI in the main agent, `cua` blocked); $-$verify disables the finish gate; $-$sustain removes compaction and the plan tool. Removing act-on-state produces the largest drop (partial 61.6% $\rightarrow$ 51.3%, below even the reference's 54.8%), consistent with code-first action

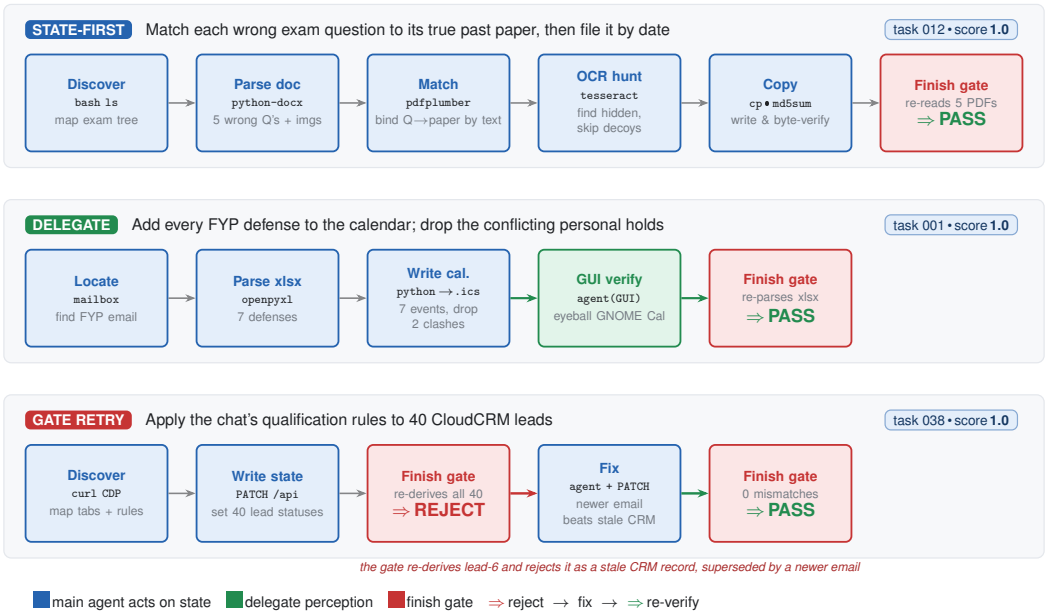

Figure 5: Three successful trajectories selected to illustrate state-first action, GUI delegation, and a gate retry (chips = key steps, color-coded by role; all reach score 1.0). State-first (task 012): the main agent acts entirely in code, using `python-docx`/`pdfplumber`/`tesseract` to bind each wrong exam question to its true past paper, correctly rejecting the distractor questions, with no GUI at all. Delegate (task 001): code writes the calendar `.ics` directly and a GUI specialist is invoked *only* to eyeball GNOME Calendar. Gate retry (task 038): the finish gate re-derives all 40 CRM lead statuses from source and rejects lead-6 as sourced from a stale record; the agent re-derives it from a newer email that supersedes the stale record, and re-verification passes.

Table 2: Mean model turns per task, by capability. For StateAct: main agent plus subagent interior turns (57+98=155). Public entries are from publicly released OSWorld 2.0 trajectories. "–" denotes per-task trajectory is not available. Capabilities are multi-label, so a task can count toward several columns; the per-capability means therefore do not average to *Overall*, which is the per-task mean over all 108 tasks.

Model	Stream	Tutor	Vis-sp	M-edit	Dyn	Implic	H-loop	M-item	X-src	Confl	Overall
StateAct	170	173	137	136	253	180	282	163	193	184	155
Opus 4.8	–	–	–	–	–	–	–	–	–	–	103
GPT-5.5	152	98	85	78	111	105	72	107	102	93	95
Opus-4.7	–	–	–	–	–	–	–	–	–	–	161
Sonnet-4.6	418	319	229	237	288	284	198	267	270	235	253
MiniMax M3	314	370	309	310	312	351	282	345	362	341	327
GLM-5V	321	293	299	341	329	313	320	338	338	333	314

on state being the largest single contributor. Disabling the gate (57.5%) or context management (58.7%) each produces a smaller drop.

Delegation depth (Table 3b). We hold state-grounding fixed and vary depth: i) flat delegation (default StateAct), ii) worker recursion (depth $\leq$2), iii) nested self-recursion with its own finish gate. Flat delegation leads on mean-partial (61.6 *vs.* 57.6 / 57.9). The recursive branch fired on only

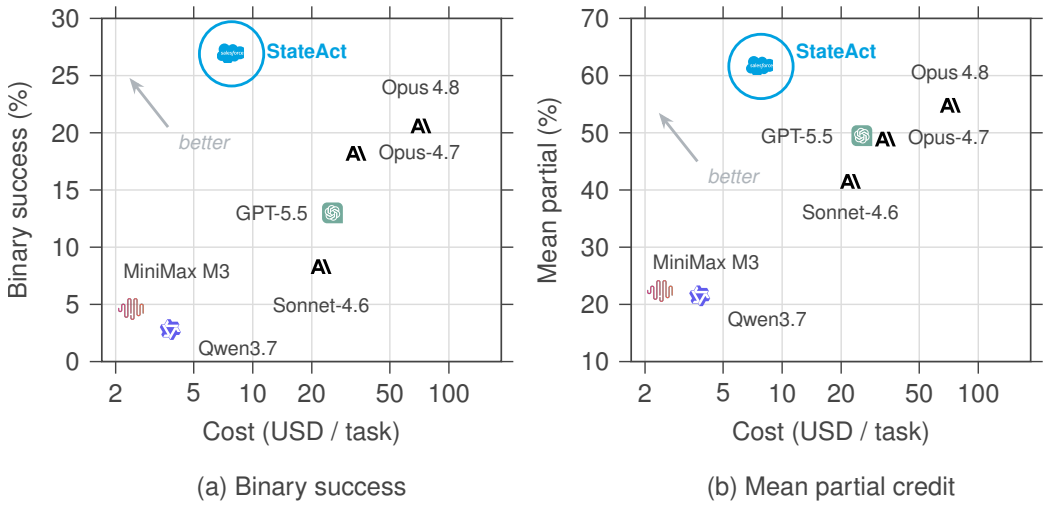

Figure 6: Cost–accuracy frontier on OSWorld 2.0 (binary success *vs.* USD per task). Public points are from publicly released OSWorld 2.0 trajectories (Yuan et al., 2026). StateAct (red) sits in the top-left: more accurate and cheaper than every stronger public baseline shown; only the much weaker MiniMax M3 and Qwen entries cost less.

Table 3: Ablation on components and delegation depth. (a) Component sensitivity (binary / mean-partial, %): each row removes one mechanism from full StateAct; act-on-state matters most. (b) Main agent depth by capability (mean partial, %): flat delegation, worker recursion, nested self-recursion.

(a) Component ablation

Config	Binary	Part.
StateAct	**26.9**	**61.6**
−verify (no finish gate)	23.1	57.5
−sustain (plan off)	21.3	58.7
−act (GUI; cua blocked)	18.5	51.3

(b) Recursion scaffolds (per capability, mean partial)

Config	M-item	Stream	X-src	Confl	Vis-sp	Implic	Tutor	M-edit	Dyn	H-loop
flat (StateAct)	**67**	**67**	**65**	**65**	**61**	**60**	**55**	**54**	**54**	**44**
worker	58	62	58	55	55	55	53	52	54	39
nested	61	67	59	56	59	57	54	49	58	35

7 of 108 tasks and never nested, so the between-config differences cannot be attributed to depth. This supports a design choice (keep StateAct flat), not a universal claim about hierarchy.

5.4 Additional designs

Table 4 reports additional designs. A bash-only configuration (no GUI, subagents, or finish gate) reaches 45.9% partial, below even the reference (54.8%): code-action alone is not enough without the scaffold. On Claude Sonnet 4.6, the same harness lifts binary success from 8.3% to 11.1%, evidence that state-grounding helps a weaker backbone. On OSWorld-Verified, a short-horizon benchmark, StateAct and the reference perform similarly (78.4% *vs.* 77.3% binary; Table 4c), consistent with state-grounding's advantage being concentrated on long-horizon tasks.

6 Discussion

6.1 Why State-Grounding Helps: A Failure Analysis

Having established the gain (§5), we ask *where* the harness pays off and *what* remains. Figure 7 breaks scores down by capability across six systems with available trajectories; Figure 8 partitions the 79 non-perfect tasks by root cause. The pattern: StateAct's largest margins fall on capabilities whose state is machine-checkable (multi-item state, cross-source reasoning, conflict disambiguation), while the weakest capabilities (human-in-the-loop, multimodal editing) need either an interaction turn the harness never solicits or a flawless long visual chain. Averaging the ten capability

Table 4: Additional system comparisons (binary / mean-partial, %). **(a)** Opus 4.8 diagnostics: bash-only exposes only a shell. **(b)** Sonnet backbone transfer against the reported reference-harness aggregate. **(c)** OSWorld-Verified, a short-horizon benchmark (same Opus 4.8 backbone).

(a) Opus 4.8 diagnostics.

System	Bin	Part
Bash-only	15.7	45.9
StateAct	**26.9**	**61.6**

(b) Sonnet 4.6.

System	Bin	Part
Reference	8.3	41.5
StateAct	**11.1**	**42.0**

(c) OSWorld-Verified.

System	Bin	Part
Reference	77.3	80.9
StateAct	**78.4**	**81.9**

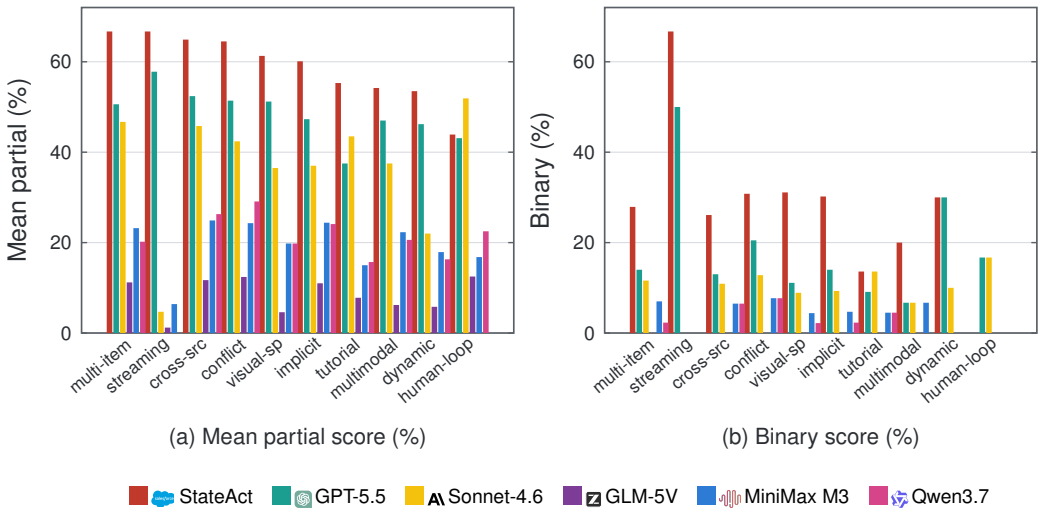

Figure 7: Per-capability performance across six systems on OSWorld 2.0's ten multi-label capability groups (% within each group): (a) mean partial credit, (b) binary success. StateAct (red) uses Claude Opus 4.8; public points are from publicly released OSWorld 2.0 trajectories (Yuan et al., 2026). The best Opus-4.7 batch run and the same-backbone reference have no released per-capability trajectories and are therefore omitted.

rows equally gives ∼59 partial *vs.* ∼27 binary (the /108 aggregate is 61.6/26.9). This persistent partial-vs-binary gap reflects the verifier ceiling analyzed next.

Two recoverable classes and a hard residual. The audit (Figure 8) groups most of the 79 non-perfect tasks into two recoverable classes: *agent-reasoning* (a wrong value or misread instruction, 38 tasks, the dominant mode) and *verifier-weak / wrong-path* finishes that the finish gate wrongly accepted (14). These 52 tasks are plausibly addressable (the errors are reasoning failures or verification misses). A further 22 tasks (∼ 20% of the suite) either hit modality bottlenecks the backbone cannot cross (audio, video, real-time interaction) or carry ambiguous instructions with several valid readings. The remaining 5 tasks were left undecomposed. Acting on state removes most perception failures: with code, "read cell `B7`" or "list the calendar store" is exact. The finish gate targets persistence failures: wrong-path deliverables, unsaved edits, format mismatches. What neither addresses is reasoning, the dominant recoverable class (38 tasks).

The verifier's ceiling. The finish gate catches *structural* defects (missing file, wrong path, format mismatch) but cannot adjudicate *value* correctness: re-deriving from the same source under the same interpretation reproduces the agent's wrong answer. Of the 79 non-perfect tasks, 76 reached the gate (Table 5); it correctly rejected only 8 and wrongly passed 68, an error rate of 68/76 (≈90%). This rate is high because the gate checks only structure: most of these passes fail on value or reasoning errors no structural check can catch, and only 14 are misses a structural check could have caught. This is not narration-conditioned agreement bias (Andrade et al., 2025) (the gate is independent of the agent's narration) but a common-mode failure: shared interpretation of the source. The 8 correct rejections are high-precision structural catches that drive the bounded retries (Figure 4). The ceiling is intrinsic: without ground-truth labels, a prompt-level verifier bounds structural, not value,

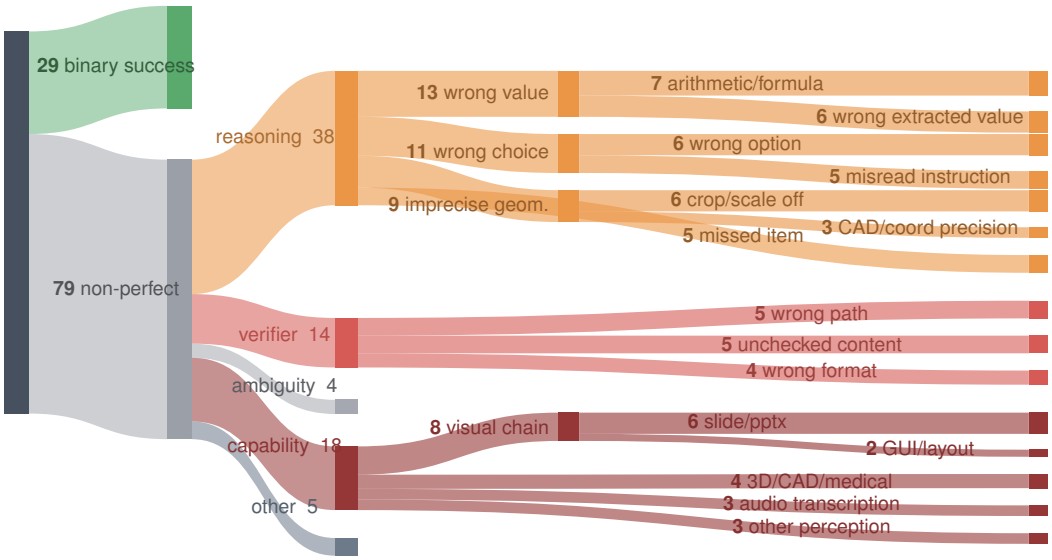

Figure 8: Per-task dominant-cause audit of the 79 non-perfect tasks. Reasoning errors (38) dominate; $\sim 20\%$ hit modality bottlenecks or ambiguous instructions. Categories are manual audit labels from trajectory inspection.

Table 5: Finish-gate operating characteristics. i) Gate verdicts *vs.* grader outcomes (29 binary successes, 79 non-perfect, 108 total; 3 non-perfect tasks never called `finish` and thus never reached the gate). ii) Operating rates of the finish gate.

(a) Final verdict *vs.* grader

	Binary success	Non-perfect	Total
PASS	28	68	96
FAIL	1	8	9
No finish called	0	3	3
Total	29	79	108

(b) Operating rates

	Fraction	Rate
Correct rejections / all rejections	8/9	88.9%
Correct rejects / non-perfect reached	8/76	10.5%
Correct passes / binary successes	28/29	96.6%
Tasks retried	28/105	26.7%
Retries ending in binary success	6/28	21.4%

correctness. This is the axis on which hard-coded per-task verifiers (Wei et al., 2026) win, at the cost of generality.

6.2 Is a strong GUI subagent necessary?

StateAct quarantines GUI interaction to a subagent invoked on only 1.1% of main-agent steps (28 of 108 tasks), so the main agent rarely relies on screen-based control. Must that subagent be a frontier GUI model? We swap Claude's computer-use model for SFR-CUA, our compact 31B in-house computer-use model, holding Claude Opus 4.8 as the main agent in both, and evaluate across five benchmarks (Table 6; the parenthetical names the GUI subagent, and Claude Opus 4.8 is the reference).

On four of the five benchmarks the substitution barely moves the end-to-end mean-partial score: OSWorld-Verified (81.1 *vs.* 81.9), WindowsAgentArena (51.2 *vs.* 50.6), AndroidWorld (84.1 *vs.* 81.9), and MobileWorld (68.4 *vs.* 70.1). This holds even though SFR-CUA on its own is far weaker than Claude Opus 4.8, scoring 66.9 *vs.* 80.9 on OSWorld-Verified and 7.6 *vs.* 54.8 on OSWorld 2.0. Once the main agent carries the task on state, a compact specialist suffices for the rare visual fallback. The exception is OSWorld 2.0, our longest-horizon suite: there StateAct (SFR-CUA) reaches only 43.2% partial and 18.5% binary, below StateAct (Claude Opus 4.8) at 61.6%/26.9% and the base model at 54.8%/20.6%, because its harder visual subgoals expose the weaker subagent. A frontier GUI model is thus unnecessary on the shorter-horizon and mobile benchmarks but still helps on the hardest long-horizon desktop tasks.

Table 6: Performance across five benchmarks (mean-partial score, %): OSWorld-Verified (Xie et al., 2024), OSWorld 2.0 (Yuan et al., 2026), WindowsAgentArena (Bonatti et al., 2024), AndroidWorld (Rawles et al., 2025), and MobileWorld (Kong et al., 2025).

Model	OSWorld Verified	OSWorld 2.0	WindowsAgent Arena	Android World	Mobile World
Claude Opus 4.8	80.9	54.8	41.6	69.0	51.3
SFR-CUA	66.9	7.6	40.9	68.1	48.7
StateAct (Claude Opus 4.8)	81.9	61.6	50.6	81.9	70.1
StateAct (SFR-CUA)	81.1	43.2	51.2	84.1	68.4

7 Conclusion

We presented StateAct, a harness that makes program state as the primary interface for the main agent while retaining a dedicated GUI subagent for visual interaction. On a standard long-horizon GUI benchmark, Claude Opus 4.8 rises from 20.6% to 26.9% binary success (54.8% to 61.6% partial) at $\sim 9\times$ lower cost, without any change to the model itself. The ablation and diagnostic analyses localize both the gain and its limit: the gain comes from *what* the agent observes (state rather than screenshots), not from added depth; the limit is value correctness, which a self-verifier without ground-truth labels cannot close. State-grounding moves the accuracy wall from perception to reasoning; it does not remove it. For long-horizon computer use, the bottleneck is now what the agent *thinks*, not what it *sees*.

References

Moises Andrade, Joonhyuk Cha, Brandon Ho, Vriksha Srihari, Karmesh Yadav, and Zsolt Kira. Let's think in two steps: Mitigating agreement bias in MLLMs with self-grounded verification. *arXiv preprint arXiv:2507.11662*, 2025.

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.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. Agentic reward modeling: Verifying GUI agent via online proactive interaction. *arXiv preprint arXiv:2602.00575*, 2026.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

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. *arXiv preprint arXiv:2508.03923*, 2025.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. *arXiv preprint arXiv:2402.01030*, 2024.

Jinbiao Wei, Qianran Ma, Yilun Zhao, Xiao Zhou, Kangqi Ni, Guo Gan, and Arman Cohan. OpenComputer: Verifiable software worlds for computer-use agents. *arXiv preprint arXiv:2605.19769*, 2026.

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 (NeurIPS), Datasets and Benchmarks Track*, 2024.

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

Yunhe Yan, Shihe Wang, Jiajun Du, Yexuan Yang, Yuxuan Shan, Qichen Qiu, Xianqing Jia, Xinge Wang, Xin Yuan, Xu Han, Mao Qin, Yinxiao Chen, Chen Peng, Shangguang Wang, and Mengwei Xu. MCPWorld: A unified benchmarking testbed for API, GUI, and hybrid computer use agents. *arXiv preprint arXiv:2506.07672*, 2025.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, Ran Xu, Liyuan Pan, Silvio Savarese, Caiming Xiong, and Junnan Li. GTA1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*, 2025a.

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

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. OSWorld2.0: Benchmarking computer use agents on long-horizon real-world tasks. *arXiv preprint arXiv:2606.29537*, 2026.

Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, Liqun Li, Yu Kang, Zhao Jiang, Suzhen Zheng, Rujia Wang, Jiaxu Qian, Minghua Ma, Jian-Guang Lou, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. UFO2: The desktop AgentOS. *arXiv preprint arXiv:2504.14603*, 2025.