ACCESSWORLD: A BENCHMARK FOR COMPUTER USE AGENTS OPERATING ASSISTIVE TECHNOLOGIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer use agents typically perceive graphical interfaces through screenshots or accessibility trees and act through mouse clicks and keystrokes. Many people with disabilities instead operate the same interfaces through assistive technologies (ATs), such as screen readers, voice control, and screen magnification. As a result, the action and observation spaces available to a user can differ greatly across interaction modalities. Assistive agents that can operate these technologies could collaborate with the people who rely on them, audit whether software actually works with ATs, and guide users through interfaces that are hard to navigate. There is also no reason to assume a GUI built for sighted mouse users is the right interface for an agent. Yet there are currently no tools or benchmarks for building and evaluating agents that operate ATs. We present AccessWorld, a benchmark of computer use tasks on macOS that agents complete by using system native assistive technology. Agents operate on the environment through an OpenClaw harness exposing each AT as an agent tool interface. We evaluate five models across five interaction modalities and find substantial differences across ATs in both task success and interaction cost. Through VoiceOver, macOS's screen reader, one model nearly matches its standard computer-use success, but requires substantially more actions and cumulative input. At $2\times$ magnification, three of five models exceed their standard computer-use accuracy, while increasing magnification to $5\times$ reduces accuracy for all five models. Voice Control produces the lowest accuracy for every model, with the largest drops on multi-item, spreadsheet, and text-editing tasks. AccessWorld measures how well agents handle the constrained observation and action spaces that AT users work with every day.

1 INTRODUCTION

People with disabilities often use assistive technologies (ATs) to perceive, navigate, and operate graphical user interfaces (GUIs). Screen readers convey interface elements and text through synthesized speech or braille for people who are blind or have low vision; screen magnifiers enlarge visual content for people with low vision; and voice control lets people with motor impairments operate interfaces through spoken commands (W3C Web Accessibility Initiative, 2024). These technologies change both the information available from an interface and the actions used to operate it. Computer use agents (CUAs), in contrast, typically observe interfaces through screenshots or accessibility trees and act through mouse clicks and keystrokes (Nguyen et al., 2025; Xie et al., 2024). For this reason, most CUAs bypass the AT through which a user may currently be interacting with an application.

Agents that operate through ATs could work within users' existing workflows (Kodandaram et al., 2024; Peng et al., 2025) and test whether software remains operable through those technologies (Taeb et al., 2024; Zhong et al., 2026). These uses require the AT itself to be part of the agent's interaction loop. The macOS native screen reader VoiceOver, for example, maintains its own focus and returns spoken feedback during navigation; directly changing the application with a mouse does not reproduce this interaction. A11y-CUA is a study comparing blind and low vision users with computer use agents operating a screen reader or magnifier, which found that task completion accuracy drops dramatically when using ATs. However, the agents in this study received desktop screenshots rather than receiving feedback solely from the running screen reader (Gubbi Mohanbabu et al., 2026). Existing work does not provide an interactive environment in which general-purpose CUAs can repeatedly observe and act through multiple native ATs.

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.

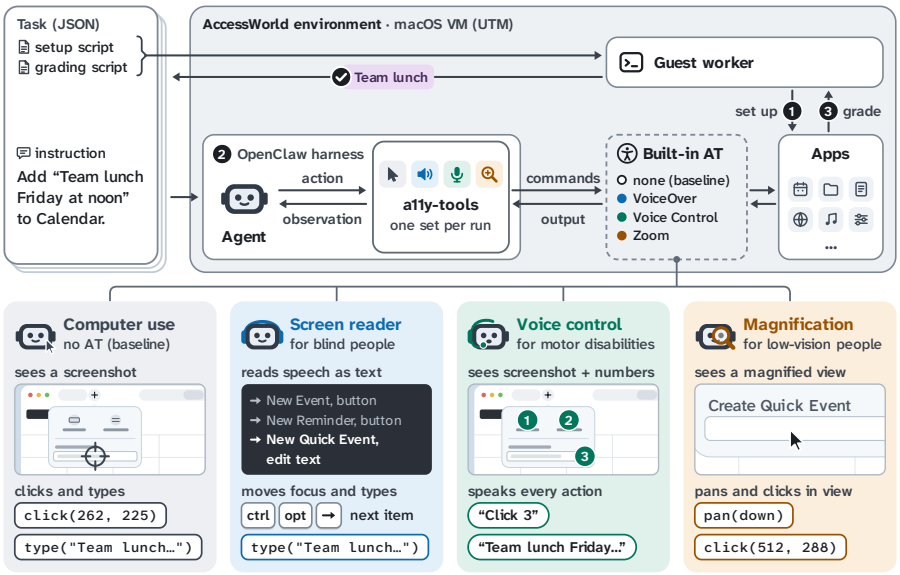

Figure 1: **AccessWorld Overview.** Each task has an instruction and grader, with task-specific setup when needed, and runs in the same macOS virtual machine. OpenClaw connects the agent to standard computer use, VoiceOver, Voice Control, or Zoom. The task and grader stay fixed while the available observations and actions change with the interaction condition.

We present AccessWorld, an interactive macOS environment that draws on the platform's built-in assistive tools to evaluate CUAs through three assistive technologies: VoiceOver for screen reading, Zoom for screen magnification, and Voice Control for speech-based interface control (Figure 1). AccessWorld contains 124 tasks adapted from existing macOS CUA benchmarks. The task instruction, environment setup, and grader remain fixed across conditions, while OpenClaw exposes the feedback and commands available through each interface. An agent using VoiceOver, for example, receives VoiceOver feedback and acts through VoiceOver commands rather than screenshots and direct GUI clicks.

We evaluate five models under standard computer use and each AT condition. VoiceOver nearly matches standard computer-use accuracy for one model but requires substantially more actions and cumulative input. At $2\times$ magnification, three of five models exceed their standard computer-use accuracy, while increasing magnification to $5\times$ reduces accuracy for all five models. Voice Control produces the lowest accuracy for every model. Aggregate accuracy also hides differences in which tasks agents complete: for GPT-5.6 Sol, standard computer use and VoiceOver differ by only 2.4 percentage points overall, yet 35 of 124 task outcomes change between the two conditions. Standard computer-use accuracy alone therefore does not characterize how an agent performs through an assistive technology.

2 Related Work

Computer-use benchmarks evaluate agents across web and desktop environments. WebArena and WorkArena focus on web interaction (Zhou et al., 2024; Drouin et al., 2024), while OSWorld and Windows Agent Arena evaluate general desktop use (Xie et al., 2024; Bonatti et al., 2025). Recent benchmarks extend these settings to long-horizon workflows, cross-application tasks, and combined GUI and command-line interaction (Yuan et al., 2026; Li et al., 2026a;b). On macOS, macOSWorld (Yang et al., 2025), MacArena (Muryn et al., 2026), and MacAgentBench (Fu et al., 2026) provide interactive environments for desktop tasks. These benchmarks generally expose screenshots, accessibility trees, or tool outputs and let agents act through direct mouse, keyboard, or general-purpose tools. Frameworks such as BrowserGym provide reusable interfaces for running agents across benchmarks (de Chezelles et al., 2025), but general-purpose CUA benchmarks do not place a native AT inside the agent's observation and action loop.

A11y-CUA is the closest prior benchmark for evaluating CUAs under AT conditions (Gubbi Mohanbabu et al., 2026). It studies agent performance under standard, screen-reader, and magnified-view conditions and provides a dataset of recorded trajectories across 60 tasks. In its screen-reader condition, the agent retains the full visual desktop and acts through the keyboard rather than receiving feedback from the running screen reader. Other systems use AT-mediated interaction for accessibility auditing or user assistance, including agents that act through TalkBack on Android or VoiceOver on iOS and interfaces designed to support screen-reader users during computer use (Zhong et al., 2026; Taeb et al., 2024; Kodandaram et al., 2026). These systems focus particular ATs or accessibility use cases rather than a common environment for evaluating general-purpose CUAs across multiple ATs. AccessWorld provides such an environment on macOS, with the same tasks, setup procedures, and graders across standard computer use, VoiceOver, Voice Control, and Zoom.

3 AccessWorld

3.1 Environment and Task Suite

AccessWorld is a benchmark for evaluating computer use agents using assistive technology to complete desktop GUI tasks. Agents interact with the environment through the native accessibility features provided in macOS such as the screen reader VoiceOver, magnifier Zoom, and speech dictation Voice Control. Each of these ATs provide unique capabilities and restrictions on how an agent perceives and operates on the computer, allowing for comparison across fixed environment tasks. Thus, the trajectory of task completion an related challenges is faced not only by the instruction, but by the system and app specific accessibility to that modality of interaction.

Each episode runs in a fresh clone of a configured macOS virtual machine: a setup script prepares the initial state, the agent attempts the task, and a grader checks the resulting computer state. The 124 tasks comprise 82 adapted from macOSWorld and 42 from MacArena (Yang et al., 2025; Muryn et al., 2026). They include typical CUA benchmark categories such as file management, productivity work, system settings, and native and third-party application use. After achieving the desired state or hitting the action cap, the environment state is inspected by a deterministic grader script, which produces a binary accuracy score.

3.2 Agent Harness

AccessWorld uses a minimal OpenClaw configuration to run each task as a separate agent session. At the start of an episode, the orchestrator provides the task instruction and the prompt for the selected interaction condition. Each condition defines a tool allowlist that determines what the agent can observe and which actions it can issue. AccessWorld adapters connect these tool calls to the corresponding macOS controls and AT feedback. The agent alternates between model responses and tool execution until it returns a final response or reaches the action limit. We disable OpenClaw's general tool profile and automatic workspace context, so agents cannot use unrelated shell, filesystem, web, memory, or subagent tools. Each task receives a separate session, and no interaction history is carried between tasks.

We use condition-specific action limits because the interfaces expose actions at different granularity. Standard computer use allows 15 charged actions, while VoiceOver allows 90 and Zoom and Voice Control allow 30. This follows and adapts the precedent of A11y-CUA, which assigns a larger step budget to AT-constrained conditions than to its default CUA (Gubbi Mohanbabu et al., 2026). We use the same action limit for every model within a condition. Because these limits differ across conditions, our cross-condition results characterize the evaluated agent–interface configurations rather than isolate the effect of the AT alone.

3.3 Interaction Conditions

We evaluate agents through five interaction conditions: standard computer use, VoiceOver, Zoom at $2\times$ magnification, Zoom at $5\times$ magnification, and Voice Control. Screen readers and magnifiers are commonly used by people who are blind or low vision, while voice control is used by people with motor disabilities, who may have difficulty operating a mouse and keyboard. These interaction modalities change the action and observation space for interacting with the desktop en-

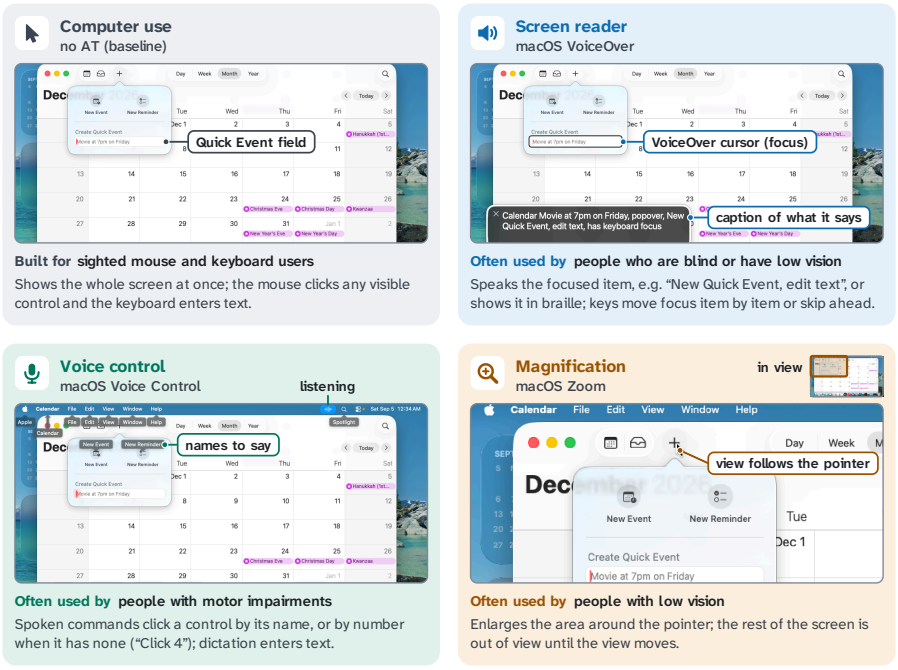

Figure 2: **The same step of a Calendar task under four interaction conditions.** With standard computer use, the full screen is visible and the user interacts with the mouse and keyboard. VoiceOver moves focus from one interface element to another and reads the focused element aloud. Voice Control lets the user select controls by saying their names or assigned numbers. Zoom enlarges part of the screen and moves the enlarged view with the pointer. The small overview at the top right of the Zoom panel shows which part of the full screen is currently visible.

vironment while operating on the same desktop state space. For each condition, we define the information returned to the agent and the actions it can issue through the corresponding interface. In comparison, standard computer use represents conventional mouse-and-keyboard, visual desktop interaction. Figure 2 illustrates how the same application appears under each of these conditions; the Supplementary Materials include additional video demonstrations.

Computer use baseline. The agent observes screenshots of the full display and acts with conventional mouse and keyboard controls, including clicks, drags, scrolling, key combinations, and text entry. This condition represents the visual interaction used in many existing computer-use agent evaluations and provides a reference for performance through the assistive technologies.

Screen reader. In the VoiceOver (macOS's screen reader) condition, the agent receives no screenshot or pointer access. It observes VoiceOver's spoken feedback and current-item text and acts through VoiceOver commands. These commands move the VoiceOver cursor through interface elements, enter and leave groups, navigate menus and structured web content, activate controls, and edit text. The agent must therefore track its position in the interface from VoiceOver feedback rather than from a visual view of the desktop.

Magnification. macOS Zoom displays a magnified region of the desktop and follows pointer and keyboard focus. The agent observes the rendered magnified view, pans to reveal other regions, and clicks or drags at coordinates relative to the returned view. Keyboard navigation and text entry remain available. Magnification preferences vary among low-vision users (WebAIM, 2018); we evaluate $2\times$ and $5\times$ as two fixed levels with different fields of view. It's worth noting that macOS Zoom magnification can go as high as $40\times$.

Voice Control. macOS Voice Control lets users operate the interface by speaking commands, referring to displayed targets on the screen. The three main available overlays are 'Item Numbers', which displays numbered marks,'Item Names' which displays the item text, and 'Show Grid' which displays a square grid with assigned numbers for clicking locations on the screen. Whenever this

overlay is displayed, the user is able to verbally refer to the content in the tags. In order to operate Voice Control, the agent observes screenshots of the overlay, and issues action using synthesized speech routed to Voice Control via a virtual cable.

3.4 BENCHMARK TASKS

Task selection and adaptation. AccessWorld draws tasks from macOSWorld (Yang et al., 2025) and MacArena (Muryn et al., 2026). We select tasks whose initial state can be reproduced and whose outcome can be evaluated reliably in a common macOS environment, allowing us to evaluate the same task through different interaction interfaces. We apply a fixed set of criteria for removing or modifying tasks. Modifications are limited to environment-specific configuration and small corrections to task setup, instructions, or graders, without changing the requested outcome. The benchmark VM contains the applications and shared assets required by the retained tasks, while task-specific setup scripts prepare additional initial state when needed. Appendix B.1 describes the complete selection and adaptation criteria.

Dataset statistics. The final benchmark contains 124 tasks, with 82 from macOSWorld and 42 from MacArena. The tasks span 33 applications and system interfaces, and 19 tasks involve more than one application. Of the 124 tasks, 102 use task-specific setup scripts and all 124 have automated graders. The source benchmark categories cover productivity (26 tasks), system applications (24), system and interface tasks (15), file management (15), multi-application tasks (15), advanced application use (14), safety (11), and media (4). We report the original category labels and their source-specific distributions in Appendix B.4.

Interaction annotations. The source categories describe the broad purpose of a task, but not the interactions needed to complete it. We therefore annotate each task with one or more interaction types. We reviewed the task set, identified recurring forms of interaction, and used the resulting 14 types as a common multi-label annotation scheme. The types include UI and menu navigation, text entry and editing, file manipulation, import and export, search, application switching, spatial interaction, spreadsheet and table interaction, copying and pasting, and continuous control. Tasks have a median of three interaction labels (mean 3.32), with a range of one to nine. The most common are UI and menu navigation (68 tasks), text entry and editing (56), file manipulation (49), multi-item interaction (37), and import and export (34). These labels describe the interactions required to complete a task rather than its difficulty or accessibility.

Task validation and scoring. We manually validate every retained task in the benchmark VM, confirming that its initial state can be reproduced, that the requested outcome can be completed, and that the automated grader distinguishes completed from incomplete end states. During evaluation, an agent either stops voluntarily or reaches its action limit. It then reports whether it believes the task is `completed`, `not_completed`, or `uncertain`, based on the interaction completed so far. We record this report separately from task success. The environment then runs the task's deterministic grader, which returns a binary completion score from the final computer state. The same grader is used for a task across all interaction conditions. Appendix D.2 gives the complete reporting and scoring protocol.

4 BENCHMARK RESULTS

4.1 RESULTS

Table 1 reports task completion across GPT-5.6 Sol, GPT-5.6 Luna, Qwen3.8-27B, GLM 5.3 Flash, and Claude Sonnet 5 for all five interaction conditions. Model rankings change substantially across interfaces: GPT-5.6 Sol performs best under CUA, Screen Reader, and Zoom $5\times$, while GLM 5.3 Flash achieves the highest accuracy under Zoom $2\times$ and Voice Control. This suggests that AT interaction modalities each demand a different area of capabilities needed for computer use.

The results show that assistive technology interaction does not impose a uniform performance penalty. Rather, Zoom $2\times$ is particularly competitive with direct computer use under the condition-dependent action budgets: GPT-5.6 Sol improves from 68.5% to 75.8%, GLM 5.3 Flash from 55.6% to 77.4%, and Claude Sonnet 5 from 56.5% to 68.5%. These results are in contrast to prior work (Gubbi Mohanbabu et al., 2026), where magnifier conditions ($1.5\times$) experienced a drastic de-

Table 1: **AccessWorld results by interaction condition and model.** Accuracy is the percentage of tasks that pass the automated grader. Claim accuracy is the percentage of tasks the models think that pass the automated grader. Actions and token counts are averaged across tasks.

Condition	Model	Accuracy ↑	Claim Acc.	Actions	Input (k)	Output (k)
CUA	GPT-5.6 Sol	**68.5**	**71.0**	**10.1**	**168.4**	**0.8**
	GPT-5.6 Luna	60.5	62.9	10.3	173.6	0.9
	Qwen3.8-27B	54.0	55.6	11.4	294.6	5.5
	GLM 5.3 Flash	55.6	60.5	10.8	281.9	3.3
	Claude Sonnet 5	56.5	60.5	11.2	238.5	2.1
Screen Reader	GPT-5.6 Sol	**66.1**	**75.0**	**53.5**	**1448.0**	**3.0**
	GPT-5.6 Luna	41.1	50.8	66.0	1961.7	3.9
	Qwen3.8-27B	37.1	41.9	75.1	3360.2	16.7
	GLM 5.3 Flash	42.7	46.0	75.1	2655.3	12.6
	Claude Sonnet 5	31.5	36.3	74.0	2601.0	9.4
Zoom (2×)	GPT-5.6 Sol	75.8	**82.3**	16.2	**418.8**	**2.2**
	GPT-5.6 Luna	50.8	56.5	20.5	595.9	3.3
	Qwen3.8-27B	54.0	58.1	**14.8**	1211.6	18.0
	GLM 5.3 Flash	**77.4**	81.5	17.3	1273.8	17.6
	Claude Sonnet 5	68.5	69.4	19.7	795.0	6.1
Zoom (5×)	GPT-5.6 Sol	**67.7**	**77.4**	**18.8**	**553.2**	**2.7**
	GPT-5.6 Luna	29.8	40.3	23.9	764.8	3.8
	Qwen3.8-27B	52.4	58.9	19.8	2991.0	25.8
	GLM 5.3 Flash	66.9	73.4	19.2	2384.9	22.5
	Claude Sonnet 5	46.0	48.4	21.2	1230.2	9.9
Voice Control	GPT-5.6 Sol	31.5	35.5	24.1	218.4	**2.3**
	GPT-5.6 Luna	29.8	36.3	24.5	**189.7**	2.7
	Qwen3.8-27B	30.6	34.7	24.9	558.5	15.9
	GLM 5.3 Flash	**38.7**	**45.2**	**24.0**	592.6	18.2
	Claude Sonnet 5	27.4	29.0	26.0	220.2	7.0

cline in accuracy. Increasing magnification to $5\times$ reduces accuracy for all models relative to $2\times$, including a 22.5-point decrease for Claude Sonnet 5 (68.5% to 46.0%). Moderate magnification therefore does not appear intrinsically difficult for current agents; rather, performance deteriorates as the observation space becomes more restricted.

Screen Reader produces much larger differences between models. GPT-5.6 Sol retains nearly its full CUA accuracy, decreasing only from 68.5% to 66.1%, whereas GPT-5.6 Luna falls from 60.5% to 41.1% and Claude Sonnet 5 from 56.5% to 31.5%. Successful screen reader interaction is therefore possible at task accuracy competitive with CUAs. Screen Reader also requires substantially longer interaction trajectories: GPT-5.6 Sol averages 53.5 actions and 1.45M input tokens, compared with 10.1 actions and 168k tokens under CUA.

Voice Control is consistently difficult, with accuracy ranging only from 27.4% to 38.7% across models. Unlike screen reader or high magnification, the central challenge is not simply recovering information from a restricted observation space. The uniformly low accuracy therefore suggests that current agents still struggle at the action level. The agent may send invalid or misdirected commands due to the unfamiliar interaction mechanism.

4.2 Efficiency and Interaction Cost

We measure interaction with charged actions and model cost with cumulative input (including cache reads) and output tokens.

VoiceOver can preserve task success while requiring substantially more interaction. GPT-5.6 Sol reaches 66.1% accuracy through VoiceOver and 68.5% under standard computer use, while its mean action count increases from 10.1 to 53.5 and its cumulative input from 168.4k to 1.45M tokens.

When we restrict each model to tasks completed successfully under both conditions, VoiceOver uses 4.7–7.1× as many actions and 5.7–9.0× as much cumulative input.

The accumulated input comes from both longer interaction sequences and more context per model call. Across models, VoiceOver makes 3.4–4.5× as many model API calls as standard computer use, while cumulative input per recorded API call is 2.1–2.9× higher. VoiceOver therefore increases cost through both the number of model–tool turns and the amount of accumulated context carried through those turns.

Magnification changes the interaction loop differently. Averaged across models, moving from 2× to 5× increases observation calls by 61% and model API calls by 38%, while cumulative input increases by 84%. Over the same comparison, average task accuracy falls from 65.3% to 52.6%. The smaller field of view at higher magnification is therefore accompanied by substantially more inspection without a corresponding improvement in task success.

Voice Control is constrained more strongly by the action sequence than by model input. Agents use an average of 24.7 actions under its 30-action budget, and 395 of the 424 failed episodes reach the configured budget. Its cumulative input remains far below VoiceOver and both Zoom conditions. The AT conditions therefore place pressure on different parts of the agent loop: VoiceOver accumulates long interaction histories, higher magnification requires more repeated observation, and Voice Control exhausts the available actions before reaching the requested state.

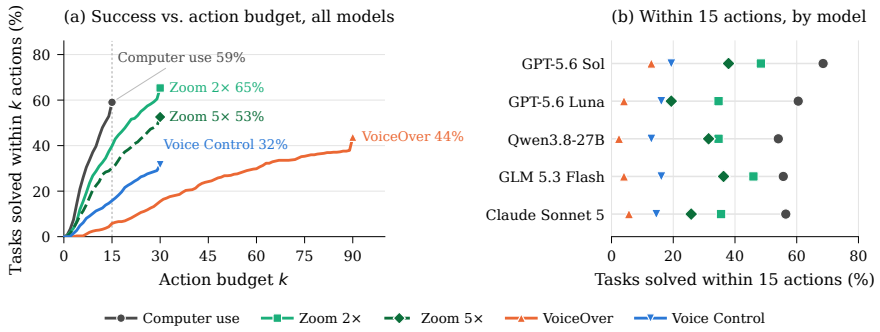

Figure 3: **Task Success Across Action Budgets.** (a) Percentage of tasks solved within k charged actions, averaged over the five models; each curve ends at its condition's action limit. (b) Success within 15 actions (the CUA limit) by model. Every AT trails standard CUA at a matched budget.

Success under a matched budget. Because the action limits differ across conditions, Figure 3 plots the share of tasks solved within k actions. Standard computer use is budget-limited: 93% of its failed episodes (235 of 254) reach the 15-action limit, and its curve is still rising there. Counting only tasks solved within 15 actions, the computer-use baseline reaches 59.0%, against 39.8% for Zoom 2×, 30.2% for Zoom 5×, 15.8% for Voice Control, and 5.8% for VoiceOver; the ordering holds for every model (Figure 3b). On tasks solved under both conditions, Zoom 2× uses 1.5–1.8× as many actions and 2.2–4.0× as much cumulative input as standard computer use. Magnified interaction can therefore match standard computer use in final success, but only given more actions; the gap between the two comparisons is what the AT costs.

4.3 Performance by Interaction Type

We use the annotations from Section 3.4 to group tasks by the interactions required for completion. Figure 4 reports the change in accuracy relative to standard computer use, averaged across the five models. Tasks can have multiple interaction labels, so the groups overlap.

The improvement under moderate magnification spans most interaction types. Among the 14 interaction types, 2× Zoom exceeds standard computer-use accuracy for 13. The largest increases occur for programming and code editing (+18.3 points), cross-application transfer (+15.7), and text entry and editing (+14.6). At 5×, this broad advantage largely disappears, with performance at or below standard computer use for 10 of the 14 interaction types.

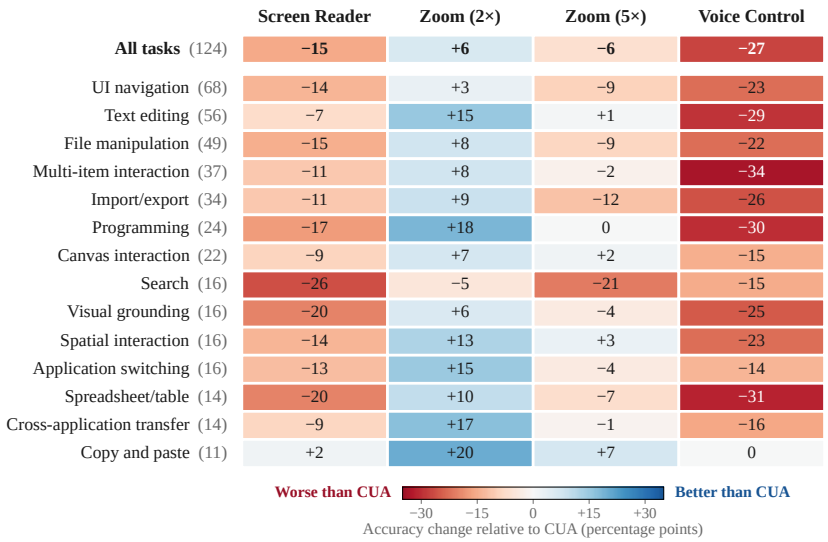

Figure 4: **Accuracy change by interaction type.** Difference in task accuracy between each AT condition and standard computer use, averaged across the five models. Tasks may have multiple interaction labels. Positive values indicate higher accuracy than standard computer use.

VoiceOver and Voice Control have different loss profiles. VoiceOver has its largest reductions on search (-23.8 points), visual grounding (-21.2), and programming and code editing (-20.0). Voice Control drops most on multi-item interaction (-33.5), spreadsheet / table (-31.4), programming (-30.0), and text editing (-29.3).

The capabilities of an agent are therefore largely conditional on the interaction interface used rather than model alone. These tradeoffs broadly align with the demands imposed by each interface, with larger losses on visually grounded tasks under VoiceOver and on tasks requiring precise text entry under Voice Control, such as programming and text editing.

4.4 Self Report and Grounding

Table 2: **Precision of self-reported completion.** Percentage of `completed` claims that pass the grader, with the false positive rate (share of failed tasks claimed as completed) in parentheses. Appendix E reports recall and F1.

Model	CUA	Screen Reader	Zoom (2×)	Zoom (5×)	Voice Control
GPT-5.6 Sol	90.9 (20.5)	87.1 (28.6)	88.2 (40.0)	84.4 (37.5)	88.6 (5.9)
GPT-5.6 Luna	**94.9** (8.2)	79.4 (17.8)	85.7 (16.4)	72.0 (16.1)	80.0 (10.3)
Qwen3.8-27B	92.8 (8.8)	88.5 (7.7)	90.3 (12.3)	87.7 (15.3)	88.4 (5.8)
GLM 5.3 Flash	90.7 (12.7)	**89.5** (8.5)	94.1 (21.4)	90.1 (22.0)	83.9 (11.8)
Claude Sonnet 5	92.0 (11.1)	84.4 (8.2)	**95.3** (10.3)	**91.7** (7.5)	**91.7** (3.3)

Table 2 compares the agents reported success with the grader. Agents rarely fail to claim tasks they successfully complete: recall is at least 94.1% for every model and condition. False completion claims are a more substantial source of disagreement. Claim precision ranges from 72.0% to 95.3% across the evaluated configurations. For instance, GPT-5.6 Luna claims nearly all of its successes at $5\times$ Zoom (97.3% recall), but only 72.0% of its completion claims pass the grader. Thus, recognizing success does not imply that an agent can reliably distinguish success from a failed attempt under the same interface.

Across models, completion claims are generally less precise under assistive technology than under standard computer use. Screen reader precision is lower than CUA precision for all five models, and increasing Zoom from $2\times$ to $5\times$ lowers precision for all five. The latter pattern is especially

pronounced for GPT-5.6 Luna, whose precision falls from 85.7% to 72.0%. Voice Control does not follow a single pattern relative to CUA: its precision is lower for four models but nearly equal for GPT-5.6 Sol. These differences suggest that agents' ability to verify completion may depend on both the action and observation space, where an unfamiliar action space such as Voice Control may damage ability to reliably verify completion, and the constrained observation spaces of VoiceOver and Zoom make it harder to inspect the final state before reporting success.

5 Discussion

Each AT necessitates a different improvement in agent capability. For VoiceOver, agents take several more model–tool turns and carry more context per turn, so better VoiceOver agents need shorter interaction sequences that still retain navigation state, through structural navigation commands, a compact representation of the current focus, and reusable navigation skills. The magnification results motivate agents that change scale as needed rather than operate at one fixed level. Voice Control failures mostly exhaust the spoken-action budget on repeated editing, selection, or manipulation; reusable skills for these operations could help if they still execute through Voice Control.

Finally, the results of self-reporting motivate an explicit verification stage. Verifying through an AT can require actions distinct from execution, such as revisiting a control through VoiceOver or moving the Zoom viewport to inspect the final state; reserving budget for these checks, or training a condition-aware verification policy, may reduce false completion claims without changing the task-solving policy.

6 Limitations and Future Work

Human evaluation. In its current state, AccessWorld is an agent-only benchmark. There are many significant underexplored areas that remain, such as task accessibility by humans, realistic interaction, and action count. As a result, agent failure and app inaccessibility are confounded, agent pathways may not match those of users, and expected action count is still unknown. The most immediate follow up on this work would be a human study of broad scope across native users of macOS accessibility features, recording what combination of ATs and preference of apps they naturally use.

AT coverage and expansion. There are many other available ATs outside of the ones used in this paper, such as keyboard control and eye tracking. Among the remaining tools native to macOS, Switch Control is a unique outlier in that its interaction is largely incompatible with current agent control loops. Switch Control users rely on timed selections of the item focus and scanning cursor, requiring immediate, real time reactions to operate. We believe that this alone could be a considerable future research direction to explore for AT agents.

Adaptive and hybrid AT agents. Our current experiments reset agents between tasks and restrict them to the interface assigned to each condition. OpenClaw can also support persistent interaction history, which could allow future agents to learn from user action traces and adapt to a person's AT workflow over time. Deployed agents may also combine AT interaction with native system tools, application commands, or automation APIs. AccessWorld provides the task setup, automated evaluation, and AT interfaces needed to study when such capabilities improve performance while preserving an interaction channel that AT users can follow and control.

7 Conclusion

Standard computer-use agents are designed and evaluated around mouse and keyboard interaction, while many people with disabilities rely on assistive technologies. We introduced AccessWorld, a macOS benchmark for evaluating agents through VoiceOver, Zoom, and Voice Control across 124 tasks. Across five frontier models, assistive technology changes both task success and model rankings: Zoom and VoiceOver can approach standard computer use, while Voice Control remains challenging. Limited observations under assistive technologies also make task completion harder to verify. These results show that performance under standard computer use does not reliably characterize an agent's ability to operate through assistive technologies. AccessWorld provides a foundation for evaluating CUA across the interfaces people with disabilities use every day.

AI USE STATEMENT

We used generative AI tools to proofread author-written text and suggest edits for clarity and readability, and to provide feedback on drafts of this paper. All research ideas, task curation, experiments, analyses, results, and conclusions were developed by the authors, who reviewed and revised all AI-assisted edits and take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Appendix A describes the virtual machine image, accessibility settings, orchestration, and logging; Appendix B.1 gives the task sampling, exclusion, and adaptation criteria; Appendix C lists the complete tool set of every condition; and Appendix D gives the harness configuration, action limits, model identifiers, inference settings, and error-handling protocol. We will release the environment-construction instructions, condition configurations, task files, orchestration code, A11y-tools plugin, and per-episode run artifacts (trajectories, claims, grader results, and token reports) used to compute every reported number.

ETHICS STATEMENT

AccessWorld involves no human participants; all tasks are inherited from public benchmarks and run in isolated virtual machines. Agents that operate assistive technologies could help people with disabilities, but they could also act on a user's behalf in ways that are hard to follow or verify through the same AT, and our results show that agents overstate completion more often under ATs. We report these failure modes so that deployed systems keep users in control, and we do not claim that agent performance reflects the experience or effort of AT users.

REFERENCES

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. Windows Agent Arena: Evaluating multi-modal OS agents at scale. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4874–4910. PMLR, 2025. URL https://proceedings.mlr.press/v267/bonatti25a.html.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Ruslan Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Yikun Fu, Bowen Fu, Zhenyu Wu, Shuang Cheng, Xiaowei Sun, Bowen Yang, Zehao Li, Yibo Zhao, Zichen Ding, Zhoumianze Liu, Shijie Wang, Biqing Qi, and Bowen Zhou. MacAgentBench: Benchmarking AI agents on real-world macOS desktop, 2026. URL https://arxiv.org/abs/2606.22557.

Ananya Gubbi Mohanbabu, Rosiana Natalie, Brandon Kim, Anhong Guo, and Amy Pavel. A11y-CUA dataset: Characterizing the accessibility gap in computer use agents. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2026. doi: 10.1145/3772318.3791896.

Satwik Ram Kodandaram, Utku Uckun, Xiaojun Bi, I. V. Ramakrishnan, and Vikas Ashok. Enabling Uniform Computer Interaction Experience for Blind Users through Large Language Models, July 2024. URL http://arxiv.org/abs/2407.19537. arXiv:2407.19537 [cs].

Satwik Ram Kodandaram, Monalika Padma Reddy, Xiaojun Bi, Jiawei Zhou, I. V. Ramakrishnan, and Vikas Ashok. Are we there yet? assessing computer-use agents for blind users' accessible interaction with desktop applications, 2026. URL https://arxiv.org/abs/2609.00524. Accepted to EMNLP 2026 (Main Conference).

Jinchao Li, Yunxin Li, Chenrui Zhao, Zhenran Xu, Baotian Hu, and Min Zhang. Windowsworld: A process-centric benchmark of autonomous GUI agents in professional cross-application environments. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 15262–15280. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.findings-acl.750. URL https://aclanthology.org/2026.findings-acl.750/.

Wanli Li, Bowen Zhou, Yunyao Yu, Zhou Xu, Yifan Yang, Dongsheng Li, and Caihua Shan. Weavebench: A long-horizon, real-world benchmark for computer-use agents with hybrid interfaces. 2026b. URL https://arxiv.org/abs/2606.09426.

Victor Muryn, Maksym Shamrai, Sofiia Mazepa, and Yehor Khodysko. MacArena: Benchmarking computer use agents on an online macOS environment, 2026. URL https://arxiv.org/abs/2606.06560. Accepted to the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Mehrab Tanjim, Nesreen K. Ahmed, Puneet Mathur, Seunghyun Yoon, Lina Yao, Branislav Kveton, Jihyung Kil, Thien Huu Nguyen, Trung Bui, Tianyi Zhou, Ryan A. Rossi, and Franck Dernoncourt. GUI Agents: A Survey, September 2025. URL http://arxiv.org/abs/2412.13501. arXiv:2412.13501 [cs].

Yi-Hao Peng, Dingzeyu Li, Jeffrey P. Bigham, and Amy Pavel. Morae: Proactively pausing UI agents for user choices. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2025. doi: 10.1145/3746059.3747797.

Maryam Taeb, Amanda Swearngin, Eldon Schoop, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. AXNav: Replaying accessibility tests from natural language. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2024. doi: 10.1145/3613904.3642777.

W3C Web Accessibility Initiative. Tools and techniques: How people with disabilities use the web, 2024. URL https://www.w3.org/WAI/people-use-web/tools-techniques/. Updated June 25, 2024.

WebAIM. Survey of users with low vision #2 results, October 2018. URL https://webaim.org/projects/lowvisionsurvey2/. Accessed: 2026-09-09.

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. doi: 10.52202/079017-1650.

Pei Yang, Hai Ci, and Mike Zheng Shou. macOSWorld: A multilingual interactive benchmark for GUI agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2506.04135.

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. Osworld 2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. URL https://arxiv.org/abs/2606.29537.

Mingyuan Zhong, Xia Chen, Davin Win Kyi, Chen Li, James Fogarty, and Jacob O Wobbrock. Taskaudit: Detecting functiona1 1ity errors in mobile apps via agentic task execution. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2026.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

A Environment and Infrastructure

A.1 macOS Virtual Machine

AccessWorld executes each task in an isolated macOS virtual machine (VM) hosted by UTM on Apple Silicon. The reference image uses the `aarch64` architecture, macOS 26.6 (Tahoe), 4 GB of memory, an 80 GB virtual disk, shared-network networking, and a fixed display resolution of 1920×1080. The exact UTM release used for the reported experiments was Version 4.7.5.

We construct a golden VM image containing the benchmark applications, the OpenClaw agent runtime, the AccessWorld orchestration worker, and the A11y-tools OpenClaw plugin.

A.2 Accessibility Settings

VoiceOver

- Welcome dialog: off
- Allow VoiceOver to be controlled with AppleScript: on
- Initial VoiceOver cursor position: keyboard-focused item
- Arrow-key Quick Nav: on
- Image Descriptions: on
- Speech voice and rate: Samantha, default rate
- Output device: Apple Virtual Sound Device

Zoom

- Keyboard shortcuts: on
- Zoom style: full screen
- Minimum/maximum zoom: both set to $2.0\times$ or $5.0\times$
- Zoomed image movement: keep pointer centered
- Smooth images: off
- Show zoomed image while screen sharing: on
- Follow keyboard focus: always
- Move screen image when focused item is off-center: center the focused item

Voice Control

- Overlay: item labels
- Microphone: VB-Cable virtual audio device

A.3 Task Initialization

Each task is represented by a JSON configuration. A representative macOSWorld task has the following form:

```
{
  "id": "<task UUID>",
  "task": {
    "en": "<task instruction>"
  },
  "pre_command": "<optional prescript command>",
  "grading_command": [
    ["<grader command>", 100]
  ]
}
```

MacArena uses the same core fields but stores `task` directly as a string and may include source, application, category, and upload-file metadata.

At the beginning of an episode, the guest worker copies the selected task configuration into the task log and executes its *prescript* (the `pre_command` or `prescript` field) inside the cloned VM.

Prescripts prepare the task-specific initial state by, for example, placing or removing files, resetting application data, opening an application or document, configuring a preference, or arranging the relevant window. Static assets referenced by inherited setup scripts are resolved against the corresponding directory under `benchmark_files/` in the mounted AccessWorld repository.

A.4 Run Orchestration and Logging

The host orchestrator performs the following sequence for every task: (1) clone the golden VM; (2) randomize the clone's MAC address; (3) publish a request to the guest worker; (4) start the clone; (5) execute the task prescript; (6) invoke the agent under the selected condition; (7) execute the task grader; (8) capture the final desktop state; (9) export logs to the shared repository; and (10) stop and delete the clone. The guest refuses to run prescripts, agents, or graders unless it is executing in the expected guest account, providing a guard against accidental host-side execution.

Logs are organized by run and task UUID. `task.json` records the task definition, `prompt.txt` the complete prompt, and `openclaw.filtered_config.json` the redacted runtime configuration. The transcript records model messages, tool calls, and tool results. `tool_calls.jsonl` records each call's sequence number, tool name, arguments, call ID, timestamp, status, source, and error when present. `events.jsonl` records a sequence number, timestamp, event name, and event-specific fields for each runner event. `summary.json` records the task, model, condition, duration, grade, final claim, budget counts, and terminal error. Separate files record prescript, OpenClaw, and grader output. A final desktop screenshot is captured after grading.

B Benchmark Tasks

B.1 Source Task Collections

AccessWorld derives its task pool from macOSWorld (Yang et al., 2025) and the 49 macOS-native tasks introduced by MacArena (Muryn et al., 2026). We randomly sampled 100 macOSWorld tasks and included all 49 MacArena tasks. We then applied the feasibility and validity criteria described below. This procedure produced a final benchmark of 124 tasks: 82 from macOSWorld and 42 from MacArena.

B.2 Task Selection and Exclusion Criteria

We excluded a candidate when at least one of the following conditions held:

1. its grader returned success regardless of the final environment state;
2. its required initial or final state conflicted with the benchmark's fixed assistive-technology configuration;
3. it was an exact copy of another selected task;
4. it was intended to evaluate functionality not included in AccessWorld;
5. it required a multilingual instruction or multilingual grader outside the English-only evaluation protocol;
6. the task or its target application crashed reproducibly during setup or execution;
7. its prescript or grading script contained malformed or nonfunctional AppleScript; or
8. the requested state was not achievable in the reference environment.

All retained tasks were checked for reproducible setup, feasibility, and a state-sensitive final grader. The release will include the retained UUIDs and the curation manifest so that the sampling and filtering decisions can be reconstructed without reproducing the manual validation process.

B.3 Instruction and Grader Adaptation

We modified a retained task only when one of the following criteria applied:

Source	Category	Tasks	Source	Category	Tasks
MacArena	Advanced Apps	8	macOSWorld	Advanced	6
MacArena	File Management	6	macOSWorld	File Management	9
MacArena	Productivity	13	macOSWorld	Media	4
MacArena	System and Interface	5	macOSWorld	Multi-Apps	15
MacArena	System Apps	10	macOSWorld	Productivity	13
–	–	–	macOSWorld	Safety	11
–	–	–	macOSWorld	System and Interface	10
–	–	–	macOSWorld	System Apps	14
MacArena	**Subtotal**	**42**	macOSWorld	**Subtotal**	**82**
				Total	**124**

Table 3: Distribution of AccessWorld tasks by source collection and inherited category.

1. a setup or grading command referred to the source guest account `ec2-user`, which was replaced with `testpanda`;
2. a setup or grading command referred to a source benchmark-files directory, which was rerouted to the corresponding directory under `AccessWorld/benchmark_files`;
3. a command used the source administrator password which was replaced with that of AccessWorld's image;
4. a small error prevented the prescript from producing its intended initial state;
5. the task instruction contained a typographical error; or
6. a small error prevented the grader from testing its intended final state.

The source task goal and grader structure were otherwise preserved. These changes did not alter the requested end state.

Each grader is a shell command that interrogates the live macOS state or a produced artifact and emits a line equal to `True` on success. Following the source benchmarks, AccessWorld only evaluates binary task score for the benchmark score.

B.4 Task Distribution

Table 3 reports the retained tasks by source and source category. Category labels are inherited from the source repositories.

C Interaction Interfaces

C.1 Computer Use Baseline

The computer-use baseline uses the following complete tool set. Pointer coordinates are absolute desktop coordinates; providers that use a normalized coordinate grid are converted to desktop pixels before execution.

C.2 Screen Reader

VoiceOver commands are dispatched through Guidepup. The non-visual condition exposes no screenshot or pointer tool. Most commands return the resulting spoken phrase and current-item text when they finish.

C.3 Magnification

The $2\times$ and $5\times$ conditions use the same tools:

Function	Operation
`cua_screenshot(label?,includeImage?)`	Capture the full display as a PNG.
`cua_click(x,y)`	Single left click.
`cua_double_click(x,y)`	Double left click.
`cua_right_click(x,y)`	Right click.
`cua_drag(startX,startY,endX,endY,durationSeconds?)`	Drag between two desktop coordinates.
`cua_scroll(verticalPixels,horizontalPixels?)`	Scroll at the current pointer location.
`cua_press_key(keyName)`	Press and release one key.
`cua_press_key_chord(keyName,modifiers)`	Press a key with one or more modifiers.
`cua_type_text(text,intervalMilliseconds?)`	Type text into the focused control.

Table 4: Computer-use baseline tools.

Group	Functions
Lifecycle	`voiceover_start`, `voiceover_stop`
Observation	`voiceover_last_spoken_phrase`, `voiceover_read_current_item`, `voiceover_wait_for_feedback`
Linear navigation	`voiceover_next_item`, `voiceover_previous_item`, `voiceover_repeat_action`, `voiceover_repeat_until_read`, `voiceover_go_to_top`, `voiceover_go_to_bottom`, `voiceover_jump_to_top_level`
Interaction	`voiceover_activate_item`, `voiceover_secondary_click_item`, `voiceover_double_click_item`, `voiceover_select_menu_or_list_item`, `voiceover_toggle_disclosure`, `voiceover_open_actions_menu`, `voiceover_sticky_mouse_toggle`, `voiceover_interact`, `voiceover_stop_interacting`
Web structure	`voiceover_next_heading`, `voiceover_previous_heading`, `voiceover_next_link`, `voiceover_previous_link`, `voiceover_next_form_control`, `voiceover_previous_form_control`, `voiceover_next_landmark`, `voiceover_previous_landmark`, `voiceover_next_frame`, `voiceover_previous_frame`, `voiceover_next_table`, `voiceover_previous_table`, `voiceover_next_list`, `voiceover_previous_list`, `voiceover_next_image`, `voiceover_previous_image`, `voiceover_open_web_item_rotor`, `voiceover_toggle_single_key_quick_nav`, `voiceover_read_link_url`, `voiceover_read_webpage_statistics`
Menus and search	`voiceover_open_commands_menu`, `voiceover_open_help_menu`, `voiceover_open_window_chooser`, `voiceover_open_application_chooser`, `voiceover_open_item_chooser`, `voiceover_find`, `voiceover_escape`, `voiceover_read_help_tag`, `voiceover_read_hint`
System locations	`voiceover_open_dock`, `voiceover_open_menu_bar`, `voiceover_open_control_center`, `voiceover_open_status_menus`, `voiceover_open_notification_center`, `voiceover_open_desktop`
Text and keyboard	`voiceover_type_text`, `voiceover_press_navigation_key`, `voiceover_press_text_edit_key`, `voiceover_read_line`, `voiceover_read_word`, `voiceover_read_sentence`, `voiceover_read_paragraph`, `voiceover_select_all_text_in_voiceover_cursor`, `voiceover_copy_last_spoken_phrase`, `voiceover_press_key_chord`
Spatial control	`voiceover_scroll_area`, `voiceover_move_window`, `voiceover_resize_window`

Table 5: Complete VoiceOver tool set.

Function	Operation
`zoom_take_screenshot(label?,includeImage?)`	Capture the rendered Zoom frame.
`zoom_pan_mouse(x,y,stepDelayMilliseconds?)`	Pan to an absolute desktop coordinate and return the new frame.
`zoom_click(x,y,stepDelayMilliseconds?)`	Click at a visible coordinate.
`zoom_double_click(x,y,stepDelayMilliseconds?)`	Double-click at a visible coordinate.
`zoom_right_click(x,y,stepDelayMilliseconds?)`	Right-click at a visible coordinate.
`zoom_drag(startX,startY,endX,endY)`	Drag between two visible coordinates.
`zoom_scroll(verticalPixels,horizontalPixels?)`	Scroll at the current pointer location.
`zoom_toggle()`	Toggle macOS Zoom.
`cua_press_key(keyName)`	Press and release one key.
`cua_press_key_chord(keyName,modifiers)`	Press a modified key.
`cua_type_text(text,intervalMilliseconds?)`	Type into the focused control.

Table 6: Magnification tools.

C.3.1 SCREENSHOTS

`cua_screenshot` uses `/usr/sbin/screencapture`. `zoom_take_screenshot` uses ScreenCaptureKit to capture the rendered full-screen Zoom frame.

C.3.2 PANNING AND POINTER CONTROL

`zoom_pan_mouse` takes absolute coordinates in the unzoomed desktop. `zoom_click` and `zoom_drag` functions take coordinates in the immediate magnified image.

C.4 VOICE CONTROL

Function	Operation
`voice_control_say(input,waitAfterMs?,includeImage?)`	Speak one command and return a screenshot of the resulting state.
`cua_screenshot(label?,includeImage?)`	Capture the current screen without speaking a command.

Table 7: Voice Control tools.

C.4.1 SPOKEN COMMANDS

Speech is generated by `/usr/bin/say` and routed through VB-Cable to the Voice Control microphone.

Command group	Examples
Applications	`Open<application>`, `Quit<application>`
Controls	`Click<item>`, `PressReturnkey`
Overlays	`Shownumbers`, `Click<number>`, `Showgrid`
Navigation	`Scrolldown`, `Movetobeginning`
Text	`Type<phrase>`, `Select<phrase>`, `Replace<phrase>with<phrase>`, `Undothat`

Table 8: Examples supplied with the Voice Control tool.

D OpenClaw Agent Harness

D.1 Agent Interaction Loop

Each episode runs one local OpenClaw session. The harness submits the composed prompt once; OpenClaw then alternates between model responses and local tool execution until the model returns a final response or the runner interrupts the session at a configured limit. The exported official-run configurations record OpenClaw version `2026.7.1-2`; the A11y-tools package version is `0.2.2`.

The temporary OpenClaw configuration makes the harness minimal in the following ways:

- `contextInjection` is set to `never`. OpenClaw therefore does not inject workspace context files such as `AGENTS.md`, `SOUL.md`, `IDENTITY.md`, `USER.md`, `BOOTSTRAP.md`, or `MEMORY.md`, including after context compaction.
- `skipBootstrap` is set to `true`, preventing OpenClaw from creating a bootstrap workspace for the episode.
- the general OpenClaw tool profile and inherited allow/deny additions are removed. All condition semantics are supplied by the condition file and enforced through the generated OpenClaw tool allowlist. Shell, file-system, Web, memory, skill, subagent, messaging, and scheduling tools are therefore unavailable unless explicitly named by that condition.
- each task receives a separate session key and temporary configuration. No conversation or agent memory is carried between tasks.
- the A11y-tools context engine retains at most the three most recent screenshots in model-facing context.

D.2 Final Claim

At the end of an episode, the model must return exactly one JSON object with two keys: `completion_status` and `final_response`. The status must be `completed`, `not_completed`, or `uncertain`; the response is one concise description of the outcome. If the runner interrupts execution at a non-time limit, it resumes the same session once with every tool denied and asks the model to make this claim using only the completed trajectory. The claim is parsed and stored separately from the task grade.

D.3 Models and Inference Settings

We evaluated five models through OpenClaw using their native provider routes. Table 9 reports the model identifiers and reasoning controls recorded during execution. OpenAI models used `medium`, Qwen and Claude used `high`, and GLM was run with reasoning `on`. Qwen and GLM were configured with a 1M-token context window, while other context and output limits followed their native OpenClaw model definitions. Claude used short-retention prompt caching.

Model	OpenClaw identifier	Provider route	Reasoning setting
GPT-5.6 Luna	`openai/gpt-5.6-luna`	OpenAI	`medium`
GPT-5.6 Sol	`openai/gpt-5.6-sol`	OpenAI	`medium`
GLM-5.3 Flash	`zai/glm-5.3-flash`	Z.ai	`on`
Qwen3.8-27B	`qwen/qwen3.8-27b`	DashScope	`high`
Claude Sonnet 5	`anthropic/claude-sonnet-5`	Anthropic	`high`

Table 9: Models and inference settings used in the official benchmark runs. Reasoning labels are provider-specific and should not be interpreted as equivalent inference budgets across model families.

D.4 Error handling

Infrastructure errors were identified through the runner's structured `error_stage` field, indicating potential environment interruption by model provider, VM, or grading infrastructure. Affected

tasks were rerun under the same model and condition from a fresh VM clone. Failed reruns remained in the provenance record but were not substituted for prior results. Four known, repeatedly reproducible exceptions were retained and explicitly flagged rather than repeatedly rerun, leaving infrastructure errors in 4 of 3,100 reported condition–task evaluations (0.13%).

E SELF-REPORTED COMPLETION

Table 10 reports precision, recall, F1, and false positive rate of the agents' completion claims for every condition and model.

Table 10: **Self-reported completion versus the grader, by condition and model.** Precision is the share of `completed` claims that pass the grader; recall is the share of graded successes that the agent claimed; FPR is the share of graded failures claimed as completed.

Condition	Model	Precision ↑	Recall ↑	F1 ↑	FPR ↓
CUA	GPT-5.6 Sol	90.9%	94.1%	92.5%	20.5%
	GPT-5.6 Luna	**94.9%**	**98.7%**	**96.7%**	**8.2%**
	Qwen3.8-27B	92.8%	95.5%	94.1%	8.8%
	GLM 5.3 Flash	90.7%	98.6%	94.4%	12.7%
	Claude Sonnet 5	92.0%	98.6%	95.2%	11.1%
Screen Reader	GPT-5.6 Sol	87.1%	98.8%	92.6%	28.6%
	GPT-5.6 Luna	79.4%	98.0%	87.7%	17.8%
	Qwen3.8-27B	88.5%	**100.0%**	**93.9%**	**7.7%**
	GLM 5.3 Flash	**89.5%**	96.2%	92.7%	8.5%
	Claude Sonnet 5	84.4%	97.4%	90.5%	8.2%
Zoom (2×)	GPT-5.6 Sol	88.2%	95.7%	91.8%	40.0%
	GPT-5.6 Luna	85.7%	95.2%	90.2%	16.4%
	Qwen3.8-27B	90.3%	97.0%	93.5%	12.3%
	GLM 5.3 Flash	94.1%	**99.0%**	**96.4%**	21.4%
	Claude Sonnet 5	**95.3%**	96.5%	95.9%	**10.3%**
Zoom (5×)	GPT-5.6 Sol	84.4%	96.4%	90.0%	37.5%
	GPT-5.6 Luna	72.0%	97.3%	82.8%	16.1%
	Qwen3.8-27B	87.7%	98.5%	92.8%	15.3%
	GLM 5.3 Flash	90.1%	**98.8%**	**94.3%**	22.0%
	Claude Sonnet 5	**91.7%**	96.5%	94.0%	**7.5%**
Voice Control	GPT-5.6 Sol	88.6%	**100.0%**	94.0%	5.9%
	GPT-5.6 Luna	80.0%	97.3%	87.8%	10.3%
	Qwen3.8-27B	88.4%	**100.0%**	93.8%	5.8%
	GLM 5.3 Flash	83.9%	97.9%	90.4%	11.8%
	Claude Sonnet 5	**91.7%**	97.1%	**94.3%**	**3.3%**