

000
001  **OS-MAP: HOW FAR CAN COMPUTER-USING**
002 **AGENTS GO IN BREADTH AND DEPTH?**
003
004

005 **Anonymous authors**
006 Paper under double-blind review
007
008
009

010 **ABSTRACT**
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031

032 Computer-using agents have shown strong potential to boost human productivity and enable new application forms across platforms. While recent advances have led to usable applications, existing benchmarks fail to account for the internal task heterogeneity and the corresponding agent capabilities, as well as their alignment with actual user demands—hindering both targeted capability development and the reliable transition of research progress into practical deployment. To bridge the gap, we present OS-MAP, a benchmark for daily computer-using automation that organizes its 416 realistic tasks across 15 applications along two key dimensions: a five-level taxonomy of automation and a generalization scope derived from a real-world user demand hierarchy. To enable fine-grained analysis of required capabilities and alignment with real-world scenarios, OS-MAP evaluates agents along two dimensions: automation level across a five-level taxonomy, and generalization scope across a demand hierarchy. This design captures varying levels of required agent autonomy and generalization, forming a performance-generalization evaluation matrix for structured and comprehensive assessment. Experiments show that even State-of-the-Art agents with VLM backbones struggle with higher-level tasks involving perception, reasoning, and coordination—highlighting the need for a deeper understanding of current strengths and limitations to drive the future progress in computer-using agents research and deployment. All code, environments, baselines, and data are publicly available at <https://anonymous.4open.science/r/OSMap-C2F5/>.
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053

1 **INTRODUCTION**

034 Computer-using agents, which can understand user intent and autonomously perform operations across digital environments, is driving the next transformation in human-computer interaction (Hu et al., 2024a;b). Powered by the extensive world knowledge, interaction capability, and tool-use abilities of Large Language Models and Vision Language Models, computer-using agents such as Operator (OpenAI, 2025), Claude 3.5 (Anthropic, 2024), UFO² (Zhang et al., 2025), and UI-TARS (Qin et al., 2025) can understand natural language instructions and interact directly with various applications in a human-like manner. Once a fixture of science fiction—like J.A.R.V.I.S. in *Iron Man*, seamlessly managing schedules, editing documents, shopping across websites, and automating routine computer tasks—such digital personal assistants are now becoming a tangible reality (Wu et al., 2024a). This transformation frees humans to focus on creative work, significantly boosting productivity and enabling new applications.

035 As research on computer-using agents continues to advance, an increasing number of models with strong functionalities (Qin et al., 2025; Bai et al., 2025; Xu et al., 2024a; Wu et al., 2024b) and agent systems (Jiang et al., 2025; Agashe et al., 2025; Zhang et al., 2025; Jia et al., 2024) are being proposed. Despite the rapid emergence of new methods, the open-ended semantics and diverse capability demands of computer-using tasks still hinder actual deployment. To bridge this research-to-practice gap, it is crucial to develop a principled benchmark that allows the community to quantify agent capabilities and identify specific failure points. However, existing benchmarks fall short of this goal. While spanning various platforms and scenarios, they treat tasks as flat collections, without decomposing task heterogeneity and required capabilities (Drouin et al., 2024; Xie et al., 2024; Bonatti et al., 2024; Rawles et al., 2024), making it difficult to perform fine-grained evaluation and differentiation. Moreover, task collections are typically organized around applications (Li et al.,

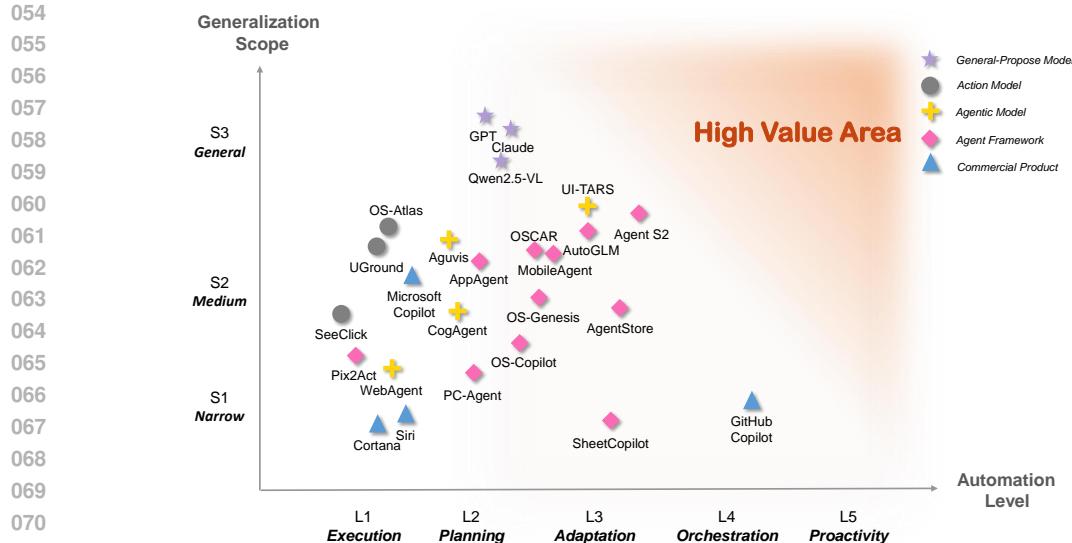


Figure 1: OS-MAP qualitative evaluation matrix, summarizing how different types of agents perform across two dimensions. General-purpose models show strong generalization, while scenario experts excel at specific tasks. Mainstream computer-using agents aim to balance both, yet still face major challenges. Agent positioning is based on reported performance, as detailed in Appendix A.

2024a; Xie et al., 2024) rather than aligned with the actual distribution of daily computer use, limiting the relevance of benchmark performance to real-world utility (Hu et al., 2024a).

To bridge these gaps, we present the OS-MAP benchmark that is grounded in dynamic desktop environments and structured along two key dimensions: **automation levels** and **generalization scopes**. First, we propose a five-level capability taxonomy based on degrees of autonomy, encompassing a wide range of computer-using tasks—from atomic execution and simple planning to disturbance adaptation, complex orchestration, and proactive behaviors. Second, we derive a real-world user demand hierarchy on daily computer-using scenarios and select representative tasks to ensure both high coverage and alignment with practical demands. Furthermore, we combine the two dimensions into a unified evaluation matrix (Figure 1), which highlights how general-purpose models, scenario experts, and mainstream computer-using agents differ in capability trade-offs between automation and generalization. The upper-right corner marks a high-value region—representing impactful yet unachieved applications—where no current agent demonstrates sufficient capability.

Across 416 tasks spanning 15 applications in OS-MAP, even the strongest existing computer-using agents achieve only an 11.5% overall success rate, with near-zero performance on higher-level tasks—falling far short of human performance. These findings underscore the importance of a principled evaluation framework. By offering both qualitative and quantitative insights into where and to what extent computer-using agents can assist humans, our framework supports comprehensive evaluation and provides a clear roadmap for future progress.

2 ENVIRONMENT

OS-MAP adopts and extends the OSWorld (Xie et al., 2024) infrastructure, which centers around a virtual machine (VM) and a host-side controller (VMC). This dynamic and executable environment offers fine-grained control, consistent reproducibility, flexible extensibility, and secure isolation, forming an ideal sandbox for evaluating computer-using agents in real-world scenarios.

2.1 TASK DEFINITION

In general, computing-using automation tasks are roughly modeled as partially observable Markov decision processes $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. At timestep t , the agent resides in the environment state $s_t \in \mathcal{S}$, but only receives a partial observation $o_t \in \mathcal{O}$ (e.g., the current screenshot). Based on o_t , the agent emits an action $a_t \in \mathcal{A}$ (e.g., a structured text click $(350, 600)$). The environment transitions to the next state $s_{t+1} = \mathcal{T}(s_t, a_t)$ via the transition function \mathcal{T} , which is governed by the underlying

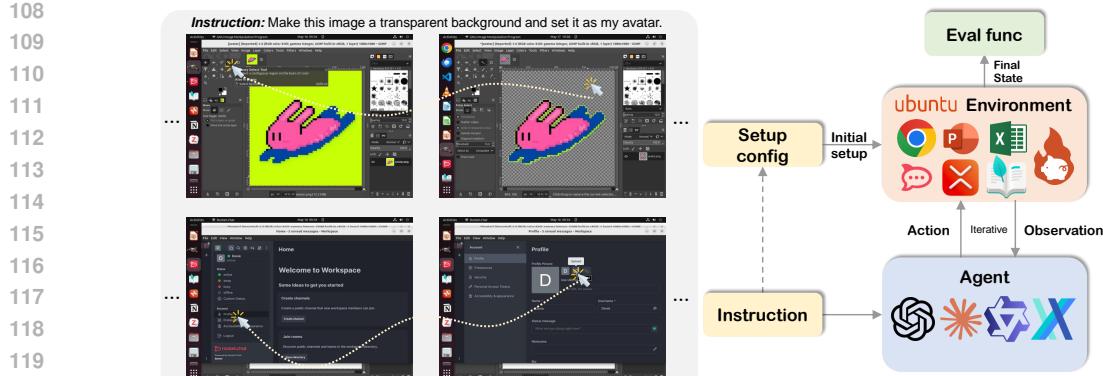


Figure 2: OS-MAP is built on an executable desktop environment designed for daily computer tasks, integrating a suite of applications and tools. It provides the infrastructure for reliable evaluation by handling task initialization and success verification. Agents interact autonomously via GUI operations, guided by instructions and screenshot perception.

software and OS logic, revealing a new observation o_{t+1} . This process continues iteratively until the agent actively issues a terminal action (*i.e.*, DONE or FAIL), or passively exceeds a predefined step limit. After termination, the system determines whether the task is successfully completed and provides a final outcome reward $r \in \mathcal{R} = \{0, 1\}$, without any intermediate process rewards.

2.2 ENVIRONMENT STRUCTURE

Task lifetime. Each task is specified by a JSON file defining initialization, instruction, and evaluation protocols. As shown in Figure 2, evaluation begins by restoring a designated VM snapshot and running lightweight setup routines. The agent then enters the interaction loop, receiving observations from and sending actions to the VM via the VMC. This loop continues until the agent terminates the episode, either actively or passively. The evaluator then compares the VM state to reference criteria and returns a binary reward. See Appendix B.1 for details.

Initialization and evaluation configuration. Task setup in OS-MAP combines VM snapshots with modular configuration functions, supporting scalable and flexible task creation. Standard initialization adopt reusable OSWorld functions (*e.g.*, file downloading, shell commands), while more complex setups—such as software installation or database configuration—are manually performed and captured as directly restorable snapshots. Evaluation integrates both state-based and action-based assessments to support tasks with varying levels of automation. Depending on the task, state evaluation may involve file comparison and system state inspection, or execution-based verification. See Appendix B.2 and B.3 for detailed initialization and evaluation modes.

Observation and action space. Recent computer-using agents research increasingly gravitates toward human-like interaction paradigms: raw pixel screenshots as observations and atomic keyboard-/mouse operations as actions. OS-MAP adopts this design, using raw screenshots as input—without accessibility trees or Set-of-Marks (Yang et al., 2023) annotations—for simplicity and broader applicability. The action space follows OSWorld’s 13 atomic operations and 3 meta-actions (*i.t.*, WAIT, FAIL, DONE). See Appendix B.4 and B.5 for detailed space descriptions with examples.

3 BENCHMARK

OS-MAP comprises 416 real-world computer-using automation tasks on 15 Ubuntu applications, spanning diverse everyday scenarios. Tasks are categorized along two orthogonal dimensions: **automation level**, capturing the degree of agent autonomy, and **generalization scope**, defined by a hierarchical demand taxonomy, measuring agents’ capability transferability. Together, they form a structured **evaluation matrix** (Fig. 1) supports systematic evaluation. The following sections introduces the automation levels (§3.1), generalization scope (§3.2), evaluation matrix (§3.3), task curation pipeline (§3.4), and benchmark statistics (§3.5).

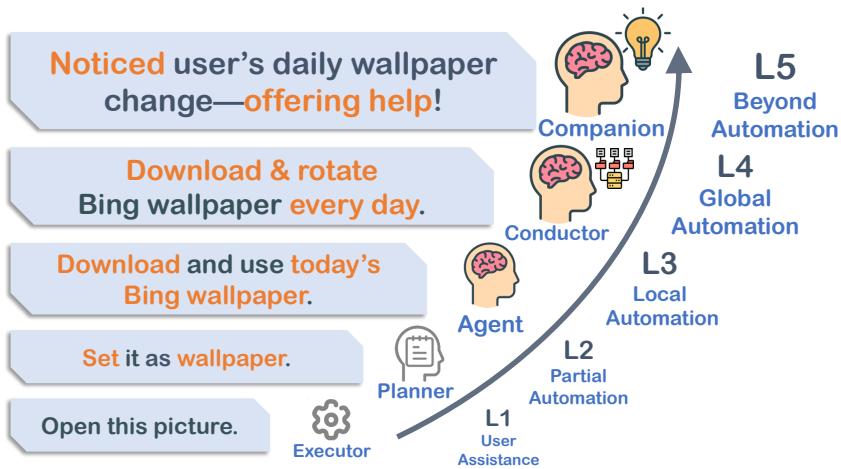


Figure 3: Automation levels demonstration on a specific task: rotating wallpapers daily. From the user’s perspective, achieving the same goal involves increasing agent responsibility and decreasing user involvement as automation level rises. Task executions become longer and more complex, reflecting the shifting division of labor between human and the agent.

3.1 AUTOMATION LEVELS

Real-world computer automation varies in task complexity, user involvement, and agent responsibility. To support consistent evaluation across these variations, we introduce a five-level automation taxonomy, inspired by SAE driving automation taxonomy (Committee, 2021) and grounded in the division of labor between humans and agents. Figure 3 illustrates how a concrete task manifests across all five interaction modes. Each level reflects a specific degree of autonomy in planning and execution, shaped by both task complexity and the expected user role.

L1: Reactive executor. The agent executes user-defined atomic operations (e.g., clicks, keystrokes) without making decisions. Task planning remains entirely user-driven. This stage primarily evaluates perceptual grounding and command-to-action mapping—capabilities that many grounding models (Cheng et al., 2024; Wu et al., 2024b; Gou et al., 2024) specifically target.

L2: Deterministic planner. The user specifies only the task goal, leaving the agent to autonomously plan and execute actions under ideal and predictable conditions. High-level task decomposition remains user-driven, and intervention is required when failures occur. This stage tests prior knowledge and basic planning, representing the operational level of most current agents and proprietary models.

L3: Adaptive agent. L3 emphasizes robustness in dynamic, noisy, and partially observable environments. Agents must adapt plans autonomously in response to unpredictable events or evolving interface states. While users still define high-level goals, they no longer need to monitor or intervene during execution. Only a small subset of agents specifically designed for adaptivity reach this level, demonstrating resilience and flexible subtask completion under real-world conditions.

L4: Global conductor. The agents take full responsibility for decomposing high-level goals and orchestrating complex workflows involving subgoals, cross-application context switching, and tool usage. Acting as autonomous top-level orchestrators, they coordinate entire tasks end-to-end, with users only issuing goals and verifying outcomes. As shown in our results, no current agent effectively handles this level, though emerging multi-agent approaches show promise.

L5: Proactive companion. L5 marks a shift from reactive execution to proactive collaboration. The agent continuously monitors context, anticipates user needs, and initiates helpful actions without explicit instructions. It learns from long-term interactions to provide personalized support as an intelligent digital companion. While still an underexplored concept, with only a few studies across different scenarios (Lu et al., 2024; Chen et al., 2025; Chaves and Gerosa, 2021; Liao et al., 2023), it holds significant promise for future applications. OS-MAP does not yet include L5 tasks and we leave them for future improvements.

216 3.2 GENERALIZATION SCOPE
217

218 While §3.1 focuses on structural organization via automation levels, this section turns to the content
219 dimension. Designing meaningful tasks for computer-using agents is challenging. Prior benchmarks
220 often rely on predefined applications sets (Xie et al., 2024; Bonatti et al., 2024; Li et al., 2024a;
221 Chai et al., 2025) or template-based generation (Rawles et al., 2024; Drouin et al., 2024). In
222 contrast, we take a demand-driven approach—identifying common daily use cases and deriving tasks
223 accordingly—to ensure realism, representativeness, and practical relevance.

224 **Demand hierarchy.** We define a three-level hierarchy: domains, scenarios, and representative tasks
225 with applications, guided by industry data (Tower, 2025) and public surveys (OECD, 2025). Starting
226 from the *State of Mobile* 2025 report, we adapt mobile usage statistics to the desktop setting by
227 excluding mobile-specific categories (e.g., payments) and adding desktop-relevant ones (e.g., office
228 work), forming six domains: work, study, life services, entertainment, creative production, and system
229 management. Figure 4 illustrates this demand hierarchy details. Scenarios are derived by aligning
230 app subcategories with activity metadata from OECD *ICT Access and Usage Database* (OECD,
231 2025). Tasks are then selected through expert review and LLM-assisted ideation, filtered by clarity,
232 reproducibility, and independence from real-world accounts or network-side effects.

233 **Generalization scope.** Anchored in this hierarchy, we define three scopes of generalization: S1
234 (Narrow), S2 (Domain-Level), and S3 (General) to characterize agents’ capability breadth across
235 diverse user demand. An S1 agent handles tasks within a single scenario (e.g., calendar management).
236 An S2 agent succeeds across multiple scenarios within a domain (e.g., document editing, emailing,
237 and scheduling in the work domain). An S3 agent demonstrates S2-level performance across most or
238 all six domains, effectively acting as a cross-domain generalist for daily computer-using assistance.

239 3.3 EVALUATION MATRIX
240

241 We further integrate the two orthogonal dimensions—automation levels (L1–L5) and generalization
242 scopes (S1–S3)—into a two-dimensional evaluation matrix, enabling a systematic assessment of both
243 the depth and breadth of agent capabilities, as presented in Figure 1. This depth–breadth perspective
244 aligns with the *performance–generality* framework proposed in earlier AGI research (Morris et al.,
245 2024; Zhang et al., 2024a). In the context of computer-using, **performance** denotes the extent to
246 which an agent can operate independently from human intervention within collaborative settings, as
247 reflected by the task complexity across the automation levels. **Generality** refers to the range of tasks
248 where the agent meets the performance threshold, anchored in its coverage of the demand hierarchy.

249 By decoupling performance and generality, the matrix provides a fine-grained evaluation of CUAs’
250 practical utility by revealing strengths and limitations and supporting clear comparisons across
251 systems with differing design priorities. The structure also scales naturally—new tasks or scenarios
252 can be added to underexplored regions without disrupting the overall framework. Most importantly,
253 it offers a clear developmental roadmap, guiding researchers and practitioners in setting progressive
254 goals along both dimensions toward building more capable and general-purpose agents.

255 3.4 TASK CURATION PROCESS
256

257 Each task in OS-MAP is created following a standardized six-step process grounded in the two-
258 dimensional organization framework: (1) task selection, (2) exploration and specification, (3) instruc-
259 tion and configuration, (4) reference state preparation, (5) evaluation setup, and (6) cross-validation.
260 Detailed descriptions of each step are provided in Appendix C.1. To ensure correctness and stability
261 in an open-ended environment, each stage of this process demands significant manual effort and
262 verification. Appendix C.2 presents the full design and refinement process of a representative L4
263 task. We also incorporate and adapt OSWorld tasks by mapping them to our difficulty levels and
264 generalization tiers. All included tasks undergo the same validation pipeline, while those with invalid
265 formats or ambiguous feasibility are excluded. Related details are discussed in Appendix C.3.

266 3.5 BENCHMARK STATISTICS
267

268 **Statistics.** Figure 4 presents the distribution of tasks in OS-MAP across the user-centered demand
269 hierarchy. Based on industry surveys (Tower, 2025; OECD, 2025), we define a three-level demand
270 hierarchy comprising 6 top-level needs, 18 sub-needs, and 45 concrete scenarios, spanning 15
271 representative applications and covering a broad range of daily computer-using situations. In total,

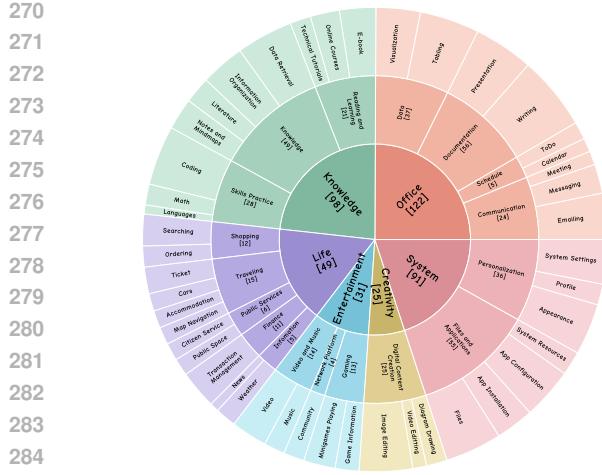


Figure 4: Task distribution on the demand hierarchy in OS-MAP benchmark.

Table 1: Statistics of OS-MAP.

Task Type	Statistics
Total Tasks	416 (100%)
- Single-App	283 (62.3%)
- Multi-App	154 (37.7%)
Automation Level	
- L1: Execution	25 (6.0%)
- L2: Planning	234 (56.3%)
- L3: Adaptability	115 (27.6%)
- L4: Orchestration	42 (10.1%)
Source	
- Authors	161 (38.7%)
- Labeled from OSWorld	255 (61.3%)
Avg. Words of Task Instructions	34.3
Avg. Steps	11.4

OS-MAP contains 416 tasks representative of their respective scenarios. Among them, 138 tasks are meticulously designed by the authors, while the remaining 296 are relabeled and filtered for ambiguity and redundancy from OSWorld (Xie et al., 2024) as Appendix C.3 describes. Table 1 provides more detailed statistics. Notably, 37.7% of tasks involve multi-app workflows, posing significant challenges to agents’ adaptation and orchestration capabilities.

Comparison with existing benchmarks. Table 2 compares OS-MAP with existing efforts across key dimensions. First, OS-MAP builds on an executable environment, inheriting the architecture, utility functions, and evaluation tools from OSWorld (Xie et al., 2024). This ensures controllability and flexible open-domain scalability. Second, we expand the number of applications and tasks, including a substantial portion of cross-application tasks, thereby enhancing the task diversity. Most importantly, we introduce a fine-grained evaluation framework based on both task difficulty levels and user demand hierarchy. These two axes are further integrated into a structured, two-dimensional evaluation matrix, enabling systematic, detailed comparisons and offering clear guidance for future development—an aspect largely overlooked by existing benchmarks.

Table 2: Comparison of different environments for benchmarking CUAs. The columns indicate: dynamic executable environment provided (Exec. Env.?), the ease of adding new tasks involving arbitrary applications in open domains (Scal. Env.), the number of applications or websites (#Apps/sites), the number of task instances and templates (if applicable) (# Inst. (# Temp.)), inclusion of cross-app tasks (Cross-app?), whether to provide evaluation based on task difficulty (Task Diff. Levels?), demand perspective (Demand Scope?), or a multi-dimensional structure (Struct. Eval.?).

Benchmark	Exec. Env.?	Scal. Env.?	# Apps/sites	# Inst. (# Temp.)	Cross-app?	Task Diff. Levels?	Demand Scope?	Struct. Eval.?
GAIA Mialon et al. (2023)	✗	-	-	466	✗	✓	✗	✗
MIND2WEB Deng et al. (2023)	✗	-	137	2350	✗	✗	✗	✗
WEB VOYAGER He et al. (2024)	✗	-	15	643	✗	✗	✗	✗
PIXELHELP Li et al. (2020)	✗	-	4	187	✗	✗	✗	✗
AITW Rawles et al. (2024)	✗	-	357+	30k	✗	✗	✗	✗
OMNIACT Kapoor et al. (2024)	✗	-	60+	9802	✗	✗	✗	✗
WEBSHOP Yao et al. (2022)	✓	✗	1	12k (1)	✗	✗	✗	✗
WEBARENA Zhou et al. (2023)	✓	✗	6	812 (241)	✗	✗	✗	✗
WORKARENA Drouin et al. (2024)	✓	✗	1	23k (29)	✗	✗	✗	✗
ANDROIDARENA Xing et al. (2024)	✓	✗	13	221	✓	✓	✗	✓
ANDROIDWORLD Rawles et al. (2024)	✓	✓	20	∞ (116)	✓	✗	✗	✗
ANDROIDAGENTARENA Chai et al. (2025)	✓	✓	21	201	✗	✓	✗	✗
OSWORLD Xie et al. (2024)	✓	✓	9	369	✓	✗	✗	✗
SPIDER2-V Cao et al. (2024)	✓	✓	20	494	✓	✗	✗	✗
WINDOWSAGENTARENA Bonatti et al. (2024)	✓	✓	11	154	✓	✗	✗	✗
THEAGENTCOMPANY Xu et al. (2024b)	✓	✗	6	175	✓	✗	✓	✗
SCIENCEBOARD Sun et al. (2025)	✓	✓	6	169	✓	✓	✗	✓
OS-MAP	✓	✓	15	416	✓	✓	✓	✓

324

4 EXPERIMENTS AND ANALYSIS

325

4.1 EXPERIMENTAL SETTINGS

326 **Agent types.** We construct three types of computer-using agents based on different types of state-of-
 327 the-art models: **(1) General baselines:** directly uses a general-purpose VLMs (GPT-4o (Hurst
 328 et al., 2024), Claude-3.7-Sonnet (Anthropic, 2025), Gemini-2.5-Pro (Team, 2025),
 329 Qwen2.5-VL-72B (Bai et al., 2025), InternVL3-8B (Zhu et al., 2025)) to perform each task
 330 end-to-end. **(2) GUI-specific model baseline:** executes tasks end-to-end using GUI-specialized
 331 VLMs (UI-TARS-72B (Qin et al., 2025)). **(3) Planning-Grounding:** to compensate for the impre-
 332 cise grounding abilities of general models, GPT-4o (Hurst et al., 2024) is used to conduct high-level
 333 plans, which are then refined by lightweight GUI action models (Aguvis-7B (Xu et al., 2024a),
 334 OS-ATLAS-Base-7B (Wu et al., 2024b), UGround-7B (Gou et al., 2024), GUI-Actor-7B (Wu
 335 et al., 2025)) for precise grounding.

336 **Agent settings.** All three agent types share a common decision-making and interaction pattern,
 337 along with similar prompting strategies. Specifically, the agent interacts with the environment under
 338 the guidance of a system prompt, which includes descriptions of the task goal, observation space,
 339 action space, and required output format. At each step, the agent generates an action based on the
 340 current screenshot and the three most recent rounds of interaction history. Detailed prompts and
 341 interaction protocols are provided in Appendix E.

342

4.2 RESULTS

343 We compare the performance of the above four computer-using agents types powered by different
 344 models on OS-MAP, as presented in Table 3. We summarize our key empirical results as follows:

345 **Computer-using agents remain far from practical deployment.** Despite recent advances, current
 346 agents exhibit consistently poor performance across all levels of automation, with many near zero,
 347 highlighting a substantial performance gap from human users. This suggests that existing models still
 348 struggle with core capabilities such as grounding.

349 **Agents' performance exhibits a stepwise decline across automation levels.** Among the evaluated
 350 models, UI-TARS-72B achieves the best balance of visual grounding, robust planning, and task
 351 generalization, significantly outperforming other competitors. It performs well when tasks include
 352 step-level guidance (L1) and maintains solid performance on basic planning (L2). However, its
 353 advantage drops markedly on environmental adaptation (L3) and multi-context orchestration (L4),
 354 suggesting that **adaptive reasoning and long-horizon planning remain key challenges**.

355 **Open-source models have achieved competitive end-to-end performance.** Although smaller in
 356 scale, open-source models fine-tuned on GUI-specific data (Bai et al., 2025) or trained in GUI-centric
 357 environments (Qin et al., 2025) demonstrate better performance than proprietary general-purpose
 358 models in end-to-end execution. Their superiority stems from targeted training in GUI contexts,
 359 which enhances planning stability and task adaptation in complex desktop environments.

360 **Tailored training and agentic setting yield better computing-using performance.** Compared
 361 to general models, GUI-specific models (Qin et al., 2025) and interleaved planning-grounding
 362 agents (Xu et al., 2024a; Wu et al., 2024b; Gou et al., 2024; Wu et al., 2025) gain a significant im-
 363 provement. The specially designed GUI training makes models familiar with computer environments,
 364 while the planning-grounding agents combine the world knowledge and strategic planning of general
 365 models with the precise perception and control ability of the GUI-oriented models.

366

4.3 ANALYSIS

367 To understand key challenges behind poor performance, this section analyzes representative failure
 368 cases to uncover core factors that lead to agent breakdowns. We highlight both general capability
 369 gaps observed across agents and level-specific bottlenecks tied to increasing automation levels. These
 370 insights shed light on where agents fall short and inform more targeted future improvements. See
 371 Appendix F for a more detailed case analysis with screenshots.

378 Table 3: Success rates of computer-using agents on OS-MAP. We present each agent backbone’s per-
 379 formance on tasks across different automation levels. Proprietary VLMs, and Open-Source VLMs
 380 are distinguished by color. In Planning-Grounding setting, GPT-4o is used as the planning model.
 381

Agent Type	Model	Success Rate (↑)				
		L1	L2	L3	L4	Overall
General Baselines	GPT-4o	12.0%	1.3%	1.7%	0.0%	1.9%
	Claude-3.7-Sonnet	0.0%	3.8%	0.0%	0.0%	2.1%
	Gemini-2.5-Pro	8.0%	10.6%	2.7%	2.4%	7.5%
	Qwen2.5-VL-72B	32.0%	7.9%	1.0%	0.0%	6.6%
	InternVL3-8B	8.0%	1.6%	1.0%	0.0%	1.6%
GUI-Specific Baseline	UI-TARS-72B	48.0%	14.0%	1.0%	0.0%	11.4%
Planning-Grounding	Aguvis-7B	4.0%	4.7%	1.8%	0.0%	3.4%
	OS-ATLAS-Base-7B	8.0%	6.4%	1.8%	0.0%	4.6%
	UGround-7B	16.0%	4.6%	1.8%	0.0%	4.0%
	GUI-Actor-7B	40.0%	15.1%	1.8%	0.0%	11.5%
Human Performance		96.0%	74.8%	65.2%	59.5%	71.9%

4.3.1 GENERAL FAILURES

Poor instruction following. This manifests as frequent violations of the required output format. A typical case is Claude-3.7-Sonnet issuing an OPEN_FILE_EXPLORER action when opening the file manager—despite the valid action space only contains atomic mouse and keyboard operations.

Severe hallucination. Due to limited perception and reasoning, agents often wrongly assume that previous actions have succeeded, and occasionally exhibit drastic hallucinations—e.g., mistaking the activities window for Chrome and attempting to search within it (Figure 13 in Appendix F).

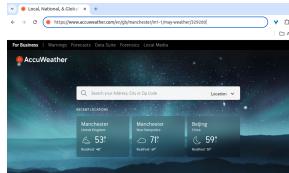


Figure 5: Agent prefers entering a URL instead of navigating websites.

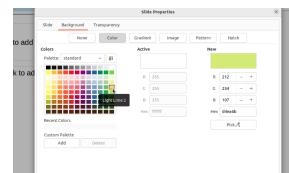


Figure 6: GUI action model fails in the grounding of the green block.

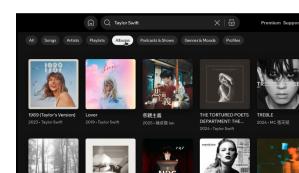


Figure 7: Searching for the album Taylor Swift instead of albums by Taylor Swift.

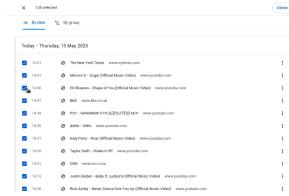


Figure 8: Agent is deleting all history, not just those from YouTube.

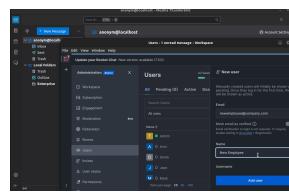


Figure 9: Agent starts filling out the form before clarifying information.

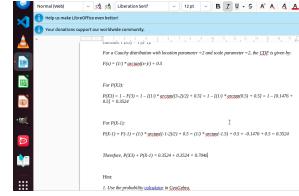


Figure 10: Calculating calculus internally instead of trying tools in the context.

Figure 11: Failure cases of each automation levels, reflecting bottlenecks in core capabilities.

4.3.2 LEVEL-WISE BOTTLENECKS

L1: execution. Proprietary models exhibit poor grounding capabilities, often preferring command-line operations or direct URL jumps (Figure 5). In contrast, GUI action models demonstrate more human-aligned interaction patterns, yet still struggle with locating non-textual elements (Figure 6).

L2: planning. Agents are prone to two common failure modes at this level: (1) **distraction by similar but incorrect options**—for example, searching for the album Taylor Swift instead of *all albums by Taylor Swift* (Figure 7); and (2) **neglecting specific task constraints**—e.g., deleting all browsing history instead of only entries related to YouTube (Figure 8).

432 **L3: adaptability.** Agents demonstrate basic proactive exploration (e.g., inspecting potential di-
 433 rectories before file operations) and reactive handling (e.g., closing unexpected pop-ups). However,
 434 they **struggle with fallback strategies under deviation**, such as failing to exit full-screen mode
 435 via hotkeys (Figure 14 in Appendix F), or activating theater mode before resizing, which hides the
 436 required controls (Figure 15). They also show **poor awareness of implicit task context** (Figure 16).
 437

438 **L4: orchestration.** Agents exhibit major bottlenecks in all challenges including goal decomposition,
 439 dependency tracking, context switching, and tool use: **unclear decomposition** leads to aimless
 440 clicking (Figure 17 in Appendix F); **misordered context switches** break task dependencies (Figure 9)
 441 or initiating transactions before checking the balance (Figure 18); and **failure to leverage external
 442 tools** (Figure 10).

443 5 RELATED WORK

444 **Computer use benchmarks.** Existing computer-using benchmarks can be broadly categorized
 445 along several dimensions: by platform (e.g., Web (Deng et al., 2023; Zhou et al., 2023), Desktop (Xie
 446 et al., 2024; Cao et al., 2024; Xu et al., 2024b), or Mobile (Rawles et al., 2023; 2024; Chai et al.,
 447 2025)); by task type (e.g., understanding (Liu et al., 2024; Chen et al., 2024a), grounding (Cheng
 448 et al., 2024; Nayak et al., 2025), and end-to-end automation); and by scenario domain (e.g., everyday,
 449 office (Drouin et al., 2024), or professional (Cao et al., 2024; Li et al., 2025)). A recent trend is the
 450 adoption of dynamic environments (Xie et al., 2024; Cao et al., 2024; Xu et al., 2024b; Rawles et al.,
 451 2024; Sun et al., 2025). Focusing on end-to-end evaluation in daily scenarios on a dynamic desktop
 452 environment, OS-MAP is the first to systematically analyze task structures and automation levels
 453 grounded in real-world user needs, bridging capability evaluation with practical relevance.
 454

455 **Computer-using agents.** Recent advances in computer-using agents have been highly diverse. For
 456 modeling, efforts have focused on enhancing visual perception through high-resolution (Hong et al.,
 457 2024; Li et al., 2024b) or adaptive cropping and token selection (Zhang et al., 2024b; Lin et al., 2024;
 458 Wu et al., 2025) techniques. For data, two trends have emerged: (1) large-scale multi-task web-based
 459 pretraining (Cheng et al., 2024; You et al., 2024; Chen et al., 2024b; Wu et al., 2024b; Gou et al., 2024;
 460 Qin et al., 2025), and (2) supervised fine-tuning on high-quality interaction trajectories (Zhang et al.,
 461 2024c; Sun et al., 2024; Su et al., 2025). Reinforcement learning has been introduced to improve
 462 error recovery, and long-horizon reasoning (Fan et al., 2025; Lu et al., 2025; Xia and Luo, 2025;
 463 Liu et al., 2025). A parallel line of work builds ReAct-style (Yao et al., 2023) agents coordinating
 464 structured functional modules, with growing emphasis on hierarchical planning, systematic memory
 465 organization, and collaborative multi-agent systems (Agashe et al., 2024; Wu et al., 2024a; Jia et al.,
 466 2024; Agashe et al., 2025; Jiang et al., 2025; Wang et al., 2025; Zhang et al., 2025).

467 **AI capability levels.** Both industry and academia have long explored ways to define graded AI
 468 capabilities (Sheridan and Parasuraman, 2005; Parasuraman et al., 2000; Goertzel, 2014). A well-
 469 known example is the levels of driving automation (Committee, 2021) based on the human–system
 470 driving collaboration model. Recently, researchers have proposed grading schemes for artificial
 471 general intelligence with the performance–generality capability framework (Morris et al., 2024;
 472 Zhang et al., 2024a). A related research (Li et al., 2024c) discusses the intelligence levels of personal
 473 LLM agents in terms of collaboration patterns, but with abstract classification, vague levels, and
 474 overlapping capabilities. Our work grounds the performance–generality perspective in the concrete
 475 domain of computer automation, introducing clear levels aligned with real-world tasks, and further
 476 instantiates this taxonomy as OS-MAP benchmark to support quantitative evaluation.
 477

478 6 CONCLUSION

479 In this work, we propose a two-dimensional evaluation framework for computer-using agents,
 480 spanning automation levels and generalization scopes. We instantiate it as the OS-MAP Benchmark,
 481 comprising 416 tasks across 15 desktop applications, executed in a controllable and extensible
 482 environment to ensure quantitative and reproducible evaluation. Despite recent progress, OS-MAP
 483 remains highly challenging—state-of-the-art general-purpose and GUI-specialized VLMs still fall far
 484 short of human performance. Through in-depth failure analysis, we identify key capability bottlenecks
 485 at each automation level, laying a foundation for targeted improvements in future research.

486 REFERENCES
487488 Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao,
489 Xiangxin Zhou, Ziyu Zhao, et al. Os agents: A survey on mllm-based agents for general computing
490 devices use, 2024a.491 Siyuan Hu, Mingyu Ouyang, Difei Gao, and Mike Zheng Shou. The dawn of gui agent: A preliminary
492 case study with claude 3.5 computer use. *arXiv preprint arXiv:2411.10323*, 2024b.
493494 OpenAI. Introducing operator. <https://openai.com/index/introducing-operator/>, 2025. Accessed: 2025-04-25.
495496 Anthropic. Claude 3.5 sonnet. <https://www.anthropic.com/news/3-5-models-and-computer-use>, 2024. Accessed: 2025-04-25.
497498 Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang,
499 Pu Zhao, Chao Du, et al. Ufo2: The desktop agentos. *arXiv preprint arXiv:2504.14603*, 2025.
500501 Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao
502 Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native
503 agents. *arXiv preprint arXiv:2501.12326*, 2025.
504505 Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao
506 Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement.
507 *arXiv preprint arXiv:2402.07456*, 2024a.
508509 Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang,
510 Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*,
511 2025.512 Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu,
513 and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv
514 preprint arXiv:2412.04454*, 2024a.
515516 Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen
517 Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui
518 agents. *arXiv preprint arXiv:2410.23218*, 2024b.
519520 Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. Ap-
521 pagentx: Evolving gui agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*,
522 2025.523 Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent
524 s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint
525 arXiv:2504.00906*, 2025.526 Chengyou Jia, Minnan Luo, Zhuohang Dang, Qiushi Sun, Fangzhi Xu, Junlin Hu, Tianbao Xie, and
527 Zhiyong Wu. Agentstore: Scalable integration of heterogeneous agents as specialized generalist
528 computer assistant. *arXiv preprint arXiv:2410.18603*, 2024.
529530 Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty,
531 Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are
532 web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.533 Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua,
534 Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osword: Benchmarking multimodal agents
535 for open-ended tasks in real computer environments. *Advances in Neural Information Processing
536 Systems*, 37:52040–52094, 2024.
537538 Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong
539 Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating
multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

540 Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth
 541 Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic
 542 benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

543

544 Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyama-
 545 gundlu, and Oriana Riva. On the effects of data scale on ui control agents. *Advances in Neural*
 546 *Information Processing Systems*, 37:92130–92154, 2024a.

547 Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark
 548 prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*,
 549 2023.

550 On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related*
 551 *to Driving Automation Systems for On-Road Motor Vehicles*, April 2021. URL https://doi.org/10.4271/J3016_202104.

552

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

556

557 Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and
 558 Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents.
 559 *arXiv preprint arXiv:2410.05243*, 2024.

560

561 Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin
 562 Cong, Zhong Zhang, Yankai Lin, et al. Proactive agent: Shifting llm agents from reactive responses
 563 to active assistance. *arXiv preprint arXiv:2410.12361*, 2024.

564

565 Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar.
 566 Need help? designing proactive ai assistants for programming. In *Proceedings of the 2025 CHI*
 567 *Conference on Human Factors in Computing Systems*, pages 1–18, 2025.

568

569 Ana Paula Chaves and Marco Aurelio Gerosa. How should my chatbot interact? a survey on social
 570 characteristics in human–chatbot interaction design. *International Journal of Human–Computer*
 571 *Interaction*, 37(8):729–758, 2021.

572

573 Lizi Liao, Grace Hui Yang, and Chirag Shah. Proactive conversational agents in the post-chatgpt world.
 574 In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in*
 575 *Information Retrieval*, pages 3452–3455, 2023.

576

577 Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan
 578 Huang, and Hongsheng Li. A3: Android agent arena for mobile gui agents. *arXiv preprint*
 579 *arXiv:2501.01149*, 2025.

580

581 Sensor Tower. State of mobile 2025: The industry’s leading report. <https://sensortower.com/state-of-mobile-2025>, 2025. Accessed: 2025-04-25.

582

583 OECD. Ict access and usage database. <https://oe.cd/dx/ict-access-usage>, 2025.
 584 Accessed: 2025-04-25.

585

586 Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksan-
 587 dra Faust, Clement Farabet, and Shane Legg. Position: Levels of agi for operationalizing progress
 588 on the path to agi. In *Forty-first International Conference on Machine Learning*, 2024.

589

590 Kaiyan Zhang, Binqing Qi, and Bowen Zhou. Towards building specialized generalist ai with system 1
 591 and system 2 fusion. *arXiv preprint arXiv:2407.08642*, 2024a.

592

593 Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia:
 594 a benchmark for general ai assistants. In *The Twelfth International Conference on Learning*
 595 *Representations*, 2023.

596

597 Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su.
 598 Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing*
 599 *Systems*, 36:28091–28114, 2023.

594 Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan,
 595 and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models.
 596 *arXiv preprint arXiv:2401.13919*, 2024.

597

598 Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language
 599 instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020.

600 Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh,
 601 and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist
 602 autonomous agents for desktop and web. In *European Conference on Computer Vision*, pages
 603 161–178. Springer, 2024.

604

605 Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable
 606 real-world web interaction with grounded language agents. *Advances in Neural Information
 607 Processing Systems*, 35:20744–20757, 2022.

608 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
 609 Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building
 610 autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

611

612 Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. Understanding the
 613 weakness of large language model agents within a complex android environment. In *Proceedings of
 614 the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6061–6072,
 615 2024.

616 Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang
 617 Xiong, Hanchong Zhang, Wenjing Hu, Yuchen Mao, et al. Spider2-v: How far are multimodal
 618 agents from automating data science and engineering workflows? *Advances in Neural Information
 619 Processing Systems*, 37:107703–107744, 2024.

620

621 Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang,
 622 Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on
 623 consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024b.

624

625 Qiuishi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao,
 626 Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, et al. Scienceboard: Evaluating multimodal autonomous
 627 agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*, 2025.

628

629 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
 630 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint
 631 arXiv:2410.21276*, 2024.

632

633 Anthropic. Claude 3.7 sonnet. <https://www.anthropic.com/news/claude-3-7-sonnet>, 2025. Accessed: 2025-04-25.

634

635 Gemini Team. Introducing gemini 2.0: our new ai model for the agen-
 636 tic era. <https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/>, 2025. Accessed: 2025-04-25.

637

638 Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Yuchen Duan, Hao
 639 Tian, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for
 640 open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

641

642 Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu,
 643 Baolin Peng, Bo Qiao, Reuben Tan, Si Qin, Lars Liden, Qingwei Lin, Huan Zhang, Tong Zhang,
 644 Jianbing Zhang, Dongmei Zhang, and Jianfeng Gao. Gui-actor: Coordinate-free visual grounding
 645 for gui agents, 2025. URL <https://arxiv.org/abs/2506.03143>.

646

647 Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. An-
 648 droidinthewild: A large-scale dataset for android device control. *Advances in Neural Information
 649 Processing Systems*, 36:59708–59728, 2023.

648 Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang
 649 Yue. Visualwebbench: How far have multimodal llms evolved in web page understanding and
 650 grounding? *arXiv preprint arXiv:2404.05955*, 2024.

651 Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Liuyi Chen, Yilin Bai, Zhigang He, Chenlong
 652 Wang, Huichi Zhou, Yiqiang Li, et al. Gui-world: A dataset for gui-oriented multimodal llm-based
 653 agents. *arXiv e-prints*, pages arXiv–2406, 2024a.

654 Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal,
 655 Nicolas Chapados, M Tamer Özsü, Aishwarya Agrawal, David Vazquez, et al. Ui-vision: A desktop-
 656 centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*,
 657 2025.

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

661 Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan
 662 Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In
 663 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages
 664 14281–14290, 2024.

665 Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Prasad Sathya
 666 Moorthy, Jeff Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui 2: Mastering universal user interface
 667 understanding across platforms. *arXiv preprint arXiv:2410.18967*, 2024b.

668 Jiwen Zhang, Yaqi Yu, Minghui Liao, Wentao Li, Jihao Wu, and Zhongyu Wei. Ui-hawk: Unleashing
 669 the screen stream understanding for gui agents. *Preprints, manuscript/202408.2137*, 2024b.

670 Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang,
 671 and Mike Zheng Shou. Showui: One vision-language-action model for generalist gui agent. In
 672 *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

673 Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Sweenyngin, Jeffrey Nichols, Yinfei
 674 Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In
 675 *European Conference on Computer Vision*, pages 240–255. Springer, 2024.

676 Xuetian Chen, Hangcheng Li, Jiaqing Liang, Sihang Jiang, and Deqing Yang. Edge: Enhanced
 677 grounded gui understanding with enriched multi-granularity synthetic data. *arXiv preprint
 678 arXiv:2410.19461*, 2024b.

679 Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu
 680 Tang. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*,
 681 2024c.

682 Quishi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu,
 683 Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory
 684 construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*, 2024.

685 Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö Arik. Learn-by-
 686 interact: A data-centric framework for self-adaptive agents in realistic environments. *arXiv
 687 preprint arXiv:2501.10893*, 2025.

688 Yue Fan, Handong Zhao, Ruiyi Zhang, Yu Shen, Xin Eric Wang, and Gang Wu. Gui-bee:
 689 Align gui action grounding to novel environments via autonomous exploration. *arXiv preprint
 690 arXiv:2501.13896*, 2025.

691 Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and
 692 Hongsheng Li. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv
 693 preprint arXiv:2503.21620*, 2025.

694 Xiaobo Xia and Run Luo. Gui-r1: A generalist r1-style vision-language action model for gui agents.
 695 *arXiv preprint arXiv:2504.10458*, 2025.

702 Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang,
 703 and Fei Wu. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative
 704 reasoners. *arXiv preprint arXiv:2504.14239*, 2025.

705 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
 706 React: Synergizing reasoning and acting in language models. In *International Conference on*
 707 *Learning Representations (ICLR)*, 2023.

708 Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An
 709 open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*,
 710 2024.

711 Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and
 712 Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint*
 713 *arXiv:2501.11733*, 2025.

714 Thomas B Sheridan and Raja Parasuraman. Human-automation interaction. *Reviews of human factors*
 715 and ergonomics

716 and ergonomics, 1(1):89–129, 2005.

717 Raja Parasuraman, Thomas B Sheridan, and Christopher D Wickens. A model for types and levels of
 718 human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A:*
 719 *Systems and Humans*, 30(3):286–297, 2000.

720 Ben Goertzel. Artificial general intelligence: concept, state of the art, and future prospects. *Journal*
 721 *of Artificial General Intelligence*, 5(1):1, 2014.

722 Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu,
 723 Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the
 724 capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024c.

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

A QUALITATIVE EVALUATION MATRIX

758 This section explains the qualitative criteria used to position each method in the evaluation matrix
 759 (Figure 1). Given the subjective and heuristic nature of this analysis, the capability levels shown are
 760 approximate and do not reflect strict objectivity or fine-grained scale.

761

- 762 • **Academic methods.** For research models, capability levels are estimated based on their task
 763 scope, qualitative behavior, and whether they address key challenges or demonstrate core abilities.
 764 Quantitative results on relevant benchmarks are then used to refine their positions. For example,
 765 SeeClick [Cheng et al. \(2024\)](#), U-Ground [Gou et al. \(2024\)](#), and OS-Atlas [Wu et al. \(2024b\)](#)
 766 are all evaluated on the ScreenSpot [Cheng et al. \(2024\)](#) dataset, which focuses on GUI action
 767 grounding—a task category near L1. The latter two models incorporate rudimentary planning
 768 and can independently complete simple end-to-end tasks, suggesting capabilities closer to L2.
 769 Their exact placement is further adjusted based on performance scores (e.g., overall accuracy) and
 770 domain generalization, as ScreenSpot includes multiple domains. Other methods are evaluated
 771 similarly, using additional benchmarks such as AITW [Rawles et al. \(2023\)](#), GAIA [Mialon et al. \(2023\)](#),
 772 WebArena [Zhou et al. \(2023\)](#), and OSWorld [Xie et al. \(2024\)](#).
- 773 • **General-purpose models.** For models like GPT, we base our assessment on their qualitative and
 774 quantitative performance in OSWorld [Xie et al. \(2024\)](#) and OS-MAP, under both end-to-end and
 775 planning-grounding settings. These models show strong generalization—able to plan in nearly any
 776 scenario—but limited adaptivity, often struggling with unexpected events or common errors. As a
 777 result, they are positioned in the upper-middle region of the matrix.
- 778 • **Commercial products.** For tools like Microsoft Copilot, which lack quantitative evaluations or
 779 OSWorld-style experiments, we rely on a combination of official capability descriptions, public
 780 user discussions, and authors’ own usage experience. Earlier commercial products like Siri offer
 781 narrow functionality and low automation, while GitHub Copilot shows high-level code generation
 782 capabilities, often anticipating user needs. Microsoft Copilot for Windows 11 provides a more
 783 balanced and moderate level of capability and coverage.

784

B ENVIRONMENT STRUCTURE

785 The core of the OS-MAP environment consists of a virtual machine (VM) and a virtual machine
 786 controller (VMC). The host machine runs a VM using virtualization software such as VMware.
 787 This VM serves both as the source of visual observations and the target for action execution by
 788 the agents. The host communicates with the VM through a virtual network, enabling initialization,
 789 observation extraction, file transfer, and other forms of control. These components, together with
 790 tools for launching the VM, loading snapshots, and managing execution states, collectively form the
 791 VMC, which runs on the host side. The following sections detail how this architecture supports the
 792 task lifetime, initialization and configuration, state-based evaluation, and the design of the observation
 793 and action spaces.

794

B.1 TASK LIFETIME

795 OS-MAP consists of 416 tasks across diverse scenarios, each controlled and executed sequentially by
 796 a main evaluation loop. For each iteration, a new task evaluation is initiated. The lifetime of a task is
 797 composed of the following five stages:

- 801 1. **Initialization.** To ensure reproducibility, each task begins by loading a designated snapshot.
 802 Afterward, a predefined initialization script is executed. Snapshots and initialization scripts are
 803 designed to work in tandem, offering both high flexibility and low initialization overhead.
- 804 2. **Task execution.** Once initialized, the system enters the execution loop. At each step, the VMC
 805 captures the current observation and passes it to the agent. Based on the current state and
 806 interaction history, the agent outputs a textual action. This action is parsed and executed by the
 807 VMC within the VM. The loop continues until the agent either terminates voluntarily (via `DONE`
 808 or `FAIL`) or reaches the maximum allowed number of steps.
- 809 3. **Post-execution configuration (optional).** For certain tasks, the final state after agent execution is
 810 not directly extractable. In such cases, additional actions are required to bring the system into a

810 verifiable state. For example, after adding an item to the cart on the Decathlon website, the system
 811 needs to open the cart page so the evaluator can verify the result by inspecting the DOM tree.
 812

- 813 4. State extraction. The VMC includes a set of state extraction functions designed to retrieve relevant
 814 information from the VM. These serve as input for the next evaluation step.
- 815 5. Evaluation. Evaluation functions are task-specific and compare the extracted state against expected
 816 conditions. Depending on the nature of the state, corresponding comparison logic is applied—such
 817 as string matching, file equivalence, or key-value comparison.

818 **B.2 INITIALIZATION CONFIGURATIONS**

820 Task initialization in OS-MAP relies on a combination of restorable VM snapshots and configuration
 821 scripts. For simple tasks, initialization can be performed directly via scripts, which are pre-written
 822 command sequences that encapsulate commonly used operations—such as file downloads, application
 823 launches, API calls, webpage interactions via Playwright, and shell commands. For more complex or
 824 customized tasks (e.g., pre-created users and messages in Rocket.Chat), manual setup is conducted in
 825 advance and saved as a snapshot for fast recovery. In practice, task initialization uses both snapshot
 826 recovery and lightweight runtime configuration: snapshots (either from leaf nodes or key intermediate
 827 nodes of the snapshot tree) are loaded first, followed by scripted configuration. This hybrid approach
 828 ensures high flexibility and minimizes initialization time.

829 **B.3 STATE-BASED EVALUATION**

830 The primary motivation for introducing a dynamic environment is to enable state-based evaluation.
 831 The underlying logic is that as long as the system ultimately reaches a predefined desired state, the
 832 task is considered successfully completed—regardless of the specific sequence of actions taken to
 833 reach that state. This approach allows for a fair comparison between different execution trajectories
 834 of the same task.

835 Accordingly, each task JSON file must define both the target state and the method for extracting
 836 relevant system states. Common examples include retrieving specific files, reading software or system
 837 configurations, or extracting the content of rendered webpages via Playwright. In certain tasks, a
 838 post-config step is required to convert hard-to-access intermediate states into more easily extractable
 839 forms. Implementing state-based evaluation requires substantial reverse engineering of software and
 840 operating systems to locate and extract relevant data. Once the VM’s state is extracted to the host
 841 machine, an evaluation function compares it against the target state to determine whether the task has
 842 been successfully completed.

843 The evaluation methods are tailored to the extracted state types, typically involving file comparisons
 844 or configuration matching. In some cases, more specialized metrics—such as image similarity or
 845 fuzzy text matching scores—are used. All evaluation results are ultimately converted into a binary
 846 outcome, indicating task success or failure.

847 **B.4 OBSERVATION SPACE**

848 OS-MAP use screenshot for the only observation modality. Following OSWorld Xie et al. (2024), the
 849 VMC takes full-screen screenshots and preserves the cursor to align with human perception of the
 850 UI. The default resolution is 1920 × 1080, and supports adjustments to avoid overfitting on absolute
 851 pixel coordinates and generalization studies.

852 While previous benchmarks Xie et al. (2024); Cao et al. (2024) also used inputs such as the accessibility
 853 (a11y) tree and Set-of-Marks Yang et al. (2023) (SoM) prompted screenshots, OS-MAP relies
 854 solely on screenshots for two main reasons:

- 855 • Screenshots are easy to capture and align closely with human perception.
- 856 • They preserve rich visual information, whereas a11y trees and SoM formats can be overly exces-
 857 sively verbose, lossy, inaccurate, or unavailable in visually complex interfaces.

858 Although structured inputs sometimes yield better performance, recent methods have increasingly
 859 shifted toward using VLMs on raw screenshots Qin et al. (2025); Bai et al. (2025); Wu et al. (2024b),
 860 making pure visual input the more general and future-proof approach.

864 Table 4: Action types and parameters defined in action space COMPUTER_13, a variance we created
 865 for the potential reinforcement learning research based on our environment.
 866

Action Type	Parameters	Note
MOVE_TO	<i>x, y</i>	<i>Move the cursor to the specified position</i>
CLICK	<i>button,</i> <i>x, y,</i> <i>num_clicks</i>	<i>Click the left button if the button not specified, otherwise click the specified button; click at the current position if x and y are not specified, otherwise click at the specified position</i>
MOUSE_DOWN	<i>button</i>	<i>Press the left button if the button not specified, otherwise press the specified button</i>
MOUSE_UP	<i>button</i>	<i>Release the left button if the button not specified, otherwise release the specified button</i>
RIGHT_CLICK	<i>x, y</i>	<i>Right click at the current position if x and y are not specified, otherwise right click at the specified position</i>
DOUBLE_CLICK	<i>x, y</i>	<i>Double click at the current position if x and y are not specified, otherwise double click at the specified position</i>
DRAG_TO	<i>x, y</i>	<i>Drag the cursor to the specified position with the left button pressed</i>
SCROLL	<i>dx, dy</i>	<i>Scroll the mouse wheel up or down</i>
TYPING	<i>text</i>	<i>Type the specified text</i>
PRESS	<i>key</i>	<i>Press the specified key and release it</i>
KEY_DOWN	<i>key</i>	<i>Press the specified key</i>
KEY_UP	<i>key</i>	<i>Release the specified key</i>
HOTKEY	<i>keys</i>	<i>Press the specified key combination</i>
WAIT	-	<i>Wait until the next action</i>
FAIL	-	<i>Decide the task cannot be performed</i>
DONE	-	<i>Decide the task is done</i>
CALL_USER	-	<i>Call the simulated user to fill the credentials when logging-in</i>

889 B.5 ACTION SPACE

890
 891 OS-MAP adopts the Computer_13 action space from OSWorld, covering all basic mouse and
 892 keyboard operations—such as mouse movement, various clicks, drags, key presses, and hotkeys. It
 893 also includes three meta-actions: WAIT, FAIL, and DONE, which allow the agent to express task
 894 progress or termination conditions.

895 To support human-in-the-loop collaboration, OS-MAP introduces a new action: CALL_USER, used
 896 when human input is required—for example, entering sensitive information like login credentials. This
 897 helps define the agent’s permission boundary and enables more realistic human-agent cooperation. In
 898 OS-MAP benchmark, this action is only used during Google account login, where the agent yields
 899 control and a script autofills the credentials.

900 In total, OS-MAP defines 17 actions, summarized with their parameters in Table 4.

902 C TASK CURATION DETAILS

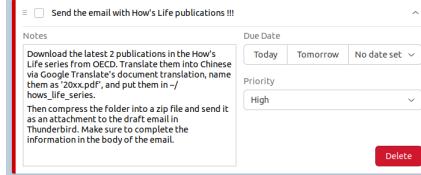
904
 905 All tasks in OS-MAP are designed according to a structured framework based on automation levels
 906 and hierarchies of user needs. Each task is implemented on the VM with a well-defined initial state
 907 and evaluation function, ensuring consistent and repeatable benchmarking. Task Curation was a
 908 collaborative effort by the authors, involving nine computer science students who jointly annotated
 909 and refined the tasks over approximately 600 hours of work.

910 Section C.1 outlines the standard six-stage pipeline for task creation, while Section C.2 provides a
 911 detailed walkthrough of how a representative Level-4 (L4) task was designed, iterated, and finalized
 912 from scratch. Section C.3 describes the process of filtering and re-annotating tasks imported from
 913 OSWorld Xie et al. (2024).

915 C.1 PIPELINE DESCRIPTIONS

916
 917 Based on the two-dimensional task organization framework described above, each task in OS-MAP
 918 is created by the co-authors following a standardized procedure:

918
919
920
921
922
923



924 Figure 12: Detailed Specification of the task goal in ToDo (not informed in the task instruction).
925

1. **Task selection.** We begin by identifying underrepresented scenes within the demand hierarchy. For each selected scene, we determine a representative application and outline a task concept aligned with that context.
2. **Exploration & specification.** Annotators study the target app or website using official documentation, demos, and hands-on interaction. They then define a concrete task objective, assign an appropriate difficulty level, and manually execute the task flow to verify feasibility. To prevent data contamination, task goals must not overlap with content from official materials; annotators are required to adapt or design new content accordingly.
3. **Instruction & configuration.** Annotators craft clear and concise task instructions and executable initialization configurations. Together, they control task difficulty—higher-level tasks omit details or include (human-recognizable) misleading cues, requiring agents to actively explore the environment for critical information.
4. **Reference state preparation.** The annotators manually complete the task to record a standard success state for the following evaluation process.
5. **Evaluation setup.** Evaluation involves comparing VM file or system states against predefined targets. Some tasks also require post-execution scripts or logic (postconfig) to expose the key status for assessment.
6. **Cross-validation.** Each task undergoes a rigorous review by two other annotators across several dimensions before inclusion: (1) task authenticity and representativeness, (2) clarity and unambiguity of instructions, (3) reproducibility, (4) correctness and (5) robustness of evaluation, (6) alignment with difficulty level, and (7) non-duplication.

948 C.2 A REPRESENTATIVE EXAMPLE

949 We illustrate the creation of a representative L4 task, from ideation to finalization:

1. **Task selection.** Upon reviewing the current task set, we found a gap in L4-level tasks within the office productivity domain—particularly tasks involving tool use, to-do management, and email communication. We thus defined a task prototype: write a to-do item that instructs the user to download multiple documents from a website, translate them using Google Translate, and send them as an email attachment to a colleague.
2. **Exploration & specification.** We selected the *How's Life* reports from the official OECD website as the document source. A to-do entry was added in a ToDo application, with a detailed task description specifying file names, save locations, and expected actions (see Figure 12).
3. **Instruction & configuration.** Through reverse engineering of the ToDo application, we identified the configuration file's location and edit protocol. Based on this, we created a config file and imported it during task initialization, so the to-do item loads automatically. Similarly, we reverse-engineered the Thunderbird email client, configuring its profile folder to pre-load an account and a draft email with a blank recipient field and no attachment.
4. **Reference state preparation.** We manually completed the task to obtain a reference success state—defined as the appearance of a new email in the recipient's local mail server directory. During testing, we observed long download times and limits on translation input and attachment size. To ensure feasibility, we reduced the requirement from translating six reports to just one.
5. **Evaluation setup.** The evaluation checks for textual equality between the expected and actual email file and is provided as part of the task package.
6. **Cross-validation.** The task was tested by two additional annotators to validate both procedure correctness and robustness—i.e., whether the task would still pass evaluation despite minor execution variations or small errors.

972 C.3 FILTERING OF TASKS FROM OSWORLD
973

974 We reused and adapted the majority of tasks from OSWorld. After careful filtering and re-annotation,
975 a total of 255 tasks were retained. The excluded tasks fall into three main categories:

- 976 • Redundant tasks within the same scenario – For example, the LibreOffice Calc tasks derived
977 from the SheetCopilot dataset often involved templated spreadsheet operations (e.g., statistical
978 summaries and charting). Only 1–2 representative tasks were kept to avoid unnecessary duplication.
- 979 • Tasks lacking general relevance – These focused too heavily on application-specific UI details, such
980 as fine-grained formatting combinations in office software (e.g., font size, line spacing, paragraph
981 alignment), rather than testing generalizable agent capabilities.
- 982 • Tasks marked as infeasible – These either had ambiguous descriptions or indirect, open-ended
983 solutions that made evaluation problematic. For example, the task *"Could you please convert a*
984 *PowerPoint presentation to video and play it with VLC?"* was removed. Retaining such tasks would
985 conflict with our design principle of aligning higher-level tasks with feasible, goal-driven behavior,
986 while rewriting them would introduce challenges in open-ended evaluation.

987 D EXPERIMENT DETAILS
988

989 D.1 MODEL BASELINES

990 We utilize the versions of gpt-4o-2024-11-20 and claude-3-7-sonnet-20250219 for
991 results of GPT-4o and Claude-3.7-Sonnet, respectively, need to be noted that result could be
992 changed from time since it is close-sourced. For all VLMs, we take the default hyper-parameters, *i.e.*,
993 we set the temperature parameter to 1.0, and top_p to 0.9, and the maximum number of tokens for
994 generation is set to 1500. We set the maximum number of interaction steps for L1, L2, L3 and L4
995 tasks to 15, 15, 30, and 50, respectively, which is sufficient to complete most tasks.

996 E PROMPTS FOR AGENTS
997

998 Multi-modal computer use agent baseline involves complex prompt engineering, including system
999 prompts, task instruction prompts, and step prompts. The following sections introduce these three
1000 types in detail and present representative examples of task instructions.

1001 E.1 SYSTEM PROMPT
1002

1003 The system prompt is the main part of prompt engineering in OS-MAP, including role description
1004 and observation space, action space, use cases, and format description with tips. The following will
1005 show the four parts of the system prompt. The complete system prompt is the splicing of the four
1006 parts.

1007 **Role description and observation space**

1008 You will act as an agent responsible for automating desktop
1009 ↵ computer tasks according to my instructions. You must possess
1010 ↵ strong knowledge of computer GUI operations and experience
1011 ↵ using common software applications.

1012 For each task, I will provide you with an instruction that
1013 ↵ describes the task goal and may include additional hints. You
1014 ↵ will then enter an operation loop, where you fully take over
1015 ↵ the control of the computer, performing one action step at a
1016 ↵ time. At each step, you will receive the history of actions and
1017 ↵ the current screenshot as observations, and you need to output
1018 ↵ what action to perform next. The action will be executed, and
1019 ↵ the loop continues with a new screenshot.

1020 Your output can include your reasoning--such as your observations,
1021 ↵ long-term planning, the objective of the current step, and the
1022 ↵ expected outcome. However, you must **ALWAYS** include a predicted
1023 ↵ ACTION and the action must conform to the action space
1024 ↵ described below. Your output must follow the specified FORMAT
1025 ↵ and include the correct `action_type` and required parameters.

```

1026
1027 Action space
1028
1029 ACTION_SPACE = [
1030     {
1031         "action_type": "MOVE_TO",
1032         "note": "move the cursor to the specified position",
1033         "parameters": {
1034             "x": {
1035                 "type": float,
1036                 "range": [0, X_MAX],
1037                 "optional": False,
1038             },
1039             "y": {
1040                 "type": float,
1041                 "range": [0, Y_MAX],
1042                 "optional": False,
1043             }
1044         },
1045         ...
1046         ...
1047     },
1048     {
1049         "action_type": "TYPING",
1050         "note": "type the specified text",
1051         "parameters": {
1052             "text": {
1053                 "type": str,
1054                 "range": None,
1055                 "optional": False,
1056             }
1057         },
1058         ...
1059         ...
1060     },
1061     {
1062         "action_type": "CALL_USER",
1063         "note": "Call the user to fill in the Google account or
1064             → password (the input box must be activated), one at a
1065             → time, according to the parameter call_type.",
1066         "parameters": {
1067             "call_type": {
1068                 "type": str,
1069                 "range": ["email", "password"],
1070                 "optional": False,
1071             }
1072         },
1073     },
1074     {
1075         "action_type": "WAIT",
1076         "note": "wait until the next action",
1077     },
1078     {
1079         "action_type": "FAIL",
1080         "note": "decide the task is failed or can not be
1081             → performed",
1082     },
1083     {
1084         "action_type": "DONE",
1085         "note": "decide the task is done",
1086     }
1087 ]
1088
1089

```

1080
1081**Use Cases**

1082

1083

Notes:

1084

1. To reiterate, regardless of whether you include reasoning, your
 ↳ output MUST contain an action in the SPECIFIED FORMAT (a
 ↳ dictionary enclosed in triple backticks as shown in the
 ↳ examples below), and it must include a valid `action_type` and
 ↳ parameters as defined above.

1085

2. For `MOUSE_MOVE`, you must specify the exact target `x` and `y`
 ↳ coordinates. The screen bounds are `X_MAX = 1920`, `Y_MAX =
 ↳ 1080`. The coordinates must fall within [0, 1920] and [0, 1080].
 ↳ Example:

1086

```
```{  
 "action_type": "MOUSE_MOVE",
 "x": 1319,
 "y": 65
}```
```

1087

3. For `[CLICK, RIGHT\_CLICK, DOUBLE\_CLICK, DRAG\_TO]`, specifying `x`  
 ↳ and `y` is optional. If omitted, the action defaults to the  
 ↳ current cursor position (often used after `MOUSE\_MOVE`).  
 ↳ However, it is RECOMMENDED to specify the coordinates  
 ↳ explicitly. Same format as `MOUSE\_MOVE`:

1088

```
```{  
  "action_type": "CLICK",  
  "x": 1319,  
  "y": 65
}```
```

1089

... more use cases ...

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

Format descriptions with tips

1112

11. Other special actions are `[WAIT, FAIL, DONE]`. Use them when
 ↳ you think it's necessary to wait, when the task has failed, or
 ↳ when it has succeeded. Each `WAIT` pauses for ~2 seconds. Do not
 ↳ declare `FAIL` lightly without attempting reasonable actions and
 ↳ explorations, but if you still cannot get out of the
 ↳ predicament after trying for several steps, you can declare it.
 ↳ Only use `DONE` when you are certain the task is completed
 ↳ successfully. Do NOT use dictionary format or triple backticks.
 ↳ Just output like the BARE "DONE" without any other thought or
 ↳ formatting.
12. If the task is file editing, make sure it is saved successfully.
 ↳ If there is no clear description of the file name, save
 ↳ location, etc., use the default.
13. If there are clear step-level instructions, please follow them
 ↳ strictly. Otherwise, you can do whatever you want as long as
 ↳ the task is completed.
14. My computer password is `password`. You may use it freely
 ↳ whenever `sudo` access is required.

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

Please think step by step. Carefully observe the current screenshot
 ↳ and then output your reasoning (optional), your plan, the
 ↳ current action and expected results, and most importantly, the
 ↳ FORMATTED ACTION.

Do NOT ask questions. Do NOT attempt to interact with the user in
 ↳ any way other than via the `CALL_USER` action. You are fully
 ↳ responsible for controlling the computer. Do NOT output
 ↳ anything else.

1134 E.2 TASK INSTRUCTION PROMPTS
 1135

1136 The task instruction is appended directly after the system prompt and is also loaded only once per
 1137 task. It specifies the concrete objective the agent is expected to complete, in the following format:
 1138

1139 You are asked to complete the following task: {{Instruction}}
 1140

1141
 1142 E.3 STEP PROMPT
 1143

1144 The system prompt is loaded once at the beginning of each task to provide the agent with general
 1145 instructions and behavioral priors. At every step during task execution, the agent also receives a
 1146 step prompt in the following fixed format, where "History" stands for the previous three interaction
 1147 history.
 1148

1149 {{History}} {{Screenshot}} Given the screenshot below, what is the
 1150 ↳ next step you will take to help complete the task?
 1151

1152
 1153 E.4 REPRESENTATIVE TASK INSTRUCTIONS
 1154

1155 Task instructions are written in a human-friendly tone, clearly stating the objective and specifying
 1156 any necessary details for evaluation. To enhance generalization and reduce overfitting to prompt
 1157 patterns, we ensure diversity in language style and phrasing across tasks. Below, we present several
 1158 representative examples.
 1159

1160 **Rotate Wallpapers (L4)**
 1161

1162 Download the Bing wallpaper for Italy from the latest 5 days in 4K
 1163 ↳ resolution to ~/Pictures/wallpapers and name them 0.jpg
 1164 ↳ (today's), 1.jpg, ..., add them to the wallpaper candidates,
 1165 ↳ and set the today's one as the wallpaper. Next, configure a
 1166 ↳ cron task to switch wallpapers in order at 00:00 every day. The
 1167 ↳ required script change_wallpaper.sh is already provided on the
 1168 ↳ desktop and can be used for cron tasks after only modifying the
 1169 ↳ wallpaper directory.
 1170

1171 **Zotero Citation (L4)**
 1172

1173 I am writing my course paper and I need to cite a reference. I
 1174 ↳ remember it in Zotero, but I can't remember which one it is.
 1175 ↳ Please help me find this article and imitate the two IEEE
 1176 ↳ formats above to complete the citation. Note that the font
 1177 ↳ format must also be the same. Then download this article to the
 1178 ↳ Documents/references folder and put it together with other
 1179 ↳ cited papers.
 1180

1181
 1182 **Meet Schedule (L3)**
 1183

1184 Did you see the meeting time sent in the DATA group in Rocket.Chat?
 1185 ↳ Add it to the event in Calendar, title it Team Meeting, and
 1186 ↳ make sure you don't make mistakes with the date, time, and
 1187 ↳ location.
 1188

1188 F CASE ANALYSIS

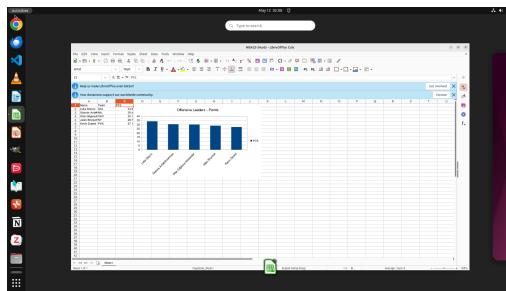
1190 This section provides the full context and failure analysis for several representative error cases
 1191 referenced in the main Analysis section §4.3.

1192 Figure 13 illustrates a severe hallucination, where the agent mistakenly identifies the current webpage
 1193 as a Chrome browser interface and treats the top search bar as a search engine input.

1194 Figures 14 and 15 show two L3 tasks in which the agent fails to adapt when the straightforward
 1195 method breaks down—for example, when the target element is missing from the screenshot.

1196 In Figure 16, another L3 task requires the agent to interact with a map embedded on the current page.
 1197 However, the agent ignores this context and instead jumps to a global Google Maps search, bypassing
 1198 the intended interaction.

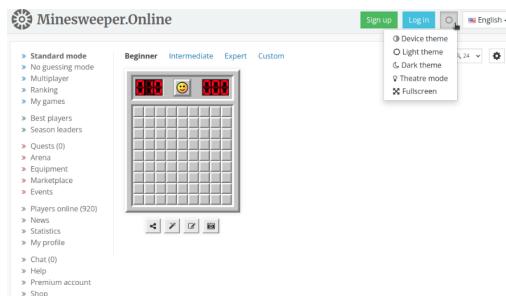
1199 Figures 17 and 18 depict two L4 tasks characterized by long instructions and complex dependencies.
 1200 In one case, the agent fails to properly decompose the instruction and proceeds with aimless explo-
 1201 ration; in the other, it confuses the order of contextual navigation and concrete operations. These
 1202 cases highlight the agent’s significant shortcomings in handling the high-level reasoning and planning
 1203 required at L4.



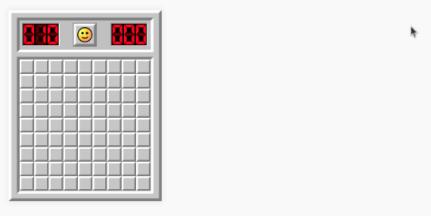
1206 Figure 13: (L2) The agent identifies this page as
 1207 Chrome and attempts to use the "search engine".



1208 Figure 14: (L3) The agent does not know to use
 1209 the keyboard shortcut to exit full-screen mode.



1210 Figure 15: (L3) Task instruction: *Enter theater mode and resize the scale to 48*. However, the resize
 1211 button is hidden in theater mode, and the agent does not know it should swap the execution order.



1212 G LIMITATIONS AND FUTURE WORK

1213 First, designing tasks that align precisely with automation levels and hierarchical needs often requires
 1214 carefully controlled initial states and evaluation functions built through extensive reverse engineering,
 1215 limiting the scalability of synthetic approaches. Moreover, the need for distribution and reproducibility
 1216 prevents alignment with many real-world scenarios, which are often tightly coupled with user accounts,
 1217 personalized content, or external effects, making them unsuitable as benchmark tasks.

1218 Future work may explore scalable methods to generate fine-grained, controllable tasks that better
 1219 cover the full range of user needs. Additionally, integrating environment-aware reward shaping could
 1220 enable finer supervision and continual improvement for computer-using agents. We hope OS-MAP
 1221 offers a solid foundation and actionable insights for advancing this direction.

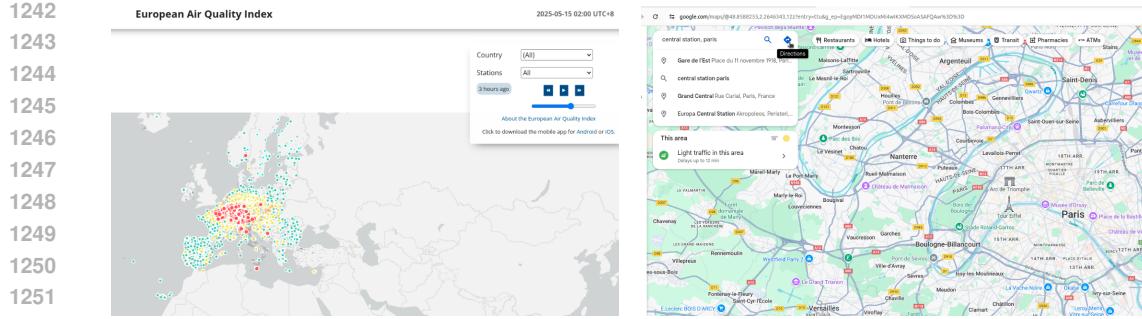


Figure 16: (L3) Task instruction: *Locate the MOST geographically central station in Paris on this map and jump to its location on Google Maps.* The agent simply ignores the current page (weather station map) and searches for subway stations in the center of Paris on Google Maps.

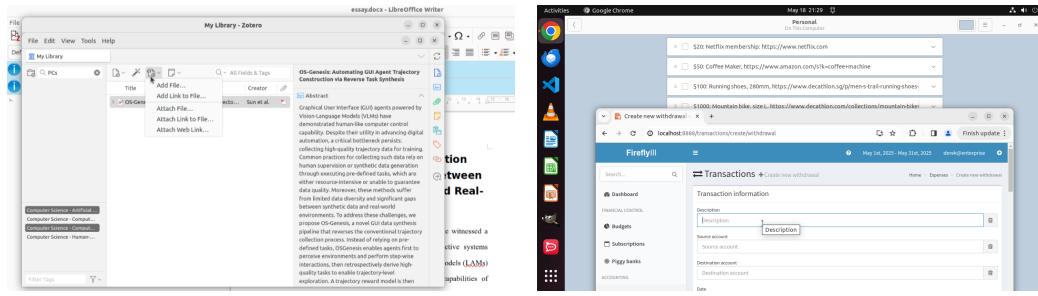


Figure 17: (L4) Task instruction: *I am writing my course paper and I need to cite a reference. I remember it in Zotero, but I can't remember which one it is. Please help me find this article and imitate the two IEEE formats above to complete the citation. Note that the font format must also be the same. Then download this article to the Documents/references folder and put it together with other cited papers.* The agent did not break down the task into subtasks and clicked aimlessly on the Zotero interface.

Figure 18: (L4) Task instruction: *I plan to use the money in my wallet to buy something to reward myself. Please check how much money is in my wallet account, and then buy the item of the corresponding amount in the todo list. Please choose the appropriate size and add it to the shopping cart. Then go back to the firefly and add a corresponding expense transaction, named the item name on the todo list.* The agent does not check the wallet balance or place an order, but tries to add a transaction record first.

H REPRODUCIBILITY STATEMENT

We provide an anonymous downloadable source code at <https://anonymous.4open.science/r/OSMap-C2F5/>. The deployment process of OS-MAP is detailed in the README.md of the code repository, while the experimental settings for running evaluations on OS-MAP are described in Section 4.