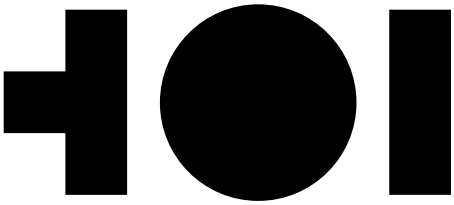
Surfer 2

The Next Generation of Cross-Platform Computer Use Agents



M. Andreux, M. Bakler, Y. Barbier, H. Benchekroun, E. Biré, A. Bonnet, R. Bordie, N. Bout, M. Brunel, A. Cambray, P.-L. Cedoz, A. Chassang, G. Cloix, E. Connelly, A. D. Constantinou, R. De Coster, H. de La Jonquière, A. Delfosse, M. Delpit, A. Deprez, A. Derupti, M. Diaz, S. D'Souza, J. Dujardin, A. Edmund, M. Eickenberg, A. Fatalot, W. Felissi, I. Herring, X. Koegler, E. Le Jumeau de Kergaradec, A. Lac, M. Langevin, C. Lauverjat, A. Loison, A. Manevich, A. Moyal, A. Nguyen Kerbel, M. Parovic, J. Revelle, G. Richard, M.L. Richter, R. Riochet, M. Santos, R. Savidan, L. Sifre, M. Theillard, M. Thibault, I. Valentini, T. Wu, L. Yie, K. Yuan, J. Zubovskij

H Company — Alphabetical order

October 2025

Abstract

Building agents that generalize across web, desktop, and mobile environments remains an open challenge, as prior systems rely on environment-specific interfaces that limit cross-platform deployment. We introduce Surfer 2, a unified architecture operating purely from visual observations that achieves state-of-the-art performance across all three environments. Surfer 2 integrates hierarchical context management, decoupled planning and execution, and self-verification with adaptive recovery, enabling reliable operation over long task horizons. Our system achieves 97.1% accuracy on WebVoyager, 69.6% on WebArena, 60.1% on OSWorld, and 87.1% on AndroidWorld, outperforming all prior systems without task-specific fine-tuning. With multiple attempts, Surfer 2 exceeds human performance on all benchmarks. These results demonstrate that systematic orchestration amplifies foundation model capabilities and enables general-purpose computer control through visual interaction alone, while calling for a next-generation vision language model to achieve Pareto-optimal cost-efficiency.

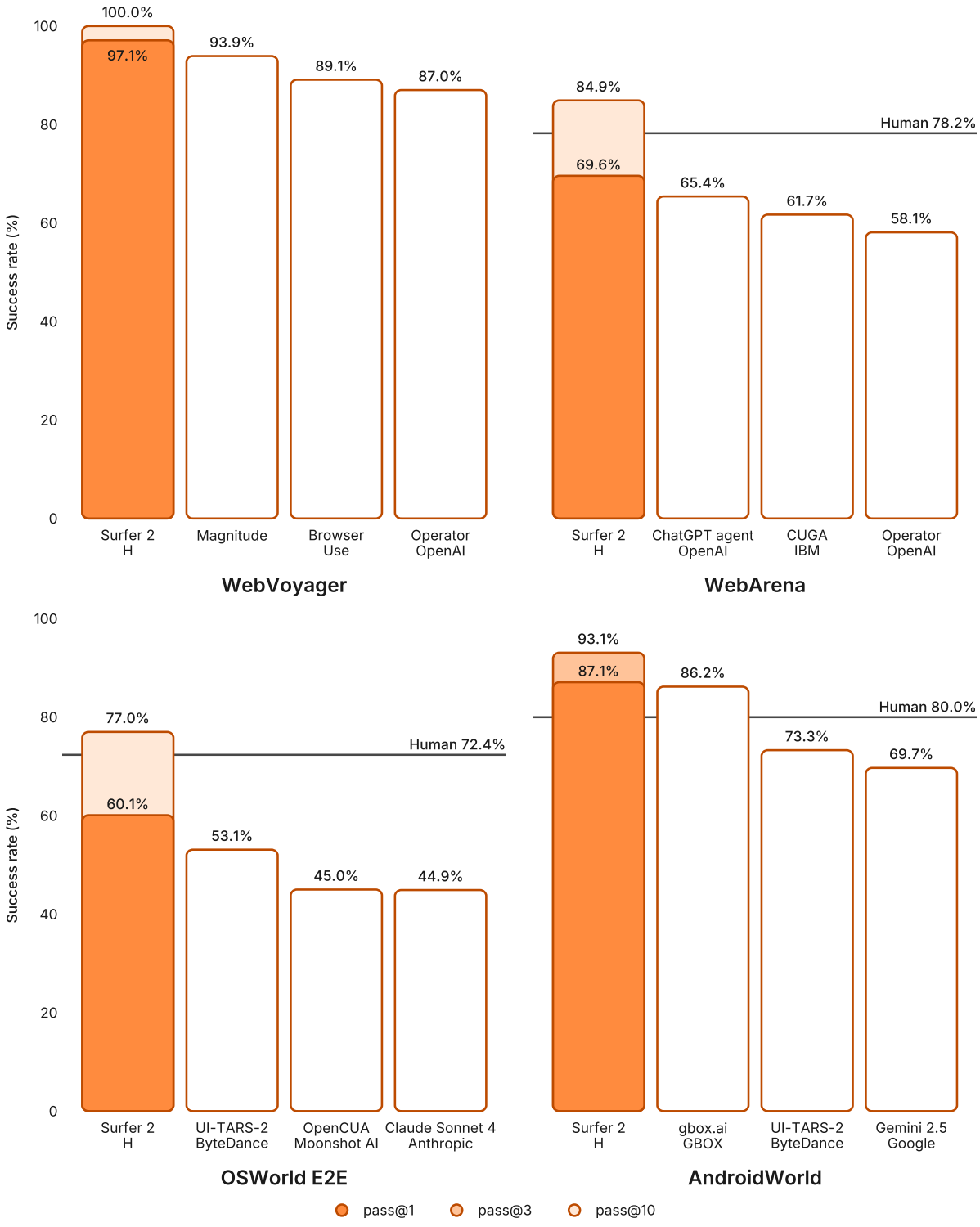


Figure 1: Surfer 2 state-of-the-art performance on WebVoyager, WebArena, OSWorld E2E, and AndroidWorld. Human performance is indicated when available.

1 Introduction

Recent advances in Large Language Models (LLMs) and Vision-Language Models (VLMs) unlocked remarkable reasoning capabilities for agentic use cases [1]. However, turning these capabilities into reliable, general-purpose agents that operate autonomously in Graphical User Interfaces (GUIs) on complex, real-world tasks remains challenging. In recent years, one dominant path has been to train increasingly large models with minimal scaffolding like tool-call [2, 3] to improve agentic capabilities [4, 5, 6]. In contrast, this work presents an alternative perspective: with proper orchestration and system design, existing state-of-the-art models can achieve human-level performance and exceed prior systems across multiple benchmarks.

Prior approaches require environment-specific adaptations, such as DOM parsers for web navigation, accessibility trees for mobile interfaces, or specialized APIs for desktop applications, limiting generalization across diverse digital environments. This work introduces **Surfer 2**, a unified, hierarchical agent architecture designed for complex tasks across desktop, web, and mobile environments using purely visual interaction.

Surfer 2 comprises three components: **Orchestrator** (optional high-level planner), a **Navigator** (low-level GUI executor), and a **Validator** (evaluation module). Surfer 2 integrates third-party frontier models and H Company's Holo1.5 models [7] in a design that separates long-term strategic planning from short-term tactical execution. A key design insight in Surfer 2 is architectural flexibility. It can enable or bypass the Orchestrator to match task complexity. For long-horizon problems, the Orchestrator runs as a high-level planner in the plan-and-act style [8], where it decomposes the user task into verifiable goals, plans ahead, and delegates targeted subtasks to the Navigator. For simple tasks, the Orchestrator is bypassed and the Navigator is invoked directly. Built on our previous SurferH agent [9], the Navigator follows a ReAct (reason+act) loop [10]. It perceives the environment purely via screenshots, reasons about the next step, and executes constrained keyboard and mouse-level controls with pixel-accurate UI localization from Holo1.5. Upon subtask completion, a Validator inspects the latest screenshots, the execution history, and the proposed answer to assess subtask success in one of two ways: (1) if the Orchestrator is enabled, it leverages the Validator's report and either advances to the next subgoal or replans accordingly, or (2) if the Orchestrator is disabled, the Validator's feedback is sent directly to the Navigator, which integrates it into its reasoning and continues the ReAct loop until the task is completed or a termination condition is reached.

Without task-specific fine-tuning, Surfer 2 attains state-of-the-art results on four major benchmarks spanning different Computer Use environments (web browser, desktop, mobile): WebVoyager [11] and WebArena [12], OSWorld [13], and AndroidWorld [14]. On OSWorld and AndroidWorld, Surfer 2 surpasses the human baseline, underscoring that expert agent design is as crucial as model capability.

2 Related Work

The development of Computer Use agents capable of controlling computers, web browsers, and mobile devices represents a key frontier in AI. In many real-world scenarios, agents encounter tools and software for which no API or Model Context Protocol (MCP) is available, leaving GUIs as the only viable control surface. Consequently, recent research has focused on enabling general-purpose agents to perceive, reason about, and act within GUIs, transforming visual interaction into a universal interface for autonomous computer use.

2.1 Agents for GUI Control

The development of agents capable of controlling computers through their graphical interfaces has been a long-standing goal in AI. Early work focused on script-based or rule-based systems for automating specific, repetitive tasks [15]. Subsequent research introduced reinforcement learning (RL) and computer vision techniques for GUI automation [16, 17]. Recent advances leverage large language models (LLMs) and vision-language models (VLMs) for more generalized and adaptable control [11, 13].

2.1.1 Browser Use Agents

Web navigation agents are a highly active research area, with benchmarks like WebVoyager [11] and WebArena [12] providing standardized evaluation environments. Early methods often relied on interpreting the Document Object Model (DOM) to understand a page's structure and content [18, 19, 12, 20]. While

effective, these text-based approaches struggle with visually-rich elements, dynamic content, or situations where visual layout and context are crucial for task success.

Our work operates on a fundamentally different, multimodal principle: we use image-based states (screenshots) as the primary input, following the approach of Surfer-H [9]. This enables our agent to perceive and interact with the digital environment in a more human-like way, leveraging the visual understanding of large multimodal models (LMMs). Previous works have already explored this path; for instance Set-of-Marks [11] augments screenshots with labeled bounding boxes for each UI element and refers to these labels when issuing clicks. In contrast, our approach operates directly on unaltered screenshots and predicts raw pixel coordinates. Other approaches apply reinforcement learning to learn skills from scratch [21, 22, 10], whereas we achieve superior performance without task-specific fine-tuning. Our work, similar to [23], focuses on architectural improvements rather than model training.

2.1.2 Desktop Computer Use Agents

Beyond the web, agents for general Computer Use present unique challenges due to the heterogeneity of application interfaces, multi-app workflows and the requirement for system-level control. OSWorld [13] has emerged as a leading benchmark for desktop automation with tasks focusing on Ubuntu, evaluating agents across diverse applications such as LibreOffice, GIMP, VS Code and the OS system. Complementing it, WindowsAgentArena [24] provides a very similar suite of tasks for Windows-based environments.

Early efforts in Computer Use were open-source, and include OSAtlas [25] and Aguvis [26]. These established the foundation for developing and evaluating vision-language-action agents capable of operating within general computer environments. Subsequent research, such as [27, 28, 29, 30, 31, 32, 33], has since expanded upon these efforts, exploring diverse architectures and training paradigms to enhance reasoning, perception, and control capabilities. Closest to our work, Agent S3 [34] introduces Behavior Best-of-N (bBoN), where it generates multiple parallel trajectories and then selects the most successful one. To make this selection feasible, it first converts dense, raw trajectories into concise "behavior narratives" that summarize the agent's actions and their effects. A judge model then compares these narratives to pick the best rollout. Agent S3 can also invoke a coding agent to perform programmatic edits such as bulk operations, file transformations, and structured parsing.

2.1.3 Mobile Use Agents

Over the last two years, the field of mobile agent research has grown with the introduction of the AndroidWorld [14] benchmark, which provides a rigorous testbed for agents that require touch-based interactions and multi-app workflows, and the Android in the Wild (AITW) dataset [35, 36, 37]. Research in this area, including work like [38, 39, 19, 40, 41], has focused on developing and training specialized models capable of interpreting mobile UIs and executing gestures. While these works demonstrate the power of model-centric approaches, our architecture proves that the same level of performance can be achieved by orchestrating existing models, accommodating the visual and interactive distinctions of mobile platforms.

2.2 Frontier Models for Agents

The performance of modern GUI agents is inextricably linked to the capabilities of the underlying frontier models, particularly LLMs and VLMs. Models like GPT-4.1 [5], o3 [42], Claude 4.5 Sonnet [4], and Gemini 2.5 [6] have demonstrated exceptional abilities in multimodal reasoning, zero-shot generalization, and long-context understanding. While many studies have focused on scaling up models [19] or fine-tuning models on large, domain-specific datasets [36, 37], our work takes a different direction. We employ frontier models and demonstrate that a carefully designed system can achieve state-of-the-art results.

2.3 Localization Models

Accurate user interface (UI) element localization remains a key technical challenge for GUI agents operating on visual data. The agent must infer the precise coordinates of a target such as a button, text field, or icon from a screenshot, often conditioned on a natural-language description. This task, known as visual grounding, has motivated the development of specialized vision-language models designed specifically for

UI contexts. For instance, UI-TARS [27], Holo1.5 [9, 7], and CogAgent [19] are models specifically trained for localizing UI elements. Our system relies on Holo1.5 [7], a specialized localization model, to bridge the gap between the agent's high-level action plan (e.g., "click the 'Submit' button") and the pixel-level action required for execution. Our work highlights how effective orchestration and integration of such specialized models are as important as their individual capabilities.

2.4 Agent Architectures and Learning Paradigms

Our work on a hierarchical, multi-agent framework builds on established principles in reinforcement learning (RL) and agent design, but it fundamentally differs by employing off-the-shelf models without training. The concept of separating high-level planning from low-level execution has been explored in various contexts [8, 43], including in GUI agent frameworks that use experience-augmented hierarchical planning and internal experience retrieval to address long-horizon tasks [44]. A notable example in the mobile domain is the K²-Agent framework [45], which explicitly separates a high-level, training-free planner from a low-level, learning-based executor. Our system extends these ideas through a persistent environment state and robust validation mechanisms, ensuring consistency and recoverability across extended workflows. Our use of a planner coordinating sub-agents is a form of multi-agent orchestration, similar in principle to [23].

Our system's self-correction and validation loop relates to recent work on autonomous evaluation and refinement of agent behavior [46, 10], particularly in methods that reinforce agents through linguistic feedback and reflective episodic memory rather than weight updates [47]. The use of an external verification module aligns with research on using language models for critique and dense rewards [48]. However, unlike many of these approaches that rely on offline or online RL to update model weights [49, 50, 51, 52, 53, 54, 41], we do not perform any parameter updates. Instead, our findings highlight that coordination and self-correction at the system level can substitute for learning at the model level. We show that even in-context learning, without any gradient updates, can achieve remarkable performance, similar to previous findings [55, 56].

3 Agent Architecture

We outline the components required to build Surfer 2 as shown in Figure 2.

3.1 Design Philosophy

The design of our architecture follows five core principles. First, we adopt a **separation of concerns**: high-level planning, managed by the Orchestrator, is decoupled from low-level execution handled by the Navigator, allowing each component to specialize and improve independently. Second, we employ an automatic **hierarchical context** mechanism, giving each component access to relevant global information such as the overarching goal, current plan, completed work, and immediate subtask while maintaining scope-specific focus. Third, we ensure a **shared environment state**, in which elements like browser sessions or open applications persist across subtasks and components, enabling incremental progress in dynamic environments. Fourth, we emphasize **explicit validation** through multi-stage verification processes to limit error propagation and promote self-correction. Finally, we employ **chain-of-thought reasoning** with all modules except the Localizer, reasoning explicitly in natural language for better long-horizon performance.

3.2 Orchestrator

Main roles. The Orchestrator shoulders three key roles simultaneously. As a **planner**, it decomposes the user goal into sequential, verifiable subtasks and delegates their execution to the Navigator. As a **coordinator**, it evaluates the Navigator's outcomes and Validator feedback to detect potential errors or incomplete progress, and replans when necessary to recover from failures. Finally, as a **communicator**, it decides when to terminate and synthesizes validated results into a coherent final response. The system operates hierarchically: the Orchestrator maintains the global plan and reasoning loop, while the Navigator executes grounded actions within its own local observation–action cycle. This organization ensures efficient task management and robustness through explicit verification and adaptive recovery.

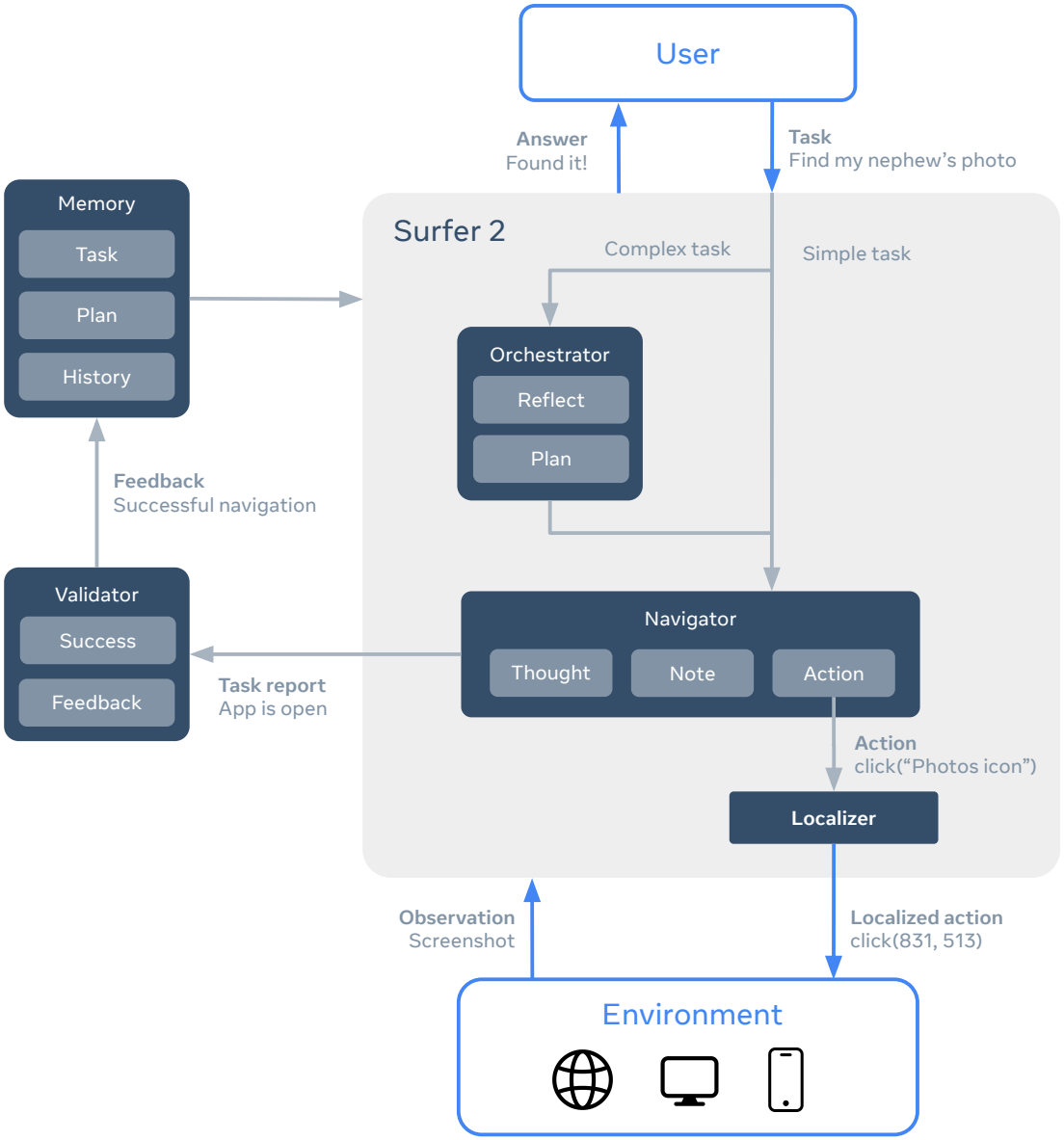


Figure 2: **Surfer 2 architecture**: an optional Orchestrator plans, the Navigator acts via a Holo1.5 Localizer, and a Validator provides feedback.

Orchestrator Memory. At each decision step, the Orchestrator maintains the overall task objective, the current plan, its execution status, a history of all past interactions with the Navigator, and the latest visual observations from the environment. This persistent state enables the Orchestrator to track progress, learn from previous attempts, and make informed decisions about when to continue, replan, or terminate.

Adaptive planning. Surfer 2 employs an adaptive strategy for deciding between single-call execution and multi-step planning based on task complexity heuristics. For simple tasks, the Navigator is directly invoked. For complex tasks involving multiple phases, cross-referencing information, or dependent subtasks, the Orchestrator is triggered to create an explicit plan that breaks the task into a sequence of manageable goals. After each Navigator execution, the Orchestrator analyzes its task report, updates the current goal status, and updates its plan with a mitigation strategy if failures have been detected. The Orchestrator

has 4 available actions, which balance high-level planning (`create_plan`, `replan`) with tactical execution (`delegate`) and external communication (`answer`).

3.3 Navigator

Our Navigator agent is an improved version of the previous agent Surfer-H [9] that now operates across web, desktop, and mobile environments. It extracts relevant information through note-taking, generates action sequences with UI grounding, and verifies task completion through integrated validation. Its vision–language policy interprets the current environment state and past trajectory to produce a structured output consisting of a **note** (information extracted from the latest observation), a **thought** (reasoning about the next step), and an **action** (the operation to execute). The localizer optionally grounds localizable actions to screen coordinates, enabling interaction with specific UI elements. Finally, the environment executes the grounded action and returns a new screenshot observation of the state. When the policy issues an `answer` action to signal task completion, the Validator assesses the result's completeness and correctness before allowing termination, providing a crucial safeguard against premature or inaccurate responses.

3.4 Localizer

The localizer bridges the gap between textual element descriptions (provided by the Navigator) and precise screen coordinates, a critical capability for reliable action execution. It grounds any UI element references in the action to precise screen coordinates, converting textual descriptions like "blue submit button" into clickable (x, y) coordinates. This visual grounding problem is fundamental to GUI automation, as even small localization errors can cause actions to miss their intended targets entirely. We use Holo1.5 models as Localizer throughout our experiments.

3.5 Validator

Validation is a critical component for preventing premature termination and ensuring answer quality through systematic verification, inspired by prior work on self-reflective agents and feedback-driven verification mechanisms [57, 58]. The Validator examines the Navigator's complete execution trace including the task specification, reasoning history, sequence of actions, proposed answer, and the most recent k screenshots. It then determines whether the solution satisfies the task requirements based on observable evidence. This *VLM-as-a-Judge* operates at two levels of the system hierarchy. Within the Navigator, it evaluates each `answer` action before allowing termination: if validation fails, the Navigator resumes execution with the Judge's feedback integrated into its context, enabling self-correction; if validation succeeds, the episode concludes and the answer is returned. At the Orchestrator level, the Validator's assessment is combined with the Navigator's final report, allowing the Orchestrator to decide whether to accept, refine, or replan the outcome.

4 Evaluation Methodology

We evaluate our system on four major benchmarks spanning web, desktop, and mobile environments: WebVoyager [11], WebArena [12], OSWorld [13], and AndroidWorld [14]. All experiments employ models without any task-specific fine-tuning, isolating the contribution of our hierarchical agent architecture from model improvements. This experimental design demonstrates that superior performance can be achieved through careful system design and agent orchestration alone, independent of model scale or domain adaptation.

4.1 Benchmarks

Here, we outline the main characteristics of the benchmarks we use and briefly describe each of them. Comprehensive details, including task distributions, evaluation metrics, and corrections to prior evaluation inconsistencies, are presented in Appendix A.

Key benchmark characteristics. **Multi-step reasoning**: Tasks require sequential actions with conditional branching based on intermediate observations, assessing the agent's ability to plan and adapt over long horizons. **Real-world environments**: Benchmarks rely on realistic environments, such as actual websites (WebVoyager, WebArena), production desktop applications (OSWorld), and authentic mobile apps (AndroidWorld) rather than simulators, exposing agents to dynamic content, varied layouts, and real-world edge cases. **High-precision visual grounding**: Success depends on accurate localization of UI elements within pixel-dense screenshots, where small coordinate errors cause action failures, demanding tight integration of vision and language reasoning. **Diverse interaction modalities**: The benchmark suite spans mouse/keyboard control (desktop), touch gestures (mobile), and hybrid web interactions, ensuring that our orchestration framework generalizes across fundamentally different action spaces and environment dynamics.

WebVoyager consists of 643 tasks spanning 15 popular websites in its original formulation, including e-commerce, travel, and information platforms (e.g., Amazon, Booking.com, ArXiv). These tasks require complex agent interactions such as visually grounded information retrieval, comparison, and multi-step form completion. However, the dynamic nature of live websites introduces instability that can render tasks obsolete. To ensure experimental comparability, we adopted the curated 590-task subset established by Magnitude [59]. Details on access restriction mitigations are available in Appendix A.1.

WebArena is a suite of 812 tasks designed to test navigation capabilities across diverse web environments, including an e-commerce site, a social forum, a GitLab instance, a content management system, and a map interface. Refinements were made to the original WebArena implementation (see Appendix A.2).

OSWorld spans 369 real Computer Use tasks on Ubuntu systems, spanning production applications such as LibreOffice, GIMP, Chrome, Thunderbird, VS Code, and VLC. Tasks test realistic workflows including document editing, image manipulation, email management, and multi-application coordination and are scored through deterministic programmatic checks. We focus on the **Foundation E2E GUI** category, which constrains the agent's action space to human-performable GUI operations: mouse clicks, drags, keyboard inputs, and shortcuts without calling APIs or executing code. This setting is the most representative of true Computer Use capability, as it requires agents to perceive and act directly on arbitrary interfaces rather than relying on handcrafted integrations. By contrast, the broader OSWorld All category permits code-level operations (e.g., Python snippets or API calls) that can bypass interface-level reasoning, potentially inflating scores through tool-specific shortcuts rather than genuine GUI understanding. We therefore evaluate Surfer 2 in the stricter Foundation E2E GUI regime and compare it against competitors within that category, emphasizing generalization to unseen applications and fidelity to human interaction. For context, the current highest score in the "All" category is held by Agent S3 at 69.9% accuracy [34].

AndroidWorld evaluates mobile agent capabilities across 116 tasks spanning the Android OS itself and 20 real-world applications. These tasks require touch-based interactions, app navigation, and multi-app workflows verified through Android Debug Bridge (ADB)-based state inspection. Each task is scored with a verification metric in $\{0, 0.5, 1\}$, corresponding to failure, partial success, and success, respectively. Importantly, the Android Emulator provides access to the accessibility tree (a11y), which agents can leverage to perform tasks. To better demonstrate the performance of our unified agent, we deliberately avoid using this accessibility tree and instead rely solely on visual (screenshot-based) inputs.

4.2 Configuration

For each benchmark, we adapt the model configuration and component selection to the specific environment and task complexity, as determined through ablation studies (see Table 1. In WebVoyager and WebArena, the Orchestrator operates for up to 20 steps, while the Navigator may take up to 50 steps to complete navigation subtasks. For OSWorld and AndroidWorld, the agent runs without an Orchestrator, relying solely on the Navigator for both planning and execution. In OSWorld, we enforce a minimum of 15 and a maximum of 100 steps to prevent premature termination. Once the upper limit is reached, the agent's memory is cleared without resetting the environment enabling continued progress while keeping the context

Table 1: Model configuration across benchmarks.

Benchmark	Orchestrator	Navigator		
		Policy	Judge	Localizer
WebVoyager	o3	Claude Sonnet 4.5	GPT 4.1	Holo1.5 7B
WebArena	o3	Claude Sonnet 4.5	o3	Holo1.5 72B
OSWorld	None	Claude Sonnet 4.5	o3	Holo1.5 72B
AndroidWorld	None	o3	o3	Holo1.5 72B

length bounded. In AndroidWorld, the step limit is set to 150, reflecting the higher difficulty of certain tasks (e.g., `OsmAndTrack` which has an empirically determined optimal horizon of roughly 60 steps).

5 Results

In this section, we describe both qualitative and quantitative main results obtained on the four benchmarks.

5.1 WebVoyager

Surfer 2 establishes a new state of the art on the WebVoyager benchmark, achieving a 97.1% success rate and surpassing the previous best performance of 93.9% [59]. This strong performance is consistent across nearly all tested websites (see Figure 3), with the exception of the Cambridge Dictionary domain, where anti-bot measures such as CAPTCHAs hindered execution. Surfer 2 achieves a perfect 100% pass@10, effectively saturating the benchmark using test-time scaling. In a localizer ablation study, substituting Holo1.5 7B with UI-TARS 7B [27] reduced performance to 94.7%, confirming that Surfer 2 's gains derive from the combination of high-quality components and effective orchestration.

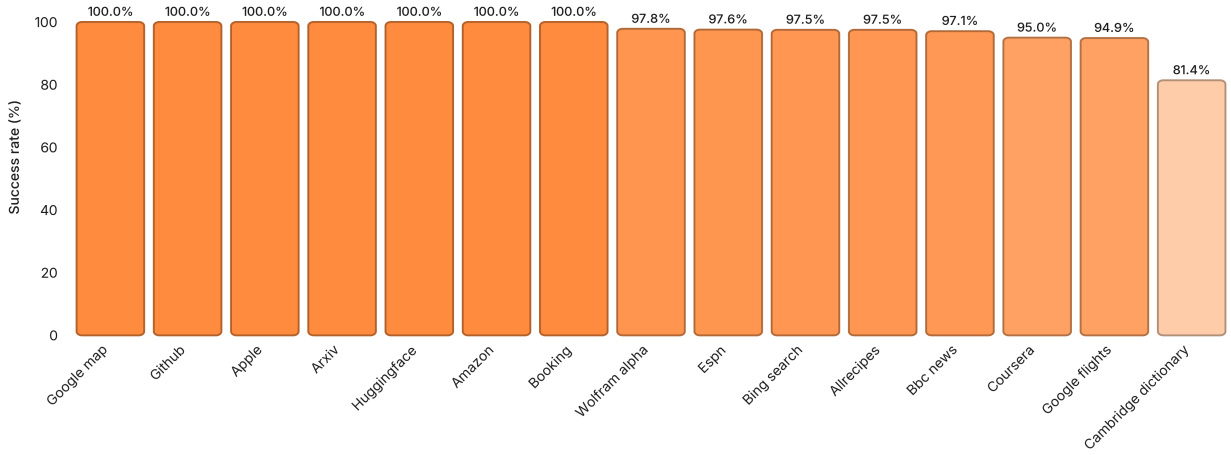


Figure 3: Per-website performance of Surfer 2 on the WebVoyager benchmark.

5.2 WebArena

Surfer 2 reaches a new state-of-the-art with a pass@1 success rate of 69.6% on WebArena. The agent performed robustly on social media tasks, e.g., 77% on Reddit, while struggling with e-commerce workflows, averaging only 58% on shopping sites. Many tasks in this domain remain challenging, see Figure 4.

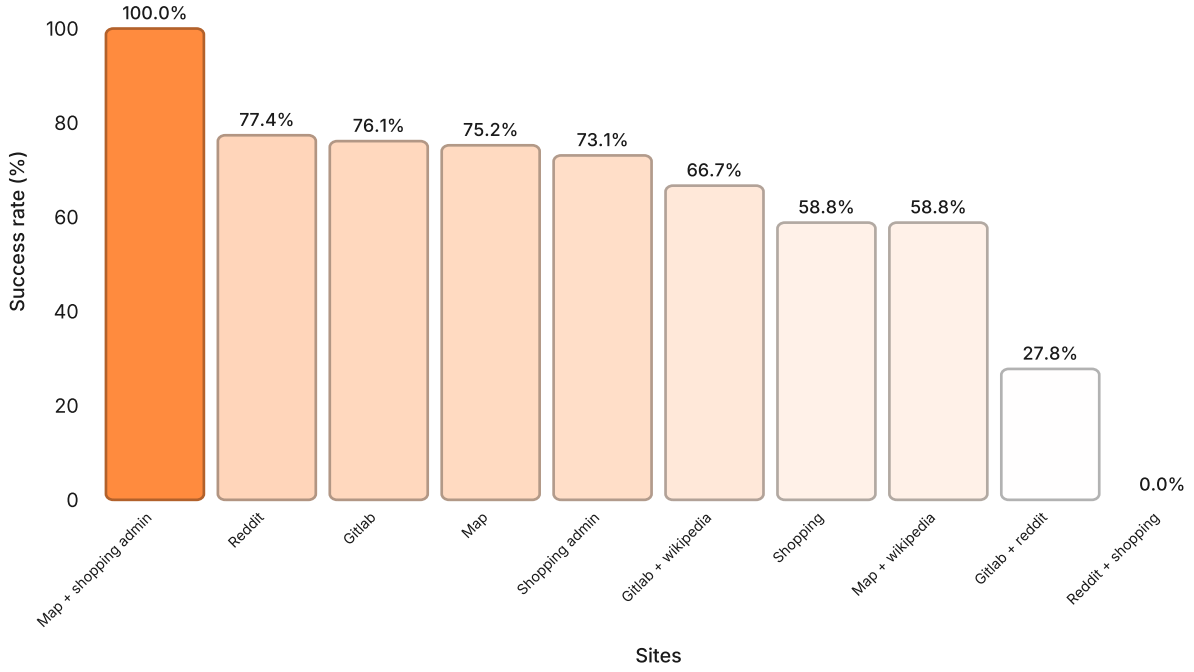


Figure 4: Per-domain performance of Surfer 2 on the WebArena benchmark.

Test-time scaling. Sampling multiple independent trajectories of Surfer 2 leads to a substantial performance gain from 69.6% with pass@1 to 84.9% with pass@10. Scaling parallel trajectories primarily expands task coverage, revealing what the agent *can* do rather than increasing single-run reliability. The diversity of successful paths indicates that most residual failures arise from local exploration traps rather than systematic reasoning errors. High task coverage offers a valuable signal about the agent's effective action space and highlights its potential for reinforcing successful behavioral patterns through model training.

5.3 OSWorld

Surfer 2 achieves a state-of-the-art success rate of 60.1% on OSWorld in the Foundation E2E GUI category. With five attempts (pass@5), performance rises to 72.0%, closely matching the human baseline of 72.4%. At ten attempts (pass@10), Surfer 2 surpasses human performance with 77.0%. In Figure 5 we report success rates across task categories; in all of them, Surfer 2 exceeds the accuracy of 50%, performing especially well on programming-related tasks in environments like VSCode and OS. Interestingly, Surfer 2 completed several tasks labeled as infeasible by human evaluators, which were excluded from our success rate (see Figure 8).

Localizer ablation. Within the same agentic framework, Holo1.5 72B Localizer achieved the highest performance, reaching 60.1% compared to 58.4% for Holo1.5 7B and 56.9% for UI-TARS 7B [27]. This confirms the importance of accurate spatial grounding for GUI reasoning: Holo1.5 generalizes more effectively to diverse Computer Use interfaces, enabling precise GUI interaction.

5.4 AndroidWorld

Surfer 2 achieves an accuracy of 87.1% across the 116 tasks, surpassing all previous approaches relying solely on visual interaction [60, 45]. Figure 6 shows the agent's success rate across difficulty levels as defined in the original AndroidWorld paper. It achieves a near-perfect performance on Easy tasks (98.4%), maintains strong results on Medium tasks (86.1%), and shows a notable drop on Hard tasks (52.6%). This trend

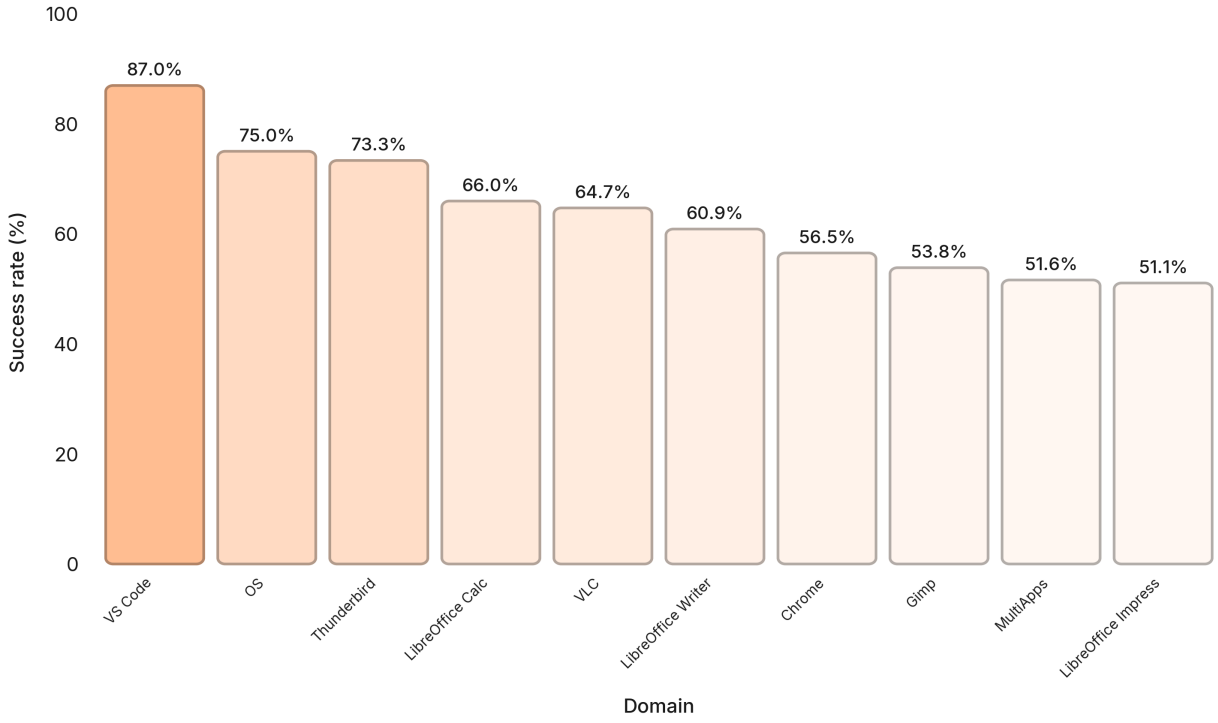


Figure 5: Surfer 2 performance per task category on the OSWorld benchmark.

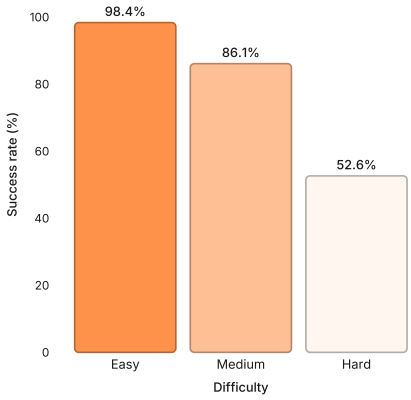


Figure 6: Surfer 2 accuracy per difficulty category on AndroidWorld.

indicates that while Surfer 2 generalizes well to moderately complex interactions, performance decreases on tasks requiring long-horizon reasoning or intricate multi-step coordination.

Performance per category. Performance varies across categories (Figure 7), with memorization and transcription (both 50%) identified as the most difficult due to limited memory and text handling, and multi-app tasks (37.5%) remaining the primary challenge for robust cross-application reasoning.

Test-time scaling. Allowing a small retry budget yields consistent improvements on AndroidWorld: pass@1 reaches 87.1%, pass@2 improves to 90.5% (+3.4%), and pass@3 climbs to 93.1% (+6.0% over pass@1). These gains indicate that a modest number of parallel attempts recovers many near-miss failures, suggesting that most errors arise from stochastic perception or planning rather than fundamental limitations.

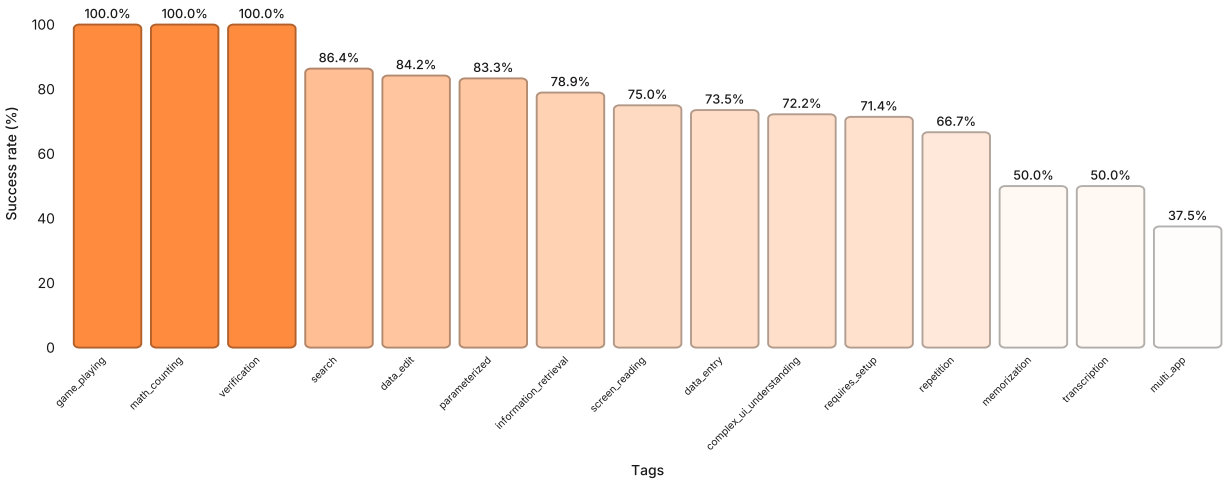


Figure 7: Surfer 2 accuracy by task category on AndroidWorld.

Localizer ablation. Substituting Holo1.5 with UI-TARS reduces pass@1 success from 87.1% to 81.9% (−5.2%) on AndroidWorld, indicating the localizer as performance bottleneck. A drop in accuracy in localization of interactions with small targets and iconographic widgets cascades into incorrect actions. This ablation underscores that the spatial grounding from Holo1.5 is crucial for end-to-end reliability.

Implications. Surfer 2 effectively manages complex multi-app, multi-step workflows. The remaining challenges lie in (i) real-time perception, (ii) maintaining memory over long sequences, and (iii) stronger GUI semantics—particularly learning app-specific visual conventions and icon mappings.

6 Key Insights and Path Forward

Our empirical evaluation reveals critical factors for agent success. Prompt engineering proves surprisingly impactful, with minor wording changes yielding 5–10% accuracy swings, underscoring the sensitivity of LLM reasoning to input formatting. Model variance remains substantial even at temperature 0 for the judge, necessitating multi-sampling strategies to achieve robust performance. Persistent context across subtasks reduces navigation steps required by 30–40%, proving essential for multi-goal tasks that build on prior subtasks. Multi-stage validation intercepts 15–20% of errors before propagation, while Orchestrator's hierarchical decomposition provides natural retry boundaries and improved interpretability.

Despite achieving human-level performance on complex tasks, several bottlenecks constrain practical deployment. Stochastic model outputs require expensive multi-sampling for reliability, with Orchestrator costs reaching $1–5 per complex task when using frontier reasoning models. Even state-of-the-art localizers fail on 5–8% of UI elements due to dynamic content and ambiguous descriptions. Long-horizon tasks exceeding 50 steps face context window limits and compounding errors, while LLM-based evaluation itself carries 5–10% error rates that complicate benchmarking.

Our results demonstrate that proper agent orchestration with fixed, general-purpose models can achieve state-of-the-art accuracy, with modular designs generalizing across web, desktop, and mobile environments.

While the principles of agent orchestration are approaching maturity, the remaining bottlenecks – cost, variance, and speed – limit practical, real-world deployment. The path forward lies in resolving these challenges. To address this, we are focusing on developing a new family of smaller, specialized models, to achieve comparable, if not superior, performance at a fraction of the current cost, reaching Pareto optimality [9].

References

[1] Google DeepMind. *Project Astra: A Universal Multimodal AI Assistant*. https://blog.google/technology/google-deepmind/gemini-universal-ai-assistant/. Announced at Google I/O2025. Demonstrates multimodal perception, visual reasoning, and autonomous interaction integrated into Gemini and Android environments. May 2025.

[2] Eleti, A., Harris, J., and Kilpatrick, L. *Function calling and other API updates*. OpenAI. June 13, 2023. URL: https://openai.com/index/function-calling-and-other-api-updates/ (visited on 10/21/2025).

[3] Schick, T. et al. *Toolformer: Language Models Can Teach Themselves to Use Tools*. 2023. arXiv: 2302.04761 [cs.CL]. URL: https://arxiv.org/abs/2302.04761.

[4] Anthropic. *Introducing Claude Sonnet 4.5*. Accessed: 2025-16-10. Sept. 2025. URL: https://www.anthropic.com/news/claude-sonnet-4-5.

[5] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[6] Comanici, G. et al. *Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities*. 2025. arXiv: 2507.06261 [cs.CL]. URL: https://arxiv.org/abs/2507.06261.

[7] Company, H. *Holo1.5 - Foundational Models for Computer Use Agents*. 2025. URL: https://huggingface.co/Hcompany/Holo1.5-7B.

[8] Erdogan, L. E. et al. *Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks*. en. arXiv:2503.09572 [cs]. Apr. 2025. DOI: 10.48550/arXiv.2503.09572. URL: http://arxiv.org/abs/2503.09572 (visited on 10/08/2025).

[9] Andreux, M. et al. *Surfer-H Meets Holo1: Cost-Efficient Web Agent Powered by Open Weights*. en. arXiv:2506.02865 [cs]. June 2025. DOI: 10.48550/arXiv.2506.02865. URL: http://arxiv.org/abs/2506.02865 (visited on 10/12/2025).

[10] Yang, Z. et al. *ReAct Meets ActRe: When Language Agents Enjoy Training Data Autonomy*. en. arXiv:2403.14589 [cs]. Apr. 2024. DOI: 10.48550/arXiv.2403.14589. URL: http://arxiv.org/abs/2403.14589 (visited on 10/08/2025).

[11] He, H. et al. *WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models*. arXiv:2401.13919 [cs]. June 2024. DOI: 10.48550/arXiv.2401.13919. URL: http://arxiv.org/abs/2401.13919 (visited on 05/15/2025).

[12] Zhou, S. et al. *WebArena: A Realistic Web Environment for Building Autonomous Agents*. arXiv:2307.13854 [cs]. Apr. 2024. DOI: 10.48550/arXiv.2307.13854. URL: http://arxiv.org/abs/2307.13854 (visited on 05/15/2025).

[13] Xie, T. et al. *OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments*. en. arXiv:2404.07972 [cs]. May 2024. DOI: 10.48550/arXiv.2404.07972. URL: http://arxiv.org/abs/2404.07972 (visited on 10/12/2025).

[14] Rawles, C. et al. *AndroidWorld: A Dynamic Benchmarking Environment for Autonomous Agents*. en. arXiv:2405.14573 [cs]. Apr. 2025. DOI: 10.48550/arXiv.2405.14573. URL: http://arxiv.org/abs/2405.14573 (visited on 10/16/2025).

[15] Cypher, A. and Halbert, D. C. *Watch what I do: programming by demonstration*. MIT press, 1993.

[16] Liu, E. Z. et al. "Reinforcement learning on web interfaces using workflow-guided exploration". In: *arXiv preprint arXiv:1802.08802* (2018).

[17] Hsiao, Y.-C. et al. "Screenqa: Large-scale question-answer pairs over mobile app screenshots". In: *arXiv preprint arXiv:2209.08199* (2022).

[18] Nakano, R. et al. *WebGPT: Browser-assisted question-answering with human feedback*. 2022. arXiv: 2112.09332 [cs.CL]. URL: https://arxiv.org/abs/2112.09332.

[19] Hong, W. et al. *CogAgent: A Visual Language Model for GUI Agents*. arXiv:2312.08914 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2312.08914. URL: http://arxiv.org/abs/2312.08914 (visited on 05/15/2025).

[20] LaVagueAI. *Lavague: Web agent framework for builders*. https://docs.lavague.ai/en/latest/. Accessed: 2025-10-21.

[21] Zhou, Y. et al. *Proposer-Agent-Evaluator(PAE): Autonomous Skill Discovery For Foundation Model Internet Agents*. arXiv:2412.13194 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.13194. URL: http://arxiv.org/abs/2412.13194 (visited on 05/15/2025).

[22] Qi, Z. et al. *WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning*. arXiv:2411.02337 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2411.02337. URL: http://arxiv.org/abs/2411.02337 (visited on 05/15/2025).

[23] Abuelsaad, T. et al. *Agent-E: From Autonomous Web Navigation to Foundational Design Principles in Agentic Systems*. arXiv:2407.13032 [cs]. July 2024. DOI: 10.48550/arXiv.2407.13032. URL: http://arxiv.org/abs/2407.13032 (visited on 05/15/2025).

[24] Bonatti, R. et al. *Windows Agent Arena: Evaluating Multi-Modal OS Agents at Scale*. Sept. 2024. arXiv: 2409.08264 [cs.AI].

[25] Wu, Z. et al. *OS-ATLAS: A foundation action model for generalist GUI agents*. Oct. 2024. arXiv: 2410.23218 [cs.CL].

[26] Xu, Y. et al. *Aguvis: Unified Pure Vision Agents for Autonomous GUI Interaction*. Dec. 2024. arXiv: 2412.04454 [cs.CL].

[27] Qin, Y. et al. *UI-TARS: Pioneering Automated GUI Interaction with Native Agents*. en. arXiv:2501.12326 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.12326. URL: http://arxiv.org/abs/2501.12326 (visited on 10/08/2025).

[28] Wang, H. et al. *UI-TARS-2 Technical Report: Advancing GUI Agent with Multi-Turn Reinforcement Learning*. en. arXiv:2509.02544 [cs]. Sept. 2025. DOI: 10.48550/arXiv.2509.02544. URL: http://arxiv.org/abs/2509.02544 (visited on 10/08/2025).

[29] Ye, J. et al. *Mobile-Agent-v3: Fundamental Agents for GUI Automation*. en. arXiv:2508.15144 [cs]. Sept. 2025. DOI: 10.48550/arXiv.2508.15144. URL: http://arxiv.org/abs/2508.15144 (visited on 10/16/2025).

[30] Putta, P. et al. *Agent Q: Advanced reasoning and learning for autonomous AI agents*. Aug. 2024. arXiv: 2408.07199 [cs.AI].

[31] Wang, X. et al. *OpenCUA: Open Foundations for Computer-Use Agents*. Oct. 2025. arXiv: 2508.09123 [cs.AI].

[32] Fu, T. et al. *Mano Report*. Sept. 2025. arXiv: 2509.17336 [cs.MM].

[33] He, Y., Jin, J., and Liu, P. *Efficient Agent Training for Computer Use*. May 2025. arXiv: 2505.13909 [cs.AI].

[34] Gonzalez-Pumariega, G. et al. *The Unreasonable Effectiveness of Scaling Agents for Computer Use*. 2025. arXiv: 2510.02250 [cs.AI]. URL: https://arxiv.org/abs/2510.02250.

[35] Rawles, C. et al. *Android in the Wild: A Large-Scale Dataset for Android Device Control*. arXiv:2307.10088 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2307.10088. URL: http://arxiv.org/abs/2307.10088 (visited on 05/15/2025).

[36] Li, W. et al. *On the Effects of Data Scale on UI Control Agents*. en. arXiv:2406.03679 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2406.03679. URL: http://arxiv.org/abs/2406.03679 (visited on 10/16/2025).

[37] Zhang, Z. and Zhang, A. *You Only Look at Screens: Multimodal Chain-of-Action Agents*. arXiv:2309.11436 [cs]. June 2024. DOI: 10.48550/arXiv.2309.11436. URL: http://arxiv.org/abs/2309.11436 (visited on 05/15/2025).

[38] Wang, J. et al. *Mobile-Agent: Autonomous Multi-Modal Mobile Device Agent with Visual Perception*. en. arXiv:2401.16158 [cs]. Apr. 2024. DOI: 10.48550/arXiv.2401.16158. URL: http://arxiv.org/abs/2401.16158 (visited on 10/16/2025).

[39] Tang, L. et al. *MagicGUI: A Foundational Mobile GUI Agent with Scalable Data Pipeline and Reinforcement Fine-tuning*. en. arXiv:2508.03700 [cs]. Sept. 2025. DOI: 10.48550/arXiv.2508.03700. URL: http://arxiv.org/abs/2508.03700 (visited on 10/16/2025).

[40] Bai, H. et al. *DigiRL: Training In-The-Wild Device-Control Agents with Autonomous Reinforcement Learning*. arXiv:2406.11896 [cs]. June 2024. DOI: 10.48550/arXiv.2406.11896. URL: http://arxiv.org/abs/2406.11896 (visited on 05/15/2025).

[41] Bai, H. et al. *Digi-Q: Learning Q-Value Functions for Training Device-Control Agents*. arXiv:2502.15760 [cs]. Feb. 2025. DOI: 10.48550/arXiv.2502.15760. URL: http://arxiv.org/abs/2502.15760 (visited on 05/15/2025).

[42] OpenAI. *OpenAI o3 and o4-mini System Card*. System Card. Version dated April 16 2025. OpenAI, Apr. 2025. URL: https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf.

[43] Zhou, Y. et al. *ArCHer: Training Language Model Agents via Hierarchical Multi-Turn RL*. arXiv:2402.19446 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.19446. URL: http://arxiv.org/abs/2402.19446 (visited on 05/15/2025).

[44] Agashe, S. et al. *Agent S: An Open Agentic Framework that Uses Computers Like a Human*. 2024. arXiv: 2410.08164 [cs.AI]. URL: https://arxiv.org/abs/2410.08164.

[45] K2-Agent. *K^2-Agent: Co-Evolving Know-What and Know-How for Hierarchical Mobile Device Control*. GitHub Repository. Accessed: 2025-10-16. 2025. URL: https://github.com/k2-agent/k2-agent.

[46] Pan, J. et al. *Autonomous Evaluation and Refinement of Digital Agents*. arXiv:2404.06474 [cs]. Oct. 2024. DOI: 10.48550/arXiv.2404.06474. URL: http://arxiv.org/abs/2404.06474 (visited on 05/15/2025).

[47] Shinn, N. et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: 2303.11366 [cs.AI]. URL: https://arxiv.org/abs/2303.11366.

[48] Cao, M. et al. *Beyond Sparse Rewards: Enhancing Reinforcement Learning with Language Model Critique in Text Generation*. en. arXiv:2401.07382 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2401.07382. URL: http://arxiv.org/abs/2401.07382 (visited on 10/08/2025).

[49] Kumar, A. et al. *Conservative Q-Learning for Offline Reinforcement Learning*. en. arXiv:2006.04779 [cs]. Aug. 2020. DOI: 10.48550/arXiv.2006.04779. URL: http://arxiv.org/abs/2006.04779 (visited on 10/08/2025).

[50] Kostrikov, I., Nair, A., and Levine, S. *Offline Reinforcement Learning with Implicit Q-Learning*. en. arXiv:2110.06169 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2110.06169. URL: http://arxiv.org/abs/2110.06169 (visited on 10/08/2025).

[51] Verma, S. et al. *CHAI: A CHatbot AI for Task-Oriented Dialogue with Offline Reinforcement Learning*. arXiv:2204.08426 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2204.08426. URL: http://arxiv.org/abs/2204.08426 (visited on 05/15/2025).

[52] Snell, C. et al. *Offline RL for Natural Language Generation with Implicit Language Q Learning*. en. arXiv:2206.11871 [cs]. May 2023. DOI: 10.48550/arXiv.2206.11871. URL: http://arxiv.org/abs/2206.11871 (visited on 10/08/2025).

[53] Wang, H. et al. *Offline Reinforcement Learning for LLM Multi-Step Reasoning*. en. arXiv:2412.16145 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.16145. URL: http://arxiv.org/abs/2412.16145 (visited on 10/08/2025).

[54] Zhai, Y. et al. *Enhancing Decision-Making for LLM Agents via Step-Level Q-Value Models*. en. arXiv:2409.09345 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2409.09345. URL: http://arxiv.org/abs/2409.09345 (visited on 09/08/2025).

[55] Ma, Y. J. et al. *Vision Language Models are In-Context Value Learners*. en. arXiv:2411.04549 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2411.04549. URL: http://arxiv.org/abs/2411.04549 (visited on 10/08/2025).

[56] Zhou, H. et al. *Memento: Fine-tuning LLM Agents without Fine-tuning LLMs*. en. arXiv:2508.16153 [cs]. Aug. 2025. DOI: 10.48550/arXiv.2508.16153. URL: http://arxiv.org/abs/2508.16153 (visited on 10/13/2025).

[57] Shinn, N. "Reflexion: Language Agents with Verbal Reinforcement Learning". In: *arXiv preprint arXiv:2303.11366* (Oct. 2023). URL: https://arxiv.org/abs/2303.11366.

[58] Agashe, S. et al. "Agent S: An Open Agentic Framework that Uses Computers Like a Human". In: *arXiv preprint arXiv:2410.08164* (Oct. 2024). URL: https://arxiv.org/abs/2410.08164.

[59] Magnitude. *Magnitude WebVoyager Benchmark Results*. GitHub Repository. Accessed: 2025-10-20. 2025. URL: https://github.com/sagekit/webvoyager?tab=readme-ov-file.

[60] gbox.ai. *GBOX*. GitHub Repository. Accessed: 2025-10-20. 2025. URL: https://github.com/babelcloud/android_world_benchmark.

[61] OpenAI. *Computer Use Agents: Evaluation Supplementary Information*. https://cdn.openai.com/cua/CUA_eval_extra_information.pdf. Accessed: 2025-10-21. 2025.

[62] Zhou, S. et al. *WebArena: A Realistic Web Environment for Building Autonomous Agents*. GitHub Repository. Accessed: 2025-10-16. 2025. URL: https://github.com/web-arena-x/webarena.

A Appendix

A.1 WebVoyager Evaluation

- **Description**: 590 tasks across 15 websites: Amazon, Apple, Google Flights, Booking.com, ArXiv, GitHub, Hugging Face, Coursera, BBC News, Cambridge Dictionary, Allrecipes, Google Maps, Bing Search, ESPN and Wolfram Alpha.
- **Task types**: Information retrieval, comparison and navigation.
- **Evaluation**: We updated the evaluation protocol by replacing the single GPT-4V judge with a majority vote over 3 GPT-4o judge calls (temperature 0, using last 5 screenshots), retaining the original WebVoyager evaluation prompt. This ensemble method is designed to reduce variance, mitigate single-judge bias, and prevent spurious results from isolated errors, thereby yielding more reliable and robust evaluations.
- **Environment**: Live websites on Selenium-controlled Chrome. Running this benchmark introduces practical challenges, such as bot-detection mechanisms like CAPTCHAs and IP-based access blockers. To mitigate these restrictions, we employed proxy rotation and redirected all Google-dependent tasks to Bing Search, as in [9], to ensure uninterrupted execution.

A.2 WebArena Evaluation

- **Description**: 812 tasks across self-hosted 6 websites: GitLab, Reddit, an E-commerce website called OneStopShop, an online store content management system (CMS), OpenStreetMap and the English Wikipedia.
- **Task types**: Information retrieval, comparison, navigation, multi-step and multi-websites actions, involving form filling and search across one or two websites. To improve grounding and consistency, we added lightweight, category-specific initialization prompts, loosely inspired by OpenAI's per-site prompting approach [61].
- **Evaluation**: modular evaluation framework consisting of three criteria that can be combined or used independently:
  - URL Match: The agent's final URL must match a predefined target.
  - HTML Artifact: A required success artifact must be present in the final page's DOM.
  - Model-based Assessment: A language model evaluates the correctness of the final output using majority voting with 5 GPT 4.1 judges.
- **Environment**: Controlled web servers with fixed initial states accessed using Selenium-controlled Chrome.

A.2.1 Methodological Difficulties with WebArena.

Technical challenges. While WebArena is a valuable benchmark, its evaluation poses several methodological challenges. The benchmark requires self-hosting of its websites, a process that is not seamless and which required manual intervention, notably to resolve issues in the OpenStreetMap environment. Its highly stateful design further complicates reproducibility, as tasks are interdependent and their outcomes can be affected by residual states from previous runs. These characteristics also make large-scale parallel execution impractical, substantially increasing the computational cost and time required to obtain robust metrics

LLM-as-a-Judge. The benchmark's original evaluation framework integrated programmatic checks of the final state with a single LLM-based assessment. We improved its rigor by replacing heuristic string matching (e.g., exact-match or keyword checks) with an ensemble of five independent GPT-4.1 judges, aggregated by majority vote. This approach reduces variance, mitigates single-judge bias, penalizes false positives in *must include* tasks where substring matching previously sufficed, and allows minor, semantically irrelevant variations in *exact match* tasks that would otherwise fail under literal comparison. Given that WebArena contains 176 *must include* tasks versus only 45 *exact match* tasks, our methodology provides a more balanced and reliable assessment.

Task Corrections. We conducted a manual review of the benchmark dataset [62], resulting in the correction of 71 tasks with erroneous labels. The scope of these modifications ranged from minor typographical fixes to the resolution of critical logical inconsistencies that fundamentally affected evaluation outcomes. A summary of these corrections is provided in Table 2. For a transparent comparison, we conducted an ablation study without these manual fixes, relying solely on a Large Language Model (LLM) as the judge for string comparison. In this configuration, our approach achieves a success rate of 67.4%, a result that still surpasses the previous state-of-the-art performance.

Table 2: Summary of WebArena corrections with examples.

Category	**Description**	**Example correction**
Data accuracy corrections	Rectifying errors in numerical values (e.g., quantities, measurements, prices), and making corrections to specific names or entities to reflect the correct data.	Corrected order count from "24 orders" to "**21 orders**" (Task 50).
Typographical and spelling fixes	Addressing simple spelling errors in task fields such as `intent`, `intent_template`, or `instantiation_dict` to improve clarity.	Corrected "canlled" to "**cancelled**" (Task 202).
URL and task focus updates	Updating URLs, parameters, and location references to ensure that tasks point to the intended content or have a more precise focus.	Updated URL parameter from `sort=created_asc` to `sort=created_date` (Task 45).
Evaluation flexibility improvements	Replacing specific data points (e.g., domains) with flexible placeholders to make the evaluation process more robust against variations in expected outputs.	Replaced a specific domain with the placeholder: <`fuzzy_general_domain`> (Task 293).
Consistency and formatting edits	Standardizing date formats, ensuring consistent phrasing, and adding URL parameters to control the number of items displayed for a uniform evaluation environment.	Standardized proximity phrasing by replacing "around" with "**near**" (Task 378).

A.3 OSWorld Evaluation

- **Description**: 369 tasks on Ubuntu Desktop. They involve different applications: LibreOffice Calc, LibreOffice Impress, LibreOffice Writer, Thunderbird, Gimp, OS, Chrome, VS Code, VLC. The task can involve several applications at the same time.
- **Task types**: document editing, image manipulation, email, file management, settings modifications, data analysis, coding, etc. The complexity of the tasks varies significantly (10-100 steps).
- **Evaluation**: Programmatic checks of final file/application state.
- **Environment**: AWS-hosted Ubuntu VMs, 1920x1080, VNC control.

A.4 AndroidWorld Evaluation

- **Description**: 116 tasks on an Android Emulator (Pixel 6 device). Tasks are specified as parameterized templates instantiated at runtime, requiring the agent to interact with apps such as Markor (notes), VLC (video), OpenTracks (activity tracker), Simple Calendar, Tasks.org, SMS, Contacts, Files, Camera, Audio Recorder, OsmAnd (maps), Retro Music Player, Recipe and Expense managers, Clock, and system settings, emphasizing multi-app, multi-steps workflows.

- **Task types**: Data entry/edit, information retrieval and search/navigation, screen reading and transcription, math/counting, verification, memorization/repetition, multi-app workflows, complex UI understanding, parameterized inputs, requires setup, game playing.
- **Evaluation**: Programmatic checks of final device and application state via the Android Debug Bridge. For information retrieval tasks, validation through exact or fuzzy matching. Each task is assigned a score of 1, 0.5, or 0 for success, partial success, or failure, respectively.
- **Environment**: Dockerized Android emulator (Pixel 6 device, 1080×2400 pixels, API level 33). Due to SIM card constraints, SMS-related tasks were executed outside the Docker containers.

B Examples

B.1 OSWorld Example

To illustrate Surfer 2 's adaptive behavior in desktop environments, we highlight one notable case where the agent successfully solved a task labeled as infeasible by human evaluators. For instance, it successfully changed Chrome's interface language to Korean–a task deemed impossible due to the absence of a visible language selector (see Figure 8). Instead of reporting failure, Surfer 2 opened a terminal, executed system commands via visual interaction, and relaunched Chrome, showcasing adaptive reasoning beyond predefined task boundaries through the integration of interface understanding and system-level knowledge.

B.2 AndroidWorld Examples

To contextualize aggregate metrics, we highlight one complex success (Figure 9a) and one representative failure (Figure 9b).

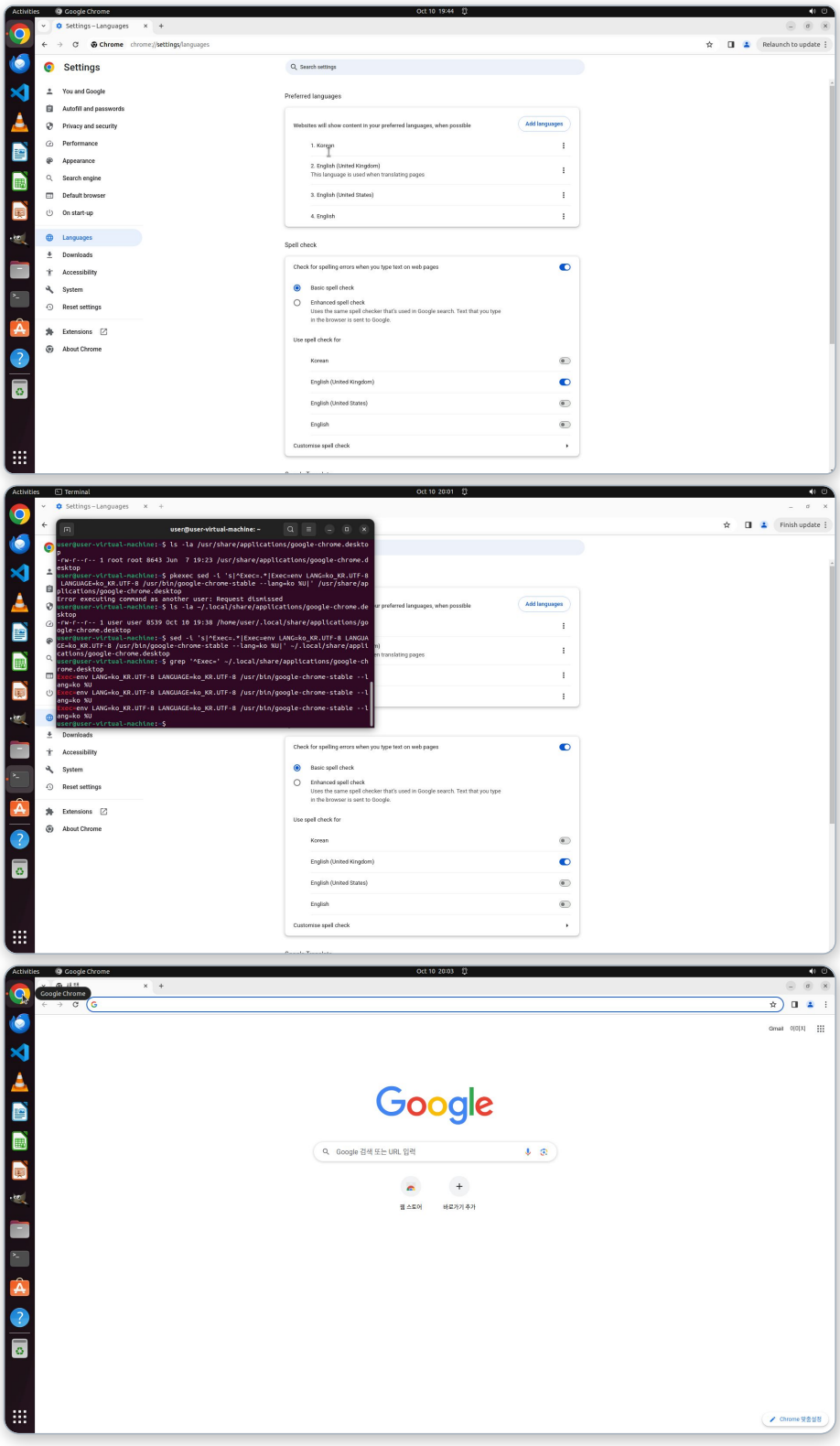
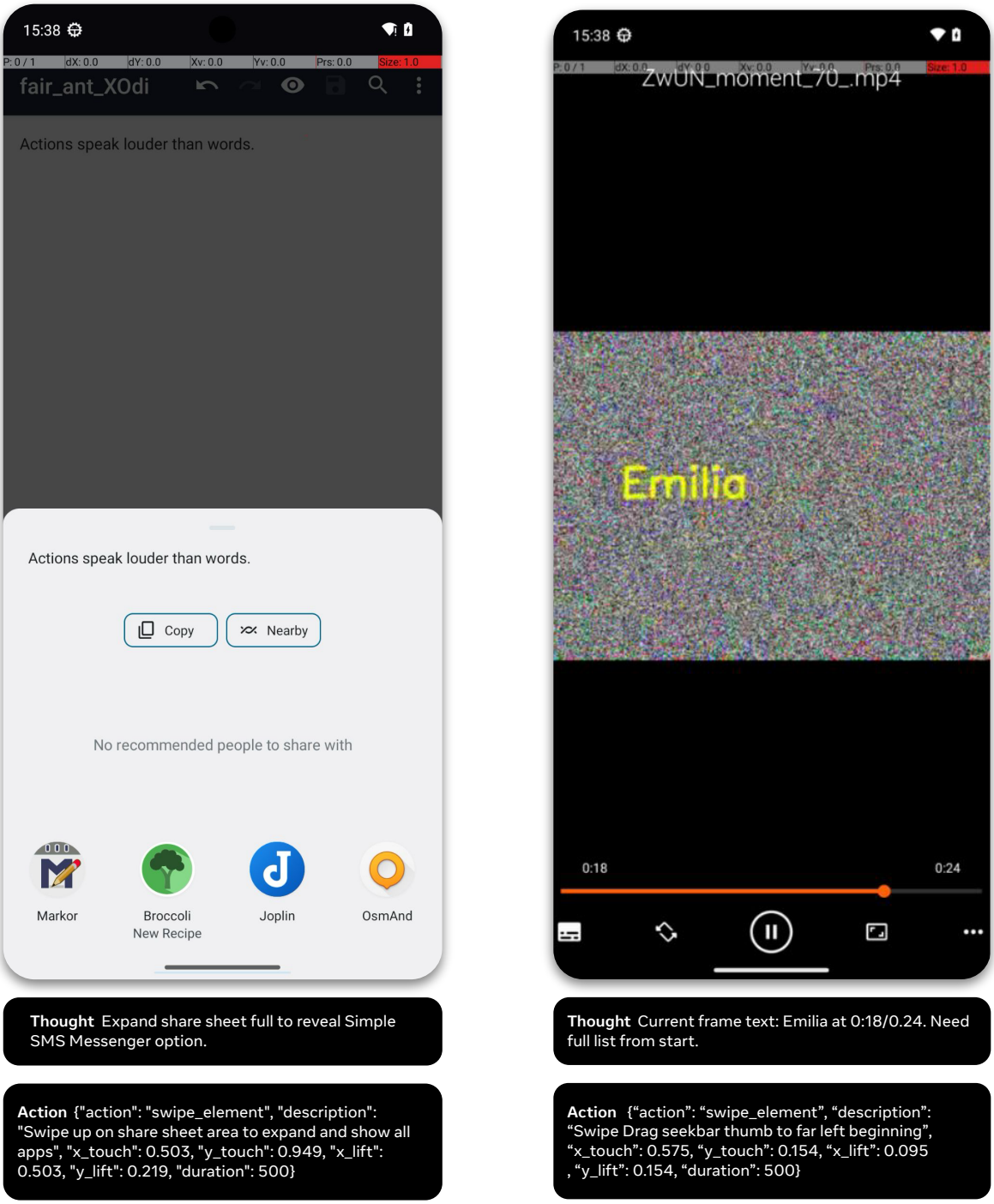


Figure 8: Surfer 2 switching Chrome's language to Korean via the terminal.



(a) **(Success) MarkorCreateNoteAndSms**: The agent creates a new note in Markor with the text *"Actions speak louder than words."*, then shares it via Simple SMS Messenger. This requires cross-app coordination (Markor → SMS), which Surfer 2 executes by creating the file, using the share action, and selecting the SMS app.

(b) **(Failure) MarkorTranscribeVideo**: The agent transcribes per-frame text from a VLC video into a Markor note. Despite reaching the correct context, limited temporal memory and sparse frame capture result in partial transcription and omissions, revealing difficulties in maintaining continuity over extended visual sequences.

Figure 9: Representative success and failure cases on AndroidWorld. Surfer 2 demonstrates robust cross-app reasoning but still struggles with long-horizon temporal perception.