Published by

British Journal of Contemporary Research

BEXFORD PUBLISHING

Development of an Intelligent Cloud-Deployed Web Browsing Agent

Original Research Article | Volume 1 | Issue 6 | 2026 | Article Number: 047

Accepted: 28 October 2026 | Published: 10 November 2026 | ISSN: 2979-8582 (Online)

Crossref : **https://doi.org/10.67693/BJCR-5D2BWIUR**

Olugbenga A. Madamidola[1], Oluwatobi S. Adebowale[1], Kolawole A. Sadiq[2], Abiola E. Fadojutimi[3]

[1]Department of Information Technology, The Federal University of Technology, Akure, Ondo State, Nigeria,

[2]Department of Cyber Security, Osun State University, Osogbo, Osun State, Nigeria,

[3]Department of Computer Science, The Federal University of Technology, Akure, Ondo State, Nigeria

OAM 0000-0001-5991-4067, KAS 0000-0001-5423-007X, AEF 0009-0004-9745-980X

Correspondence: Olugbenga A. Madamidola, oamadamidola@futa.edu.ng

Abstract

Web automation supports many administrative, academic, and commercial workflows, yet scripts built on DOM selectors break easily on single-page applications, hashed class names, and inconsistent accessibility markup. Vision-Language Models (VLMs) combined with Set-of-Mark (SoM) prompting let an agent act on screenshots and select elements by numbered reference, but most existing visual agents are single-user research prototypes tested on sanitised benchmarks, or commercial services whose safety checks depend on the model's own judgement and are exposed to indirect prompt injection. This study presents the development of a cloud-deployed vision-based web agent that combines a serverless FastAPI orchestration engine hosted on Modal, a Next.js web interface deployed to Vercel, in-container headless Chromium controlled through the Chrome DevTools Protocol, password-gated multi-tenant sessions, and a deterministic Human-in-the-Loop (HITL) safety gate whose risk classification cannot be lowered by any model output. The system is evaluated on four production websites (Google Fonts, Jiji Nigeria, Jumia Nigeria, and 10minutemail) using a four-tier test pyramid of 77 tests, with five fault types injected mid-execution. Empirical evaluation shows a 93% end-to-end resilience pass rate (14 of 15 runs), 100% recovery for every injected fault type, zero hangs and zero crashes across 52 execution steps, and a total model cost of approximately one US cent using Gemini 2.5 Flash-Lite, demonstrating that vision-based web agents with programmatic safety enforcement can operate reliably as cloud-hosted multi-tenant services.

Keywords: Web Agents, Vision-Language Models, Set-of-Mark Prompting, Browser Automation, Cloud Computing

© 2026 The Authors. Published by Bexford Publishing Ltd

1. INTRODUCTION

Web automation has become an essential part of digital operations, allowing individuals and organisations to complete multi-step workflows across administrative, academic, financial, and commercial platforms with little manual effort. Conventional automation relies on scripts that parse the Document Object Model (DOM) and locate elements through CSS selectors or XPath expressions (He *et al.*, 2024). Modern web frameworks, however, use virtual DOMs, single-page application routing, and generated class names that change with almost every deployment, so these selectors become obsolete quickly. The outcome is broken workflows, high maintenance cost, and automation that does not scale across platforms (Zhou *et al.*, 2024; Zheng *et al*., 2024).

The problem is more severe on regional infrastructure. In Nigeria, institutional platforms such as university portals feature fragmented user interfaces, nested dynamic dropdowns, legacy visual styles, and inconsistent DOM trees, and minor front-end updates are enough to invalidate pre-configured selectors (Gur *et al*., 2023). Existing evaluation environments such as WebArena (Zhou *et al.*, 2024), VisualWebArena (Koh *et al.,* 2024), and Mind2Web (Deng *et al*., 2023) use standardised, simulated sites that do not reflect this reality. They also measure whether a task is completed, not how an agent behaves when the browser tab closes, the model API stalls, the provider rate-limits requests, or the user interrupts with a new instruction, all of which are routine in production cloud environments.

Vision-Language Models (VLMs) offer a different approach. A VLM processes raw screenshots and recognises interface elements by their appearance, so it can identify a checkout button without depending on the underlying markup (Gur *et al*., 2023). Set-of-Mark (SoM) prompting further improves visual grounding by overlaying numbered boxes on interactive elements, allowing the model to select a target by reference number instead of predicting continuous pixel coordinates (Yang *et al.*, 2023). General-purpose computer-use models (Anthropic, 2024; OpenAI, 2025) and open-source platforms such as Browser-use (Müller & Žunič, 2024) show that such agents can run as managed services, but they rely on model-judged safety checks that a hostile webpage can subvert through indirect prompt injection.

The aim of this study is to develop, deploy, and empirically evaluate a cloud-deployed vision-based web agent capable of autonomously navigating and automating real-world workflows on dynamic websites, with programmatic safety enforcement and measured execution resilience under fault injection. The specific objectives of the study are to: (i) design a cloud-deployed, multi-tenant vision-based web agent architecture comprising a serverless orchestration engine, in-container headless browser orchestration, HTTP-served screenshot persistence, and password-gated authentication; (ii) implement the designed system with an accessibility-infused Set-of-Mark visual grounding layer, a deterministic Human-in-the-Loop safety gate that cannot be overridden by model output, a schema-repair loop for malformed model outputs, and persistent audit logging; and (iii) evaluate the execution resilience of the system on real-world production websites under five fault types using a four-tier test pyramid of 77 tests.

The remainder of this paper is organised as follows. Section 2 describes the literature review. Section 3 presents the methodology. Section 4 discusses the results, and Section 5 concludes the paper.

2. LITERATURE REVIEW

A review of research on autonomous web agents, visual grounding, and agent evaluation informs this work. Early web agents parsed HTML directly, but the field has moved towards multimodal models that read the rendered page in the same way a human user does. Two developments made this possible: Vision-Language Models capable of reasoning over screenshots, and grounding techniques that convert the model's intent into a precise on-screen target (Yang *et al.,* 2023; Zheng *et al.*, 2024). Empirical studies

© 2026 The Authors. Published by Bexford Publishing Ltd

have repeatedly shown that visually grounded agents cope better with unseen and dynamic interfaces than agents restricted to the DOM (He *et al*., 2024; Koh *et al*., 2024).

He *et al*. (2024) proposed WebVoyager, an end-to-end agent that uses GPT-4V to process raw page screenshots and generate sequential action plans, and demonstrated strong zero-shot generalisation across real-world websites. Zheng *et al.* (2024) showed that a grounded GPT-4V can match or exceed HTML-parsing approaches as a generalist web agent. Gur *et al.* (2023) combined an LLM planner with long-context understanding and program synthesis to complete long-horizon tasks on live sites, although its reliance on verbose HTML input made the pipeline prone to context-window bloat. Yang *et al.* (2023) introduced Set-of-Mark prompting, which overlays numbered identifiers on candidate targets and substantially reduces spatial coordinate errors, while Cheng *et al.* (2024) pre-trained SeeClick on GUI grounding data to improve element localisation, with weaker results on long-horizon, multi-turn planning.

WebArena (Zhou *et al.,* 2024), VisualWebArena (Koh *et al.*, 2024), and Mind2Web (Deng *et al*., 2023) established reproducible benchmarks for web agents, but all three rely on sanitised or self-hosted environments and measure task completion only. Commercial computer-use systems (Anthropic, 2024; OpenAI, 2025) and open-source cloud platforms such as Browser-use (Müller & Žunič, 2024), Skyvern (Skyvern-AI, 2024), and Stagehand (Browserbase, 2024) have moved visual agents into production use. Importantly, these systems assign risk to an action using the model's own judgement, which leaves them structurally vulnerable to indirect prompt injection from hostile webpages.

Despite these advances, gaps remain: evaluation on pristine benchmarks rather than live regional websites, no characterisation of agent behaviour under mid-run faults, safety checks that the model itself can override, and few open, multi-tenant cloud architectures with session isolation. This paper addresses these gaps by combining SoM-based visual grounding with a deterministic, non-overridable risk gate in a serverless multi-tenant deployment, and by evaluating the resulting system under fault injection on real production websites.

3. METHODOLOGY

This study uses a build-and-evaluate engineering methodology to design, implement, deploy, and assess a cloud-deployed vision-based web agent. The research design combines a targeted review of vision-language models, browser automation protocols, and cloud-native agent architectures with iterative system construction and empirical fault-injection testing. System design, visual preprocessing, model integration, safety enforcement, and resilience evaluation are all part of the methodology.

Figure 1 shows the architectural layout of the proposed system, which separates the runtime into presentation, orchestration, actuation, and persistence tiers. The presentation tier is a Next.js 15 web interface compiled as a static export and served from Vercel's edge network, with a password gate, a route guard, and a session token held in browser storage. The orchestration tier is a Python 3.12 FastAPI service running in a Modal serverless container (4 GB memory, 2 CPUs, up to four concurrent tasks per container) that hosts the authentication and session store, the owner-scoped task registry, the screenshot server, and the agent loop. The actuation tier runs in the same container, where the agent-browser Rust binary drives a headless Chromium instance over the Chrome DevTools Protocol (CDP) on port 9222. Screenshots are written to a persistent Modal Volume under an owner- and task-scoped path and served over HTTP, while the shared password and provider API keys are injected through Modal Secrets.

© 2026 The Authors. Published by Bexford Publishing Ltd

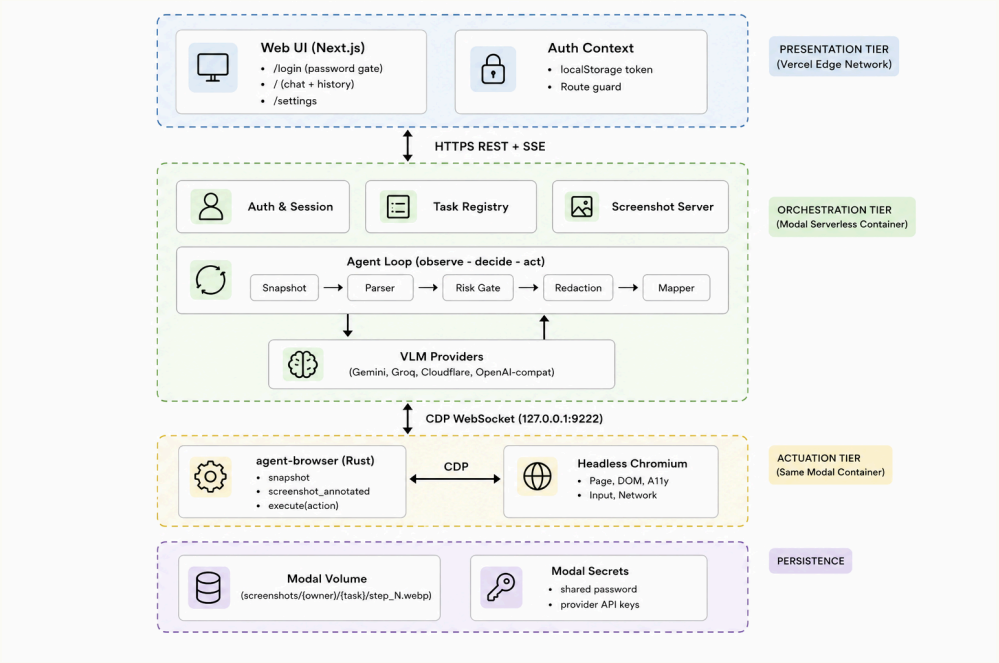


Figure 1: System Architecture of the Cloud-Deployed Web Browsing Agent

Figure 2 shows the observe-decide-act loop executed for every task. At each step, the orchestrator captures the viewport and the browser's accessibility tree, prunes the tree to interactive roles, overlays numbered Set-of-Mark labels, and compresses the screenshot to WebP (maximum side of 1,280 pixels at quality 75). The annotated snapshot is sent to the VLM through a provider-agnostic ModelClient interface, with automatic fallback between Google Gemini and OpenRouter on rate-limit errors. The model's reply is parsed through a strict Pydantic v2 schema that permits only click, type, scroll, wait, navigate, and done actions, and malformed output triggers a repair prompt instead of a crash. Each validated action is then re-classified by deterministic policy rules, and the effective risk is taken as the maximum of the model-reported and policy-calculated risk. High-risk actions are held until the user approves them in the web interface within 300 seconds.

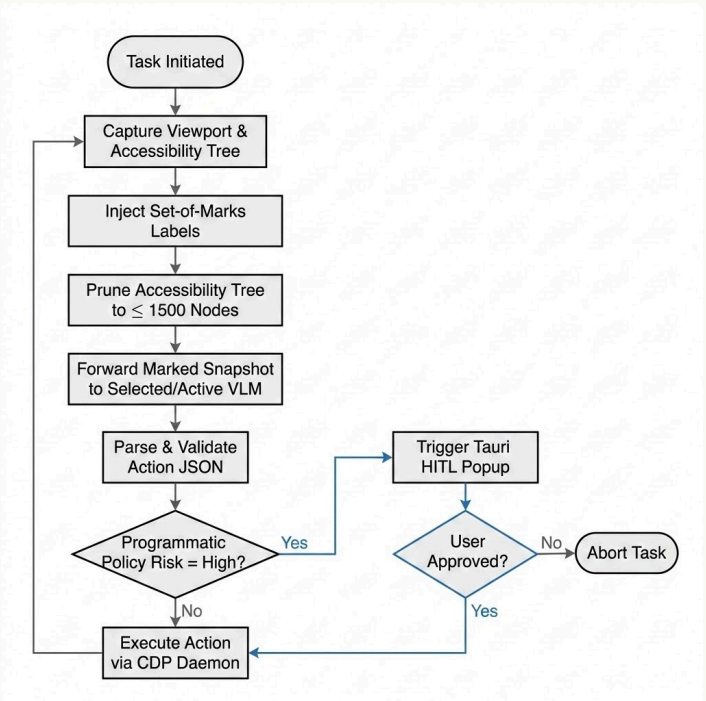


Figure 2: Observe-Decide-Act Execution and Safety Flowchart

© 2026 The Authors. Published by Bexford Publishing Ltd

The orchestrator was implemented in Python 3.12 with FastAPI, Uvicorn, Pydantic v2, httpx, aiolimiter, and Pillow, and the web interface in TypeScript with React 19 and Next.js 15, organised as a polyglot monorepo whose shared JSON Schema contracts are compiled into both languages. Evaluation uses the Resilience Pass Rate (RPR), the proportion of runs that satisfy all five invariant classes (termination, status validity, forward progress, resource bounds, and protocol correctness), the Fault Recovery Rate (FRR) for each fault type, and the total model cost of the evaluation.

4. RESULTS

This section presents the findings from the experimental assessment of the developed web agent. The system under evaluation integrates SoM-based visual grounding, a multi-provider VLM layer, and a deterministic HITL safety gate within a serverless orchestration engine. End-to-end runs used Google Gemini 2.5 Flash-Lite through OpenRouter, Google Chrome 150 with remote debugging on port 9222, and the FastAPI orchestrator executed in-process, with pytest 9.0.3 and a custom resilience harness on Windows 11 and Python 3.12.11. Each run was bounded by a 600-second wall-clock timeout and took between 8 and 20 steps depending on task complexity.

4.1 TEST SUITE AND FAULT INJECTION DESIGN

The system was validated through a four-tier test pyramid in which each tier targets a different class of failure. Unit and contract tests verify internal logic and the API surface against faked browser and model clients, resilience tests verify loop termination under faked faults, and end-to-end (E2E) tests exercise the full stack with a real model on real websites. Table 1 summarises the four tiers.

Table 1: Composition of the Four-Tier Test Pyramid

Tier	Scope	Dependencies	Tests
Unit	Action schema, mapper, risk classifier, parser, redaction, providers	None (pure logic)	17
Contract	API surface, SSE streaming, approval gating, sessions, agent loop lifecycle	In-memory FastAPI app with faked browser and model	27
Resilience	Loop termination under browser failure, model failure, malformed output, repeated actions, and abort	Fault-injecting fakes	18
End-to-end	Full stack on Google Fonts, Jiji, Jumia, and 10minutemail	Real model and real websites	15
Total			77

The 15 E2E scenarios were split into three groups: three baseline tasks without faults (searching for a font on Google Fonts, an iPhone 14 Pro on Jiji, and a power bank on Jumia), ten fault-injection runs, and two long-horizon workflows (a 15-step price analysis on Jiji and a 20-step 10minutemail-to-Jumia signup, search, cart, and checkout flow). Each fault was injected at step 3 or step 5 while the agent was actively executing. Table 2 lists the five fault types.

© 2026 The Authors. Published by Bexford Publishing Ltd

Table 2: Fault Types Injected during End-to-End Evaluation

Fault Type	Injection Mechanism	Expected Behaviour
User abort	POST /v1/tasks/{id}/abort	Terminate with status aborted and release all resources
Chrome tab closure	CDP PUT /json/close/{targetId}	Re-attach to a new tab, or terminate cleanly with status failed
Model hang	30-second sleep in the model client	Tolerate the delay and continue, or end cleanly if the step budget is exhausted
API rate limiting (HTTP 429)	ProviderError("rate_limited") raised by the model client	Back off, fall back to another provider, or fail cleanly
User steering	POST /v1/tasks/{id}/message with a new instruction	Add the instruction to history and continue with the updated goal

4.2 SAFETY GATE VALIDATION

The deterministic risk classifier was validated with seven units' tests covering every trigger rule and the preservation of minimal risk for benign actions. All seven tests passed in 1.48 seconds (Table 3). An action targeting a button named “Delete account” was promoted from low to high risk regardless of the model's self-report, which is the behaviour required to resist indirect prompt injection.

Table 3: Risk Classification Rules and Unit Test Validation

Trigger Rule	Description	Test Case	Status
High-risk keyword match	Delete, pay, confirm, or submit in accessible name	test_delete_button_text_promotes_to_high	Pass
Disallowed domain navigation	Navigate to host outside allowed_domains list	test_navigate_to_disallowed_host_is_high	Pass
Allowed domain navigation	Navigate to host within allowed_domains list	test_navigate_to_allowed_host_keeps_low	Pass
Financial form detection	Node flagged in_form_with_money_or_account_action	test_financial_form_promotes_to_high	Pass
Password field typing	Type action on node with is_password_or_otp	test_password_field_type_is_high	Pass

© 2026 The Authors. Published by Bexford Publishing Ltd

File upload interaction	Node with tag=input and input_type=file	test_file_upload_is_high	Pass
Stale snapshot state	Observation hash changed between capture and action	test_stale_snapshot_promotes_to_high	Pass

When an action is promoted to high risk, the orchestrator blocks it before it reaches the browser, captures a crop of the target element, and streams an approval request to the web interface. Figure 3 shows the approval modal raised when the agent attempted to remove an item from a Jumia shopping cart; the action is executed only after the user selects Approve.

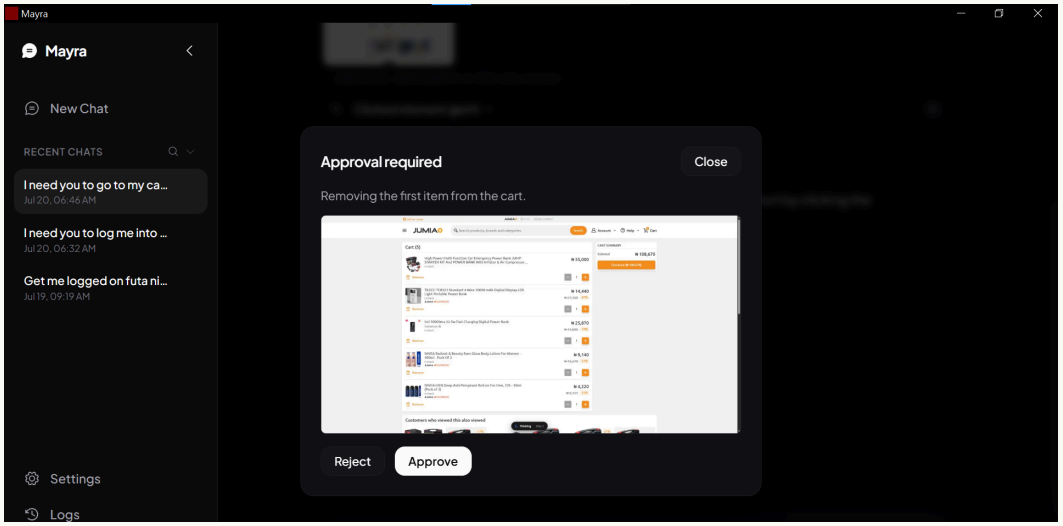


Figure 3: Approval Modal Raised by the HITL Gate for a High-Risk Action

4.3 VISUAL GROUNDING AND SYSTEM OPERATION

Figure 4 shows the Set-of-Mark overlay applied to the FUTA student portal login page. Each interactive control identified in the pruned accessibility tree receives a numbered box, so the model selects a target such as the matriculation number field or the login button by its reference number rather than by pixel coordinates or a CSS selector.

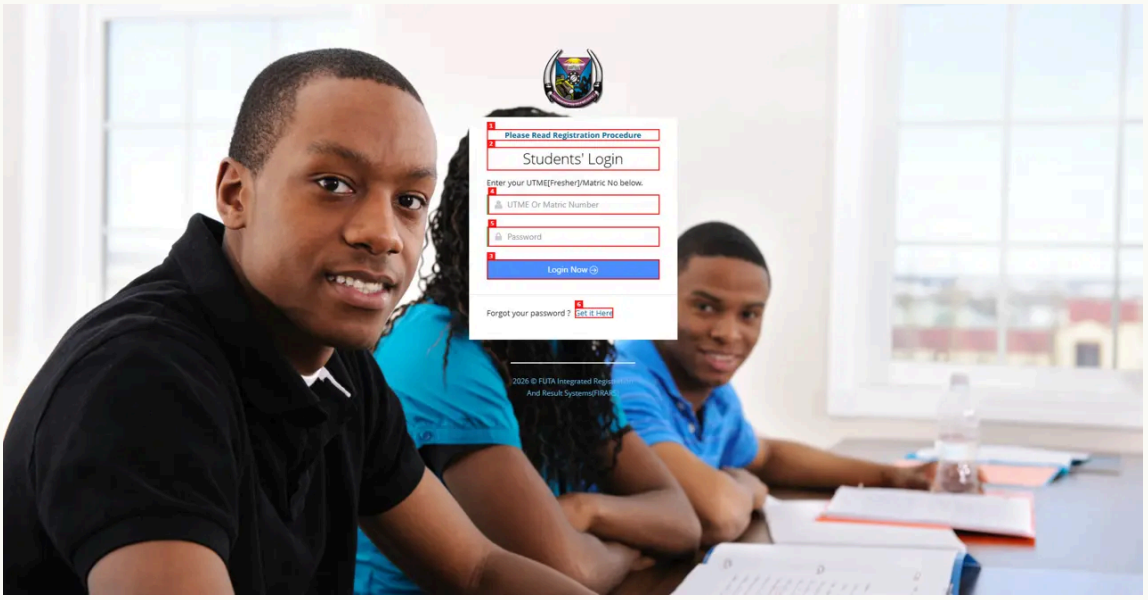


Figure 4: Set-of-Mark Overlays on the FUTA Student Portal Login Page

© 2026 The Authors. Published by Bexford Publishing Ltd

Figure 5 shows the web interface during a live task. The agent's reasoning is streamed to the user token by token, each step is logged with the executed action and its element reference, and the screenshot captured at that step is displayed as a thumbnail served from the orchestrator's screenshot endpoint. The composer remains active during execution, so the user can steer or stop the agent at any point.

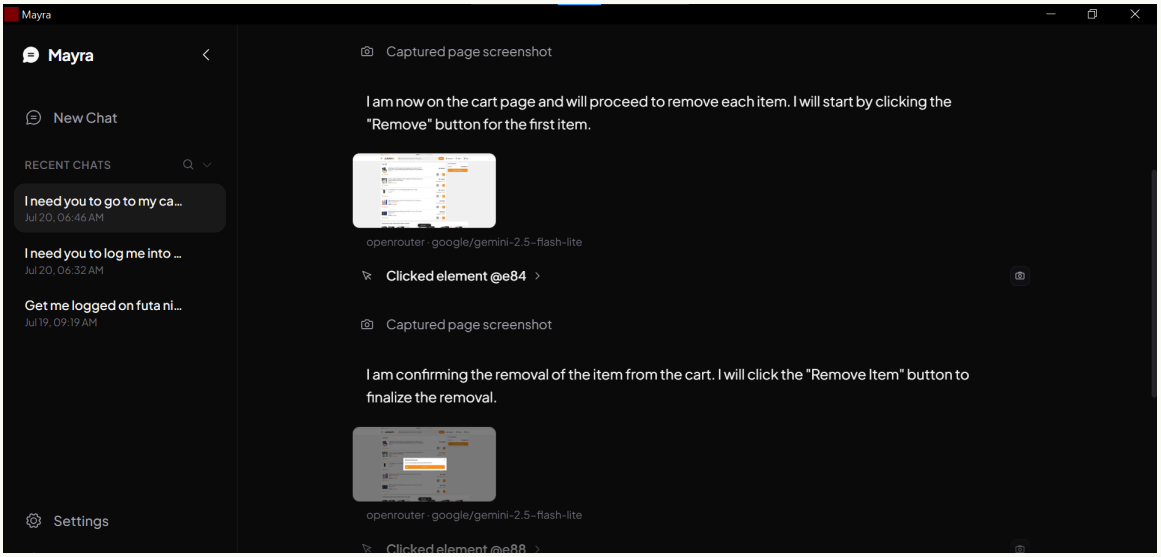


Figure 5: Web Interface Streaming Agent Reasoning, Actions, and Step Screenshots

4.4 END-TO-END RESILIENCE RESULTS

A key test of a cloud-deployed agent is whether it terminates cleanly, recovers from faults, and avoids leaking resources when it operates without supervision. Table 4 summarises the end-to-end results, and Figure 6 shows the outcomes across all four tiers of the test pyramid.

Table 4: End-to-End Resilience Evaluation Results

Metric	Result
Resilience Pass Rate (RPR)	93% (14 of 15 runs)
Fault Recovery Rate (each of the five fault types)	100%
Websites with 100% pass rate	3 of 4
Hangs	0
Crashes	0
Total execution steps	52
Critical bugs found and fixed	6
Total model cost (Gemini 2.5 Flash-Lite)	Approximately US$0.01

© 2026 The Authors. Published by Bexford Publishing Ltd

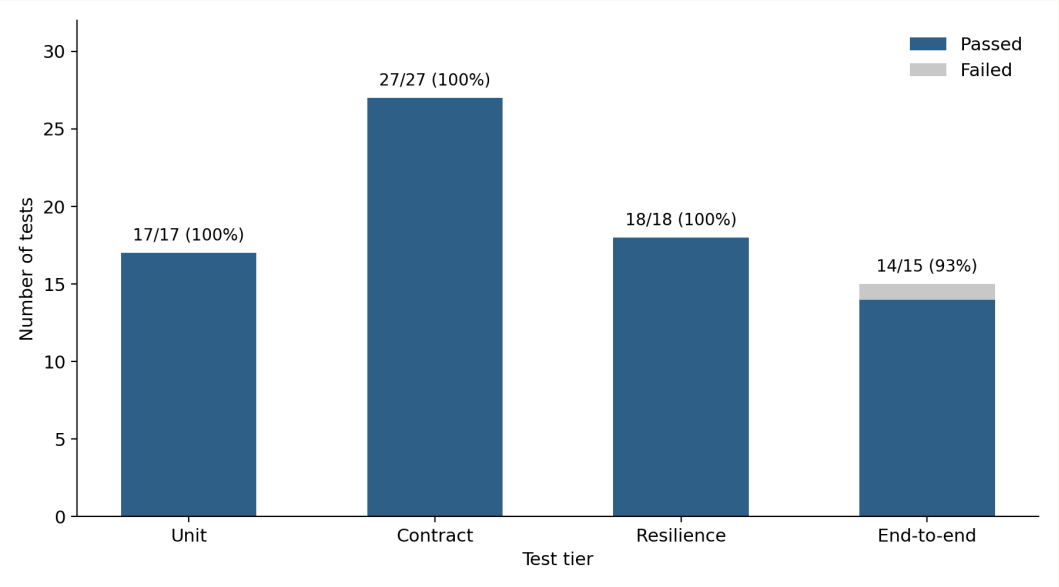


Figure 6: Test Outcomes across the Four Tiers of the Test Pyramid

The agent survived every injected fault. Aborts, tab closures, model hangs, rate limits, and mid-run steering all ended in a valid terminal state, with zero hangs and zero crashes across 52 execution steps. Iterative E2E testing also exposed six critical bugs that the unit and contract tiers could not detect, including an infinite click loop on Jumia product pages and an unreachable task-completion signal in the action schema, and these were fixed before the final runs. The total model cost of approximately one US cent shows that a vision-based agent built on a small model is affordable for routine use.

5. CONCLUSION

The results confirm that the developed system meets the objectives of the study, achieving a 93% end-to-end resilience pass rate and full recovery from every injected fault type on real production websites. The main contribution of this study is the programmatic safety gate, rather than the task results alone. Existing visual agents and commercial computer-use systems rely on model-judged risk, which a hostile webpage can manipulate through indirect prompt injection (Anthropic, 2024; OpenAI, 2025). By computing risk from fixed policy rules and taking the maximum of the policy and model risk, the system ensures that no model output can lower the risk of a destructive action. This design directly fills that research gap.

The fault-injection methodology also addresses a gap in how web agents are evaluated. Benchmarks such as WebArena and VisualWebArena measure task completion in sanitised environments, whereas this study measured termination, status validity, forward progress, resource bounds, and protocol correctness under the faults that occur routinely in cloud deployment. Together with owner-scoped session isolation, password-gated authentication, and HTTP-served screenshot persistence, these results indicate that vision-based web agents can move from single-user research prototypes to multi-tenant cloud services, including for workflows on regional websites where DOM-based scripts fail.

Future work should extend fault-injection testing to longer tasks, higher concurrency, and additional faults such as network partitions and container restarts, add long-term memory with context compaction, explore edge-hosted small vision models to reduce cloud API dependency, establish standardised benchmarks on legacy regional portals, and apply formal verification to the safety policy so that the blocking of destructive actions can be proven rather than only tested.

© 2026 The Authors. Published by Bexford Publishing Ltd

References

Anthropic. (2024). Developing computer use. Anthropic Research Blog. https://www.anthropic.com/news/developing-computer-use.

Browserbase. (2024). Stagehand [Computer software]. GitHub. https://github.com/browserbase/stagehand.

Cheng, K., Sun, Q., Chu, Y., Xu, F., Li, Y., Zhang, J., & Wu, Z. (2024). SeeClick: Harnessing GUI grounding for advanced visual GUI agents. arXiv preprint arXiv:2401.10935. https://arxiv.org/abs/2401.10935.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., & Su, Y. (2023). Mind2Web: Towards a generalist agent for the web. Advances in Neural Information Processing Systems, 36. https://arxiv.org/abs/2306.06070.

Gur, I., Furuta, H., Huang, A., Safdari, M., Matsuo, Y., Eck, D., & Faust, A. (2023). A real-world WebAgent with planning, long context understanding, and program synthesis. arXiv preprint arXiv:2307.12856. https://arxiv.org/abs/2307.12856.

He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., & Yu, D. (2024). WebVoyager: Building an end-to-end web agent with large multimodal models. arXiv preprint arXiv:2401.13919. https://arxiv.org/abs/2401.13919.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M. C., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., & Fried, D. (2024). VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. arXiv preprint arXiv:2401.13649. https://arxiv.org/abs/2401.13649.

Müller, M., & Žunič, G. (2024). Browser Use: Enable AI to control your browser [Computer software]. GitHub. https://github.com/browser-use/browser-use.

OpenAI. (2025). Computer-using agent. OpenAI. https://openai.com/index/computer-using-agent/.

Skyvern-AI. (2024). Skyvern [Computer software]. GitHub. https://github.com/Skyvern-AI/skyvern.

Yang, J., Zhang, H., Li, F., Zou, X., Li, C., & Gao, J. (2023). Set-of-Mark prompting unleashes extraordinary visual grounding in GPT-4V. arXiv preprint arXiv:2310.11441. https://arxiv.org/abs/2310.11441.

Zheng, B., Gou, B., Kil, J., Sun, H., & Su, Y. (2024). GPT-4V(ision) is a generalist web agent, if grounded. arXiv preprint arXiv:2401.01614. https://arxiv.org/abs/2401.01614.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U. & Neubig, G. (2024). WebArena: A realistic web environment for building autonomous agents. International Conference on Learning Representations (ICLR). https://arxiv.org/abs/2307.13854.

© 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the **Creative Commons Attribution (CC BY 4.0) licence** (https://creativecommons.org/licenses/by/4.0/)