WEBSITEBENCH: CAN AI AGENTS REBUILD WEBSITES THROUGH BROWSER EXPLORATION?

Anonymous authors
Paper under double-blind review

ABSTRACT

Humans learn how unfamiliar environments work by observing them, trying different actions, and revising their understanding based on what happens. Websites provide a concrete setting for studying this process: they combine interfaces, application logic, and persistent data, while each browser interaction reveals only part of their behavior. Can AI agents decide what to explore and use what they learn to rebuild a website faithfully? We introduce WEBSITEBENCH, which contains 74 expert-built and expert-reviewed website reconstruction tasks and 50,356 hidden tests. In each task, an agent explores a locally hosted website through a browser and builds a runnable offline site that looks and behaves the same way. The agent decides which pages to visit and which interactions to try, and may revisit the reference website while coding and testing. The hidden test suite consists of three groups: Request checks page responses and form submissions; Browser checks what users see and can do; and System checks application requirements. Our experiments show that WEBSITEBENCH is challenging for frontier models: even the top model, Claude Opus 5.5, scores only 22.5%; GPT 5.6 Sol scores 15.6%, and Kimi K3 scores 8.9%. We systematically analyze agent trajectories and identify three failure modes: (1) underexploration, (2) failure to implement observed behavior, and (3) early stopping. We also find that agents given human exploration traces achieve higher reconstruction scores. These results show that agents struggle to learn how an unfamiliar system works through interaction and to use that understanding to rebuild it faithfully.

1 INTRODUCTION

"What I cannot create, I do not understand." — Richard Feynman

Newton's observation of a falling apple raised a question about the rule behind its motion. Further observation and exploration helped turn that question into a general physical law. Feynman's remark suggests a complementary test of understanding: building something that reproduces what we observe. Differences between what we build and what we observe can reveal gaps in our understanding and guide further exploration. Can AI agents similarly learn about an unfamiliar environment through interaction and use what they learn to recreate it?

Websites provide a concrete setting for this question: an agent can see the interface, but must interact with it to learn how it behaves. Their appearance reveals only part of how they work. A registration form displays its fields and buttons, but interaction is needed to discover which inputs are accepted and whether the resulting account can be used later. Other behaviors depend on the current user, prerequisite actions, or changes made on another page. An agent must choose which paths and states to inspect without knowing in advance what it has missed. It must then reproduce the connections it discovers: a form submission may update a record, change another page, and affect what a different user sees. Reproducing each page in isolation does not reproduce these dependencies.

Prior benchmarks study agents using websites and agents building them. Web-agent benchmarks ask agents to complete specified goals on existing websites (Garg et al., 2025; Zhou et al., 2024). Website-generation benchmarks ask agents to build from supplied screenshots, sketches, visual prototypes, or written requirements (Si et al., 2025; Li et al., 2024; He et al., 2026). These benchmarks

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

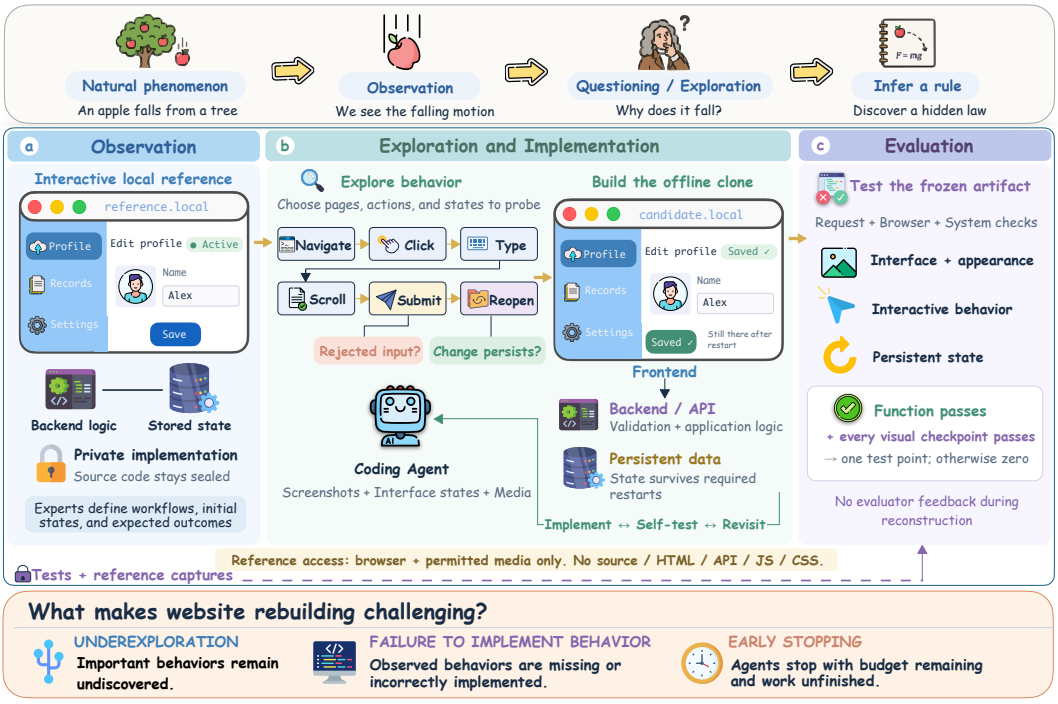


Figure 1: Overview of WEBSITEBENCH. Top: Observation prompts questions and exploration before a rule is inferred. (a) Website development experts build a locally hosted reference website; its source code is hidden from the agent. (b) The agent explores the reference in a browser, builds a runnable offline reconstruction, and may revisit the reference while coding and testing. (c) Hidden tests compare the reconstructed and reference websites on appearance, behavior, and persistent state.

do not require agents to choose what to explore on a working website and use what they discover to rebuild it as a runnable full-stack website.

We introduce WEBSITEBENCH, a benchmark comprising 74 full-stack website-reconstruction tasks across nine website categories (Table 3). The reference websites implement selected functionality of widely used web services and include interfaces, application logic, and persistent data. The corpus includes workflows spanning multiple pages, state that persists across restarts, and behavior that depends on the user. Table 7 compares WEBSITEBENCH with the benchmarks discussed above.

For each task, an agent explores the local reference through a browser and submits a codebase for deployment. Reconstruction can require hours of sustained work. The agent may revisit the reference while coding and testing. It must use observations of pages, interactions, and state changes to decide what to implement. As it revises its code, it must test the local website against the reference, for example by repeating observed actions and comparing the resulting pages and state. Exploration, implementation, and local testing share a 16-hour limit. The reference source code, hidden tests, and evaluator feedback remain unavailable during reconstruction. After submission, hidden tests evaluate the resulting website's appearance, behavior, and persistent state, including responses to individual actions and outcomes of multi-step workflows.

Sixteen website development experts design and build the tasks under shared guidelines. For each website, they specify its pages, forms, controls, workflows, test inputs, and observable results. One expert implements the reference website; another independently reviews the implementation for correctness and workflow coverage, then develops the hidden tests. Before inclusion, the experts run each test against the reference and check that its assertions match the observed behavior. Figure 1 illustrates a WEBSITEBENCH task: website development experts build the reference, an agent explores it and submits a codebase, and hidden tests evaluate the website deployed from that codebase. Its bottom panel highlights three failure modes: underexploration, failure to implement observed behavior, and early stopping.

We study how agents explore the reference, whether they implement the behavior they observe, and when they stop. We also test whether human exploration trajectories improve scores and examine how scores vary across websites.

We evaluate nine foundation models on 74 website reconstruction tasks with 50,356 hidden tests. Claude Opus 5.5 achieves the highest score, 22.5%. Agents often explore part of a website but fail to reproduce its functional behavior and persistent state.

In 17 inspected trajectories, agents visited 97.6% of core page families and attempted 99.2% of core workflows before their first implementation edit, but reached terminal outcomes for only 3.3% of workflows. They also failed to implement most observed requirements that we could assess. In paired runs, agents given human exploration trajectories achieved higher reconstruction scores. Together, these results show gaps in both exploration and implementation. Additionally, some runs stopped with time remaining. In an in-depth analysis of 20 websites, deeper workflows were associated with lower reconstruction scores.

Our contributions are:

(1) We introduce WEBSITEBENCH, a benchmark combining active browser exploration with full-stack website reconstruction, featuring reference websites and evaluation tests designed, constructed, and reviewed by 16 website development experts.

(2) We develop a deterministic hidden-test protocol that evaluates interface fidelity, functional behavior, and persistent state through targeted checks and multi-step user journeys.

(3) We identify incomplete workflow exploration and failures to implement observed behavior, test whether human exploration trajectories improve scores, and examine early stopping and variation across websites.

2 WEBSITEBENCH

2.1 TASK DEFINITION

Each WebsiteBench task asks an agent to reconstruct a reference website through browser exploration. The reference is a locally hosted website built by our website development experts to reproduce selected functionality of a widely used web service. The static reference design incorporates the complexities of real-world websites while ensuring reproducibility for the benchmark and preventing reference drift. The observation interface and access restrictions are specified in Appendix B.

Figure 2 summarizes the task bundle, agent workspace, and isolated evaluation flow.

2.2 EXPERT CONSTRUCTION AND VERIFICATION

Our task inventory comprises 74 website reconstruction tasks developed by 16 website development experts. These experts design, construct, and verify both the reference websites and the evaluation test suites. The benchmark spans a diverse set of real-world domains, encompassing everyday web services such as e-commerce and retail platforms, productivity and SaaS tools, content and community portals, travel and reservation systems, marketplaces, educational platforms, and service-oriented applications.

For each reference website, website development experts construct a local self-contained implementation matching the selected pages and workflows from an existing web service. To ensure consistency among the corpus and correctness, all websites follow shared construction guidelines and a standardized project structure. After one website development expert has fully completed implementation, another expert independently reviews its technical correctness and behavioral coverage and match against the real-world web service.

Appendix A describes the construction and review procedures. Appendix D provides the test-suite validation details.

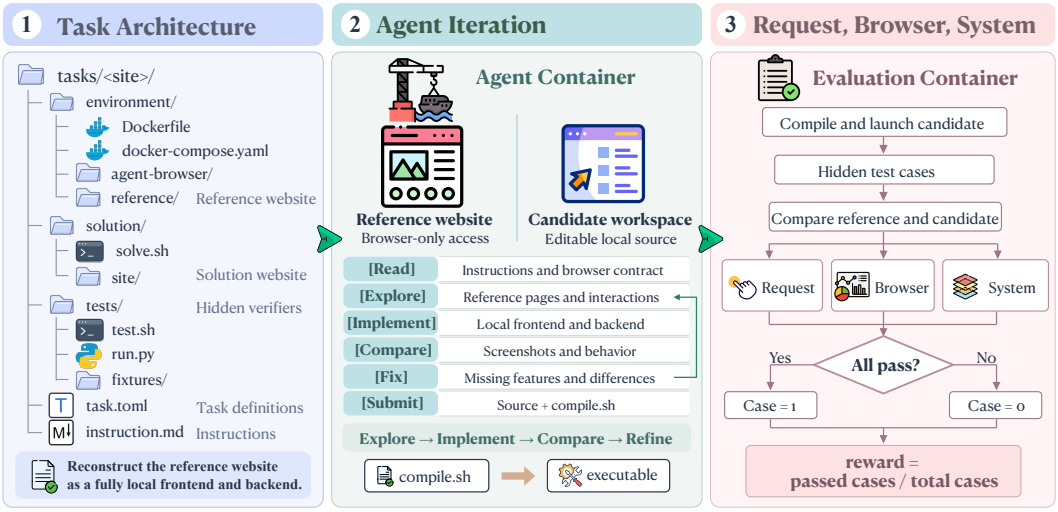


Figure 2: WebsiteBench task structure and evaluation pipeline. Left: schematic task-bundle organization, including reference assets, hidden verifiers, and agent instructions. Middle: the agent explores the reference website and iterates on a local reconstruction. Right: isolated offline evaluation produces binary test outcomes. See Section 2.4 for the formal score.

2.3 Test Coverage

To evaluate reconstruction correctness, the website development expert who reviewed the local reference implementation independently develops a suite of hidden tests. These tests evaluate behavioral fidelity while remaining strictly implementation-agnostic. All experts follow standardized guidelines to ensure consistency across tasks and comprehensive coverage over expected behaviors.

Evaluation tests are organized into three primary categories: (1) **Request (T1)** tests evaluate localized, unit-level behaviors via direct API calls or isolated element interactions, validating user-observable page responses and native form submissions; (2) **Browser (T2)** integration tests are executed via deterministic browser interactions and Playwright scripts to reliably simulate end-to-end user workflows; and (3) **System (T3)** tests evaluate system-level runtime requirements and operational constraints, including environment compilation, restart behaviors, persistent process health, and security integrity.

Within Browser (T2), tests cover immediate interface responses (L1), observable state changes (L2), and persistence, user isolation, or recovery after rejected actions (L3). Examples include input validation, reading back an edited record, and checking data after a restart.

Additionally, a subset of Browser test cases includes visual similarity checks that compare rendered screenshots of specified regions against the reference website, requiring at least 75% visual similarity to confirm UI fidelity.

To ensure test validity, all hidden tests are verified against their reference implementations, confirming a 100% pass rate with zero false negatives across the benchmark.

2.4 Evaluation and Scoring

After the agent completes the reconstruction task, the evaluation environment builds the submission offline and evaluates the resulting website against the hidden test suite. A test receives full credit ($p_i = 1$) only when its functional checks succeed and all specified visual checkpoints meet the similarity threshold $\tau = 0.75$. Formally, let $F_i \in \{0, 1\}$ denote functional success for test i, $\mathcal{V}_i$ the set of declared visual checkpoints, and $v_{ik} \in [0, 1]$ the normalized visual similarity of the target

region at checkpoint k:

$$p_i = F_i \prod_{k \in \mathcal{V}_i} \mathbf{1}[v_{ik} \geq \tau], \qquad S_{c,s,r} = \frac{100}{N_s} \sum_{i=1}^{N_s} p_i. \tag{1}$$

Here, N_s is the total number of tests for website s, while c and r denote the agent configuration and trial index, respectively. For tests without visual checkpoints ($\mathcal{V}_i = \emptyset$), the product evaluates to 1, and credit depends solely on functional success. All tests are weighted equally without partial credit.

To prevent code leakage and unauthorized reference copying, submissions that violate integrity constraints receive a score of zero.

Further details on build runtimes, screenshot comparison metrics, and scoring thresholds are provided in Appendices B.2 and D.

3 Experiments

3.1 Models and Experimental Settings

Models. We evaluate nine models on all 74 website reconstruction tasks: Claude-Opus-5.5 (Anthropic, 2026a), Claude-Opus-5 (Anthropic, 2026b), GPT-5.6-Sol (OpenAI, 2026), GPT-5.6-Luna (OpenAI, 2026), Grok-4.6 (Xai, 2026), Qwen-3.8-Max (Qwen Team, 2026), DeepSeek-V4.1-Flash (Deepseek, 2026), Kimi-K3 (Kimi, 2026), and GLM-5.3-Flash (Z.ai, 2026). We run each model with a fixed Claude Code harness under the same configuration. During execution, the agent's runtime trajectory and reconstruction artifacts are preserved as evidences for evaluation. Detailed experiment configurations are provided in Appendix E.

Websites and access. Each trial asks an agent to reconstruct a reference website through independent browser exploration. For each task, the agent is launched in a standard isolated environment, provided general instructions on the reconstruction task and a browser gateway for accessing the reference website yet prevented from directly accessing the reference source code.

Budgets and execution. Each trial has a 16-hour time limit to avoid hanging jobs. After completion, the agent is instructed to keep the reconstructed artifacts in the designated directory and provide an entrypoint matching the standard required in the initial instructions.

Once the agent completed its reconstruction, the generated artifacts will be copied into a separate testing environment where hidden tests are applied against the implementation, ensuring the evaluation process not influenced by the agents changes. Appendix E.

3.2 Evaluation Metrics

Overall reconstruction score. As defined in Section 2.4, each trial is scored by the percentage of hidden tests passed, with functional success and all declared visual checkpoints required for credit. Our primary aggregate pools tests across the common website set. Each eligible trial contributes its site's declared number of tests:

$$M_c = \frac{\sum_{s \in \mathcal{S}} N_s \sum_{r \in \mathcal{R}_{c,s}} P_{c,s,r}}{\sum_{s \in \mathcal{S}} N_s |\mathcal{R}_{c,s}|}. \tag{2}$$

Here $P_{c,s,r}$ is the score of trial r for model c on site s after applying copying disqualification, $\mathcal{R}_{c,s}$ is the set of eligible trials, N_s is the number of tests for site s, and $\mathcal{S}$ is the common 74-site set. Eligibility follows Appendix E: documented absent submissions receive zero, while infrastructure-invalid trials are excluded. We report M_c as *Overall*, with eligible zero-score trials included.

Performance breakdowns. We report test pass rates separately for Request, Browser, and System tests, using the binary test outcome defined in Equation 1.

Resource use. We report mean cost, token usage, and number of trace steps for with available usage records.

Table 1: Reconstruction performance and resource usage. Overall is the test-pooled score M_c across 74 websites. Overall and Request, Browser, System test values are percentages. Cost, tokens, and turns are averages per task. Tokens includes input and output tokens.

Model	Overall	Request	Browser	System	Cost	Tokens	Turns
Claude-Opus-5.5	**22.5**	**63.5**	**18.0**	34.8	**$261.1**	**930.5M**	**2107.1**
Claude-Opus-5	18.7	43.1	14.0	33.2	$221.7	429.2M	980.5
GPT-5.6-Sol	15.6	38.2	10.1	**37.4**	$115	20.5M	212.3
GPT-5.6-Luna	13.2	29.8	8.9	30.0	$3.8	14.3M	174.0
Grok-4.6	11.6	37.8	7.8	29.6	$76.9	38.0M	403.9
GLM-5.3-Flash	9.7	20.4	6.6	23.1	$2.4	37.0M	318.8
Kimi-K3	8.9	22.6	4.7	27.1	$37.2	91.6M	604.3
DeepSeek-V4.1-Flash	8.5	23.7	4.2	26.5	$4.6	167.8M	572.6
Qwen-3.8-Max	5.7	22.6	1.8	20.2	$9.5	38.3M	252.1

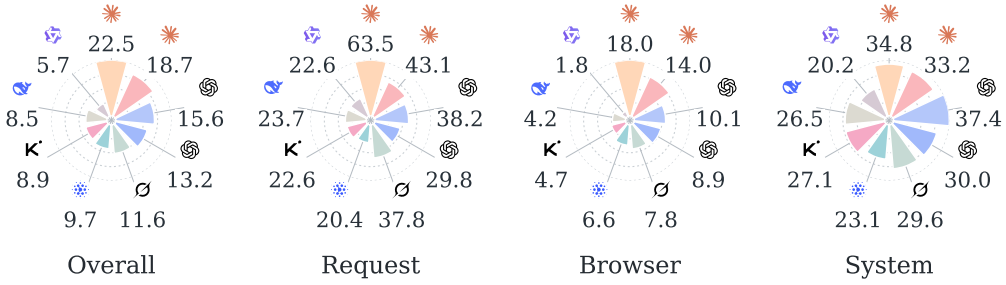


Figure 3: Evaluated models' performance on WEBSITEBENCH. Clockwise from the top: Claude Opus 5.5, Claude Opus 5, GPT 5.6 Sol, GPT 5.6 Luna, Grok 4.6, GLM 5.3 Flash, Kimi K3, DeepSeek V4.1 Flash, and Qwen 3.8 Max. Scores (%) match Table 1. Bars start at zero at the inner circle. Each panel is scaled independently to its highest score: 22.5%, 63.5%, 18.0%, and 37.4%, respectively.

4 RESULTS

4.1 OVERALL RECONSTRUCTION PERFORMANCE

Across the 74-website evaluation set, the nine models in Table 1 obtain test-pooled effective scores from 5.7% to 22.5%, with Claude-Opus-5.5 scoring highest. All five frontier models outscore all four evaluated open-source models in Overall and each test group. Appendix F details the rankings.

Across all nine configurations, Browser test has the lowest pass rate, ranging from 1.8% to 18.0%, compared with 20.4%–63.5% for Request and 20.2%–37.4% for System test. Browser test contains the most tests and therefore accounts for the largest share of possible points in the test-pooled Overall score. Its low pass rates constrain the aggregate results.

4.2 RESOURCE USE

Figure 6 compares Overall scores with mean cost, token usage, and turns. Higher resource use does not consistently correspond to higher scores: GPT-5.6-Luna scores 13.2% at a mean cost of $3.8 per attempt, compared with 11.6% at $76.9 for Grok-4.6. Appendix F provides the detailed resource comparisons and score-to-cost ratios.

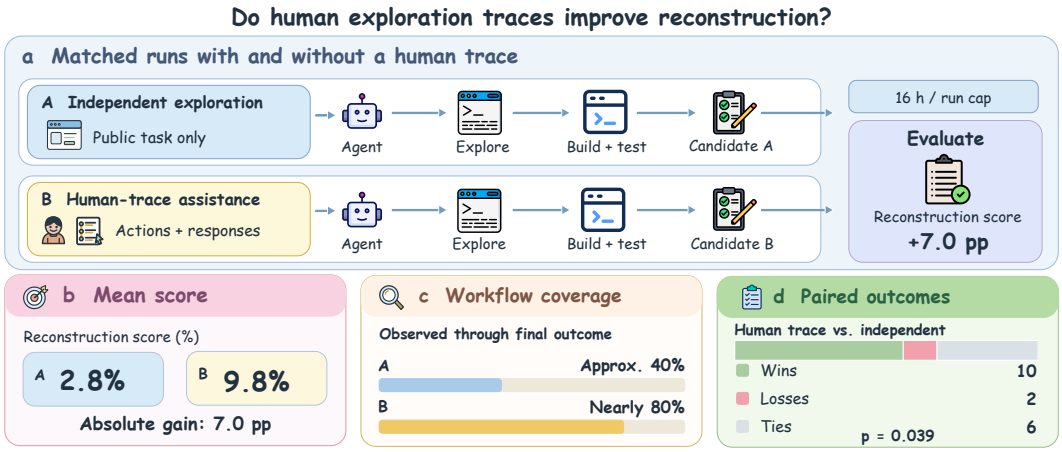


Figure 4: Human-trajectory assisted exploration analysis. **a**, Compare matched 16-hour runs where the agent explores independently and where it is supplied with a human exploration trace. **b**, Compare the overall reconstruction scores for the paired runs. **c**, Observed agent exploration coverage on the reference websites. **d**, Human-trajectory-assisted runs show a statistically significant improvement on overall test score.

5 Analysis

5.1 Human Trajectories and Active Exploration

Website reconstruction requires agents to discover reference behaviors before implementing them. During the experiments, we observed that many agent runs did not explore the reference website comprehensively, especially when encountering long sequences of state-changing actions. To compare the effectiveness of human and agent exploration, we evaluate whether supplying a human exploration trace improves reconstruction scores and exploration coverage.

Figure 4a outlines the two conditions. We compared reconstruction runs with and without human exploration traces on 18 of the 20 websites analyzed in Section 5.2 and Appendix I.3, using matched model and agent configurations. In independent runs, the agent accesses the reference website through the browser gateway and is required to explore and discover its behaviors independently. In trace-assisted runs, the agent additionally receives a human exploration trace containing a sequence of browser actions recorded by a website development expert attempting to explore the reference website comprehensively. These traces only contain browser action records without any web page content. Thus, they can only be used as a supplementary resource, and the agent must still use its own tools to explore the reference website. The trace is mounted read-only into the reconstruction environment, and the agent is explicitly instructed to read it and use the information to guide its exploration.

As Figure 4b shows, the human trace raised the mean reconstruction score across the 18 pairs of reconstruction runs from 2.8% to 9.8%, a paired mean gain of 7.0 percentage points. Each website contributes equally to these means. Figure 4d breaks the pairs down. Trace-assisted runs scored higher in 10 pairs, lower in two, and equal in six; an exact two-sided sign test excluding ties gives $p = 0.039$. Most of the gain came from websites where the independent run passed no hidden tests. Six of the 18 independent runs scored above zero, compared with 12 of the 18 trace-assisted runs, and the five largest gains, on Meetup, Samsung, L.L.Bean, Target and Backcountry, all started from a baseline of zero. The two regressions were AMC Theatres, where the baseline run scored 19.8% and the trace-assisted run scored zero, and Medium, which dropped from 7.5% to 4.5%. Figure 4c shows where the trace changes exploration. Before the first implementation edit, independent runs had observed the final outcome of roughly 40% of workflows, and trace-assisted runs nearly 80%.

Comparing human exploration traces with agents' independent exploration reveals substantial differences. First, humans explore websites much faster. The recorded human sessions have a median duration of just over five minutes, with more than 450 actions and 28 visited pages. In comparison,

agent exploration takes approximately 1.5–2.5 hours and 200–300 steps, reaching only around ten distinct paths. Human activity largely consists of frequent lightweight interactions such as hovering, scrolling, and clicking, whereas individual agent steps are much slower and can take tens of seconds. These differences result in the observation that website development experts are able to explore around ten times as many paths as agents in only one-twentieth of the time. Additionally, these experts are able to explore complex workflows much more comprehensively, while agents often fail to understand the intent of multi-step interactions and abandon them before reaching their final steps.

5.2 Exploration and Implementation Gaps

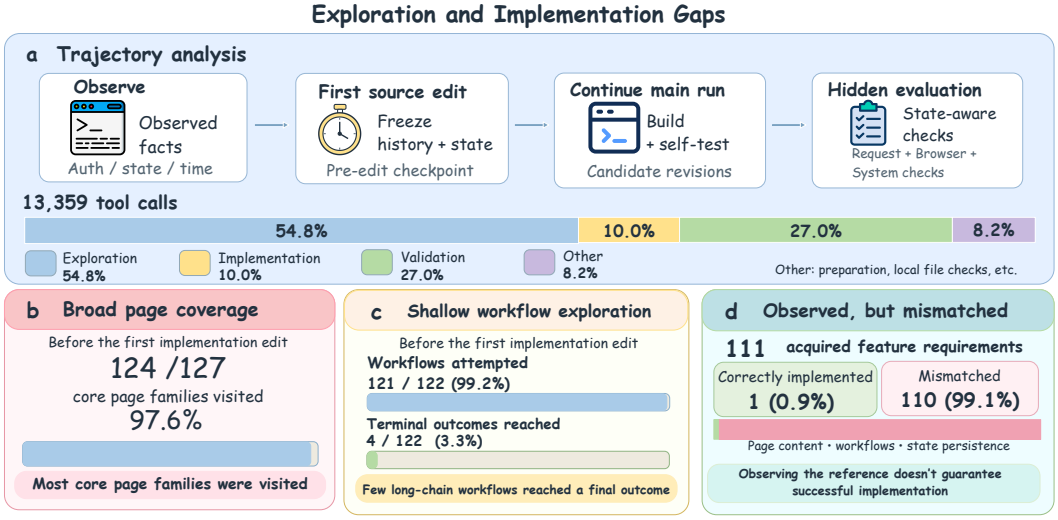


Figure 5: Exploration and implementation gaps across 20 reconstruction runs. **a**, tool-call split. **b–c**, Among the inspected trajectories, agents had visited 124 of 127 core pages and attempted 121 of 122 core workflows before starting implementation, but reached only four terminal outcomes. **d**, Of 111 observed feature requirements, only one was correctly implemented, indicating the gap between exploration and implementation.

For a reconstruction task, the agent's implementation goal is obtained entirely through its exploration of the reference website. Incomplete or incorrect exploration therefore leads to missing features, incorrect behaviors, or implementation errors. To examine these sources of failure, we analyzed reconstruction trajectories from 20 websites, tracing failures back to reference observations and candidate implementation decisions. We examined whether the relevant behavior had been observed before implementation and whether the candidate reproduced the information obtained. Figure 5 summarizes how agents split their effort across exploration, implementation and validation, and how much of what they observed reached the candidate.

Across the 20 trajectories, agents issued 13,359 tool calls: 54.8% concerned exploration, 10.0% implementation, 27.0% validation, and 8.2% other operations such as environment preparation and reading local files. Though a large proportion of tool calls were dedicated to exploration, substantial browsing activity did not imply complete exploration. Among the 17 trajectories that reached a first implementation edit, agents had by then visited 124 of 127 core page families (97.6%) and attempted 121 of 122 core workflows (99.2%), yet only 4 workflows (3.3%) had reached a terminal outcome. Agents see almost every page and finish almost no workflow before they start building.

Information that agents had already obtained was also not consistently reflected in their implementations. Across the 20 inspected websites, 111 feature requirements observed during exploration, covering page contents, workflows and state persistence, had enough evidence to judge whether the observation was carried into the implementation. Only 1 of these (0.9%) was implemented in a way that matched the observation; the remaining 110 (99.1%) were mismatched. Such gaps between

exploration and implementation indicate that agents still have challenges in translating observed information into correct implementation in long-running reconstruction tasks.

The analysis identifies limitations in both exploration and implementation. Agents often visit the relevant pages without getting deep into the interactions, and even correctly acquired information may not be reproduced in the candidate. Improving reconstruction therefore requires both deeper exploration of workflow outcomes and more effective translation mechanisms that connect observed behavior to correctly implemented behavior.

5.3 Time Budgets, Early Delivery and Continuation

Across the experimental trajectories, we observed that despite a 16-hour time limit, most reconstruction runs terminated prematurely, exhibiting a median per-task duration of 3.57 hours. To investigate these early terminations, we examined whether compelling agents to utilize remaining time alters their behavioral patterns and reconstruction performance. Specifically, we conducted supplementary experiments enabling Claude Code's `/goal` hook with a 16-hour budget. Figure **??** illustrates the comparison, showing how a run unfolds with and without `/goal`, what happens after the agent first tries to stop, and which measures of the extra work are available.

In the Lowe's reconstruction task with Kimi-K3, enabling `/goal` extended the run duration from 1.0 to 10.9 hours, increased total agent steps from 130 to 879, and expanded token consumption from 10.6M to 87.9M. Crucially, this pushed the hidden-test score from 0% to 31.3% (+31.3 percentage points). Trajectory analysis reveals that at the 10.25-hour mark, the agent initially declared repairs complete and attempted to exit. This termination was rejected by the hook review, prompting the agent to utilize its remaining 5.75 hours. Over the subsequent time, the agent went 19 steps and executed additional targeted persistence checks across cart, order, account, and session states following a service restart.

At 10.9 hours, the agent submitted a second termination request. The hook accepted this completion, citing verified criteria: exact text matching across 17 routes, end-to-end checkout validation via both HTTP requests and browser interactions, verified state persistence post-restart, and a passing preflight check.

This case highlights early stopping as a critical failure mode in autonomous reconstruction. Even when an agent deems a task resolved, enforcing continued execution compels agent's self-auditing and rigorous validation, uncovering latent bugs and possibly improving overall solution fidelity.

6 Conclusion

We introduce WebsiteBench, a benchmark of 74 expert-built and reviewed website reconstruction tasks evaluated against 50,356 hidden tests. By requiring agents to explore locally hosted reference websites and synthesize runnable, offline systems that replicate both visual and interactive behaviors, WebsiteBench evaluates a capability absent from existing browser use and code generation benchmarks. Our framework pairs autonomous browser-based discovery with a comprehensive hidden test suite spanning Request, Browser, and System tests. Experiments across 9 frontier and open-source models show that current agents struggle to reconstruct unfamiliar sites through autonomous interaction, with Claude Opus 5.5 achieving the top score of just 22.5%. Systematic trajectory analysis reveals three primary failure modes: underexploration, failure to implement observed behavior, and early stopping. Moreover, providing agents with human exploration traces yields substantially higher reconstruction scores, pinpointing autonomous discovery as a fundamental bottleneck. We release the benchmark, reference websites, hidden tests, evaluation infrastructure, and agent trajectories to support future research on agents' capabilities of discovering and rebuilding interactive software.

AI Use Statement

We used generative AI tools to polish the manuscript's language by correcting grammar and improving sentence phrasing.

Reproducibility Statement

Sections 2 and 3 describe the reconstruction task and evaluation protocol. Appendices A–E detail the reference environments, test construction, agent access, functional and visual scoring, and the treatment of resource limits, failed trials, and missing results. Appendix H specifies the records needed to link each result to its reference environment, submitted implementation, test suite, and evaluator version, and to recompute trial scores and site-level summaries.

References

John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. Active vision. *International journal of computer vision*, 1(4):333–356, 1988.

Anthropic. Introducing Claude Opus 5.5, September 2026a. URL https://www.anthropic.com/claude-opus-5-5. Accessed: 2026-09-26.

Anthropic. Introducing Claude Opus 5. https://www.anthropic.com/news/claude-opus-5, 2026b.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Grace Man Chen, Litao Guo, Yifan Wu, Yiyu Chen, Yenchi Tseng, Sicheng Liu, Yuyu Luo, and Ying-Cong Chen. Ui2app: Benchmarking visual interaction inference in executable web application generation. *arXiv preprint arXiv:2607.06306*, 2026a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Yang Chen, Minghao Liu, Yufan Shen, Yunwen Li, Tianyuan Huang, Xinyu Fang, Tianyu Zheng, Wenxuan Huang, Cheng Yang, Licheng Wen, et al. Iwr-bench: Can lvlms reconstruct interactive webpage from a user interaction video? In *International Conference on Learning Representations*, volume 2026, pp. 52158–52189, 2026b.

Deepseek. Deepseek-v4.1-flash. https://www.deepseek.com/news/deepseek-v4-1-flash/, 2026.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, pp. 28091–28114. Curran Associates, Inc., 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html.

Design Arena. Video to website leaderboard. https://www.designarena.ai/leaderboard/video-to-website, 2026. Accessed August 2026.

John H. Flavell. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American Psychologist*, 34(10):906–911, 1979. doi: 10.1037/0003-066X.34.10.906.

Divyansh Garg, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, Christian Schroeder de Witt, and Sumeet Ramesh Motwani. REAL: Benchmarking autonomous agents on deterministic simulations of real websites. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=Un1sWxmZuI.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pp. 1834–1845. Association for Computing Machinery, 2025. ISBN 9798400712746. doi: 10.1145/3696410.3714889. URL https://doi.org/10.1145/3696410.3714889.

JunJia Guo, Yuhang Yao, Jiawei Zhou, and Jingdi Chen. Vista: An end-to-end benchmark for visual spec-to-web-app coding agents, 2026. URL https://arxiv.org/abs/2605.26144.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification, 2026. URL https://arxiv.org/abs/2603.26648.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Philip Nicholas Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6 in Cognitive Science Series. Harvard University Press, 1983.

Kimi. Kimi-k3. https://www.kimi.com/news/kimi-k3, 2026.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Ryan Li, Yanzhe Zhang, and Diyi Yang. Sketch2code: Evaluating vision-language models for interactive web design prototyping, 2024. URL https://arxiv.org/abs/2410.16232.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

David Marr. *Vision: A computational investigation into the human representation and processing of visual information*. The MIT Press, 2010.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: A benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

George A. Miller, Eugene Galanter, and Karl H. Pribram. *Plans and the structure of Behavior*. Henry Holt and Company, New York, 1960. URL https://cse.buffalo.edu/~rapaport/Papers/Papers.by.Others/milleretalch1.pdf.

Allen Newell and Herbert A. Simon. *Human problem solving*. Prentice-hall Englewood Cliffs, NJ, 1972.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6, 2026.

Judea Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, 2000. URL https://assets.cambridge.org/97805217/73621/frontmatter/9780521773621_frontmatter.pdf.

Qwen Team. Qwen3.8-max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Donald A. Schön. *The reflective practitioner: How professionals think in action*. Basic Books, 1984. ISBN 9780465068784. URL https://www.hachettebookgroup.com/titles/donald-a-schon/the-reflective-practitioner/9780465068784/.

Kean Shi, Zihang Li, Tianyi Ma, Zengji Tu, Jialong Wu, Xinbo Xu, Qingyao Yang, Ruoyu Wu, Weichu Xie, Ming Wu, et al. Saas-bench: Can computer-use agents leverage real-world saas to solve professional workflows? *arXiv preprint arXiv:2605.15777*, 2026.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3956–3974, 2025.

Anneliese Von Mayrhauser and A Marie Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.

Xai. Grok-4.6. https://x.ai/news/grok-4-6, 2026.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R. Lyu. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 241–253. IEEE, November 2025. doi: 10.1109/ase63991.2025.00028. URL http://dx.doi.org/10.1109/ASE63991.2025.00028.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Z.ai. GLM-5.3-Flash. https://huggingface.co/zai-org/GLM-5.3-Flash, 2026. Accessed September 26, 2026.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Zhuofeng Li, Xingwei Qu, Zhengkang Guo, Yuanzhe Shen, Dingjie Song, Han Zhou, Tuney Zheng, Xian Wu, Hao Yu, Songcheng Cai, Yi Lu, Yunzhuo Hao, Minyi Lei, Liang Chen, Kai Zou, Huifeng Yin, Wendong Xu, Dongfu Jiang, Ping Nie, Jiaheng Liu, Wenhu Chen, and Kelsey R. Allen. Clawbench: Can ai agents complete everyday online tasks?, 2026. URL https://arxiv.org/abs/2604.08523.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Website Construction and Test Coverage

A.1 Reference Environments and Scope

The reference environment defines the reconstruction target. Each environment contains a local implementation of selected website functionality together with the resources and initial state needed to explore it. Evaluation compares the candidate with the selected functionality implemented in this local environment. Each of the 74 website tasks includes a local reference implementation.

The corpus covers selected website domains and user workflows. Each local reference environment implements the pages, interactions, and state transitions needed for evaluation. External dependencies may be replaced with local services or omitted when they fall outside the task. For example, email and payment workflows use local services that reproduce the required user-visible behavior.

We count each distinct behavior under a stated initial condition as one unit of coverage. Route counts describe navigation structure, while test counts describe the executable suite; several tests may cover the same behavior. Table 3 reports the corpus composition by website category; behavior-level coverage additionally requires the workflow inventory.

A.2 From Site Inventories to Tests

Test construction combines manual specification with programmatic expansion. For each website, website development experts identify the pages, forms, controls, and workflows to be evaluated, and define concrete inputs together with the observable outcomes expected from each interaction. Construction tools use these specifications to produce executable test variants and visual checkpoints. Before inclusion, these experts compare the resulting tests against the reference environment to verify that the encoded interactions and assertions reflect the intended website behavior.

Experts review each assertion used in scoring by asking three questions. First, where can the agent discover the relevant behavior or public requirement? Second, do the test actions actually exercise it? Third, does the assertion distinguish success from a plausible incorrect implementation? For a workflow, a generic success banner may be insufficient if the operation is meant to save a record with particular contents. The observation should check the result that matters to the user, such as the saved values on a detail page or the items shown in an order summary.

Website development experts may inspect reference code. Agents, however, must be able to discover each tested requirement through the permitted interface or public specification. A hidden storage key, private response header, or internal JSON field can be required only if it is specified in information available to the agent. A downloadable JSON file can be tested when users can find and use the reference's export feature. The review therefore considers how users access the information, rather than its file format.

Direct HTTP requests can test page responses and submissions from rendered forms. We report direct HTTP checks separately from workflows tested through visible controls.

A.3 Stateful Behavior

A persistence test separates saving data from reading it later. Its initial interaction creates or changes a record; a subsequent page visit or application restart is followed by a read of that record. Repeating the write after restart would not establish persistence. The test also specifies whether the data directory is retained or cleared during the restart.

Multi-user tests specify what another user can access. A second user is expected to see shared content or to be denied access to private content, according to the reference's sharing and access rules. The test identifies authentication setup separately from the user operation being evaluated. If login or registration is the target behavior, those steps are part of the test.

Recovery tests specify the rejected operation, the visible feedback, and a subsequent valid operation. This checks that rejection leaves the application in a state from which a user can continue. Setup actions, such as dismissing a cookie banner, are distinguished from the operation being tested; they are not automatically repeated after restart when their original precondition no longer holds.

Table 2: Illustrative test design for a record-saving workflow. The examples show the review criteria used during test construction.

Behavior	Interaction and observation	Error detected
Submission	Fill the rendered form, submit, and inspect the saved record.	A success message appears but the submitted fields are discarded.
Reading from another page	Open the record through a list or detail page.	Only the submitting page reflects the change.
Persistence	Restart with the same data, reopen the record, and read its contents.	State survives navigation but is lost on process restart.
User access	Sign in as another user and try to view the shared or private content.	Ownership or sharing differs from the reference.
Recovery	Submit invalid input, inspect the rejection, correct the input, and submit again.	Rejection corrupts state or prevents a subsequent valid operation.

A.4 Test Classification and Redundancy

We classify tests by how they run, what they inspect, and which behaviors they check. T1/T2/T3 identify how tests run. Additional annotations identify the content, control, page response, or application state inspected. Within T2, L1 checks visible content or the immediate result of an interaction on one page or component. L2 checks a state change through a subsequent observation on the same page or another page or component. The distinction is between immediate feedback and reading back the changed state, rather than the number of pages visited. L3 adds a check involving the application lifecycle, a different user, or recovery from a rejected operation. We assign these labels from the test's actions and assertions.

Test variants can share a workflow while checking different requirements, such as rejected input and successful submission. Changing an identifier or repeating the same observation does not create a new requirement. The construction audit therefore distinguishes exact duplicates from variants that share a template but differ in their inputs, preconditions, or checked outcomes. Counts of distinct test templates and the distribution of tests across routes help describe coverage alongside the total test count.

A.5 Corpus Categories

Table 3: WebsiteBench task categories.

Family	Sites	Family	Sites	Family	Sites
E-commerce & retail	22	Food & dining	11	Content & community	9
Professional services	9	Productivity & SaaS	5	Travel & outdoors	5
Events & entertainment	5	Education	5	Jobs & careers	3
				Total sites	74

B Agent Inputs and Candidate Execution

B.1 Observation Interface and Asset Access

The agent accesses the reference through the provided browser gateway. It can navigate, inspect screenshots and the interface state reported by the gateway, and issue the permitted browser actions. It cannot attach a separate debugging client or use the reference endpoint as a general-purpose HTTP interface.

The browser gateway allows agents to retrieve media files from the reference site. JavaScript and CSS are filtered out by the gateway and are not exposed to the agent. Agents can therefore reuse media while implementing page layout and interactive behavior from browser observations and public task specifications. The main experiments use this interface. HTML-document and JSON/API-response exports, arbitrary page evaluation, and server-side source access remain unavailable.

These restrictions apply to the reference. The agent can inspect, modify, and debug its own candidate with local file and development tools. Public instructions and integration specifications supplement browser evidence; for example, they specify the endpoint through which a candidate sends mail. Hidden tests, private reference observations, and scoring outputs remain unavailable during generation.

Table 4: Agent information access in the main experiments.

Information	Access	Role in reconstruction
Rendered reference	Browser screenshots and interface state reported by the gateway	Discover layout, content, controls, and state transitions.
Reference media	Gateway retrieval of media files; JavaScript and CSS are filtered out	Reuse media while implementing layout and behavior from observations.
Reference internals	No general HTML/API export, arbitrary page evaluation, or server-side source access	Restrict access to implementation details.
Public specifications	Task instructions, runtime requirements, and specifications for local services	Specify requirements that browsing alone may not reveal.
Candidate implementation	Local file, shell, build, and debugging tools	Implement and check the reconstruction.
Hidden evaluation	Unavailable during generation	Assess the submitted website after reconstruction ends.

B.2 Build and Runtime Requirements

The submission is a source tree with an executable `compile.sh` build script. The verifier copies the tree into a private build directory, removes prebuilt runtime output, and invokes compilation without public network access. Compilation produces a root entrypoint named `executable`, which may use other files in the packaged build. The complete build directory is kept unchanged during evaluation; the entrypoint need not be a self-contained binary.

The process binds to the supplied host and port and writes persistent state under the assigned data directory. It remains in the foreground and handles termination signals. The single-process runtime permits threads but forbids child-process creation after startup. These constraints limit deployment choices even though the reference and candidate need not share an implementation language or source organization.

Table 5: Public candidate runtime requirements. Per-run resource limits are recorded separately from this interface.

Component	Requirement
Build	No-argument `compile.sh`; offline build in a relocated directory.
Launch	Root `executable`; packaged dependencies; foreground process.
Address	Bind to `HOST:PORT`.
Readiness	HTTP 200 at `/healthz` with JSON {`"status":"ok"`}.
State	Runtime writes beneath `DATA_DIR`; use supplied `SEED` and `TZ`.
Lifecycle	Handle `SIGTERM`; threads allowed, child processes forbidden.
Network	No public network; only the explicitly permitted local services.

B.3 Mail and Other Local Integrations

During reference exploration, the agent uses an independent email address assigned to that trial and reads delivered messages only through the browser inbox. Candidate runtimes may send mail through a local write-only endpoint with a public request format, while only the verifier can read the candidate's mailbox.

Tests use the email codes generated during each run to complete the interaction, then verify the resulting state, such as successful registration or account access. Payment flows are simulated locally and require no live charge or external payment credentials.

B.4 Information Flow

Table 6 lists the inputs and outputs across benchmark preparation, agent execution, evaluation, and release. The authoring row identifies materials prepared for the agent; the release row describes access after publication. Neither is an additional stage of agent generation. Each evaluation record includes file identifiers and content hashes that link the result to its candidate, tests, and evaluator.

Table 6: Information access and saved files at each stage of WebsiteBench. Private files are stored outside the agent's environment and directories writable by the candidate.

Stage	Information available to the agent	Information kept private	Saved output
Authoring	instructions, starter code, public runtime requirements	server-side reference source, hidden tests, reference solution	validated website and task-instance records
Exploration	restricted reference browser; screenshots, reported interface state, actions, permitted media files	recorded final states and behaviors covered by hidden tests	timestamped agent trajectory
Construction	writable candidate files, browser and development tools	reference and evaluator directories	candidate source and submission record
Compilation	compilation status only	isolated offline build directory and build logs	read-only `executable` directory with content hashes
Evaluation	no access after submission	hidden tests, reference observations, fixed executors, copying checks	test results, logs, failure evidence, evaluation summary and score
Release preparation	released files after removal of private information	record verification and files awaiting publication	record listing each published file and its hash

C Related Work

Web agents and computer-use agents. WebShop and Mind2Web evaluate shopping tasks and webpage action prediction, respectively (Yao et al., 2022; Deng et al., 2023). WebArena, VisualWebArena, and WebVoyager assess task completion in self-hosted or live web environments (Zhou et al., 2024; Koh et al., 2024; He et al., 2024). AgentBench, OSWorld, and GAIA extend evaluation to diverse environments, desktop use, and general-assistant tasks (Liu et al., 2024; Xie et al., 2024; Mialon et al., 2023). SaaS-Bench covers professional SaaS workflows, while ClawBench evaluates everyday tasks on live websites (Shi et al., 2026; Zhang et al., 2026). These benchmarks assess actions or task outcomes, rather than the reconstruction of the underlying system.

Code generation and software engineering agents. HumanEval, MBPP, and LiveCodeBench evaluate solutions to specified programming problems (Chen et al., 2021; Austin et al., 2021; Jain et al., 2025). SWE-bench evaluates issue resolution in existing repositories (Jimenez et al., 2024). WebsiteBench also evaluates executable software, but agents must infer the target behavior through browser interaction without access to the reference source code.

Website generation and reconstruction. Design2Code studies screenshot-to-code generation, while Interaction2Code adds visual evidence of interactions (Si et al., 2025; Xiao et al., 2025). VISTA varies textual, visual, and structural inputs alongside development-stack constraints (Guo et al., 2026). Vision2Web includes multi-page frontend and full-stack development (He et al., 2026). UI2App and IWR-Bench evaluate behavioral reconstruction from screenshot sets and interaction videos, respectively (Chen et al., 2026a;b). In WebsiteBench, agents additionally choose which interactions to perform on a running reference and may revisit it during implementation. Table 7 compares these settings by source evidence, agent objective, output artifact, and evaluation signal.

Table 7: Comparison with representative web-agent and website-generation benchmarks. Among the compared settings, WebsiteBench requires agents to actively explore an existing website, recover its implicit browser-observable specification, and reconstruct it as an offline runnable website. Scale uses each benchmark's reported units.

Benchmark	Task Setup			Agent Task		Evaluation	
	Source	Scale	Specification	Objective	Result	Visual	Functional
Mind2Web (Deng et al., 2023)	Static traces	2,350 tasks 137 sites	Task instruction	Operate	Actions	✗	Action match
WebArena (Zhou et al., 2024)	Self-hosted sites	812 tasks 5 sites	Task instruction	Operate	Actions	✗	State check
VisualWebArena (Koh et al., 2024)	Self-hosted sites	910 tasks 3 sites	Task instruction	Operate	Actions	✗	State check
SaaS-Bench (Shi et al., 2026)	Deployable SaaS	106 tasks 23 systems	Task instruction	Operate	Actions	✗	Checkpoints
Design2Code (Si et al., 2025)	Screenshot	484 pages	Provided artifact	Generate	Webpage	✓	✗
WebCode2M (Gui et al., 2025)	Screenshot + design	2.56M pairs	Provided artifact	Generate	Webpage	✓	✗
Vision2Web (He et al., 2026)	Prototype + requirements	193 tasks 16 categories	Provided artifact	Generate	Website	✓	✓
Design Arena (Design Arena, 2026)	Video	Dynamic	Provided artifact	Generate	Website	Human preference	Partial
WebsiteBench *(Ours)*	**Self-hosted sites**	**74 sites 50,356 tests**	**Implicit in source**	**Explore + Reconstruct**	**Runnable website**	✓	✓

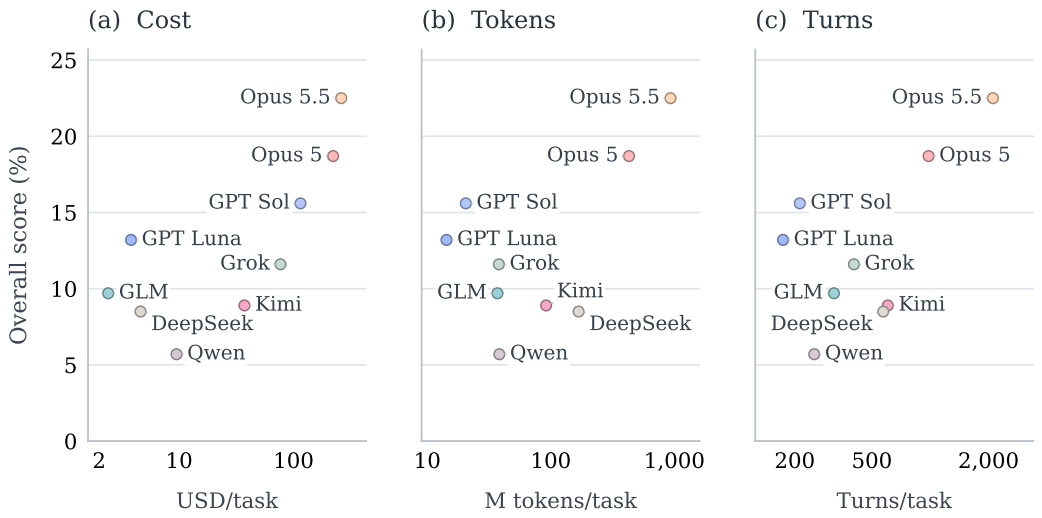


Figure 6: Resource use and reconstruction performance on WEBSITEBENCH. Each point represents a model from Table 1. Cost, tokens, and turns are averages per task. Horizontal axes use logarithmic scales.

D EVALUATION AND TEST VALIDATION

D.1 REFERENCE OBSERVATIONS AND FUNCTIONAL TEST RESULTS

Each evaluation records how its reference observations were obtained. The verifier either loads previously captured observations and screenshots or runs a fixed version of the private reference and captures the required observations during the trial. These observations are stored outside the agent's environment. Requirements with explicit expected values are checked directly.

For a direct test, let $D_i \in \{0, 1\}$ denote failure or success. A journey test checks a sequence of browser interactions that completes a user task. Let F_i^{PW} and F_i^{BU} be its results from the fixed Playwright and Browser Use executors. Functional success is defined as

$$F_i = \begin{cases} D_i, & \text{direct interface or fixed-requirement test,} \\ F_i^{\mathrm{PW}} F_i^{\mathrm{BU}}, & \text{journey test.} \end{cases} \tag{3}$$

Each executor uses an independent candidate runtime and browser state. Browser Use executes test instructions deterministically. Functional test results come from deterministic checks. LLM judgments and human ratings are excluded from the test score.

A journey test passes its functional checks only when both executors succeed. Executor disagreements are retained for diagnosis. Tests that use direct requests and tests that complete a workflow through visible controls are distinguished when interpreting functional coverage.

D.2 VISUAL REPLAY AND REGION SELECTION

A visual checkpoint specifies its route, action sequence, viewport, regions, and masks. The verifier creates a fresh candidate runtime for the checkpoint and replays those actions before taking a screenshot. This run is separate from functional testing. The test specification identifies the behavior and application state that the screenshot should show.

Reference and candidate screenshots are converted to RGB and checked against the declared viewport. Regions must lie within the image, must not overlap, and must contain unmasked pixels. A missing or incorrectly sized candidate screenshot fails the checkpoint. Invalid reference material or an incorrectly specified checkpoint is recorded as an evaluation error.

Masked candidate pixels are replaced by the corresponding reference pixels before SSIM maps are calculated; masked positions are also excluded from aggregation. This avoids attributing differences in masked content to nearby unmasked pixels through the local SSIM window. Region and mask definitions are fixed in the test specification before candidate outputs are examined.

D.3 Baseline-Normalized Similarity

Large uniform areas can contribute substantial raw similarity while conveying little about reconstructed foreground content. The comparator uses a constant-color image as a reference baseline for each region. Its color is the rounded channel-wise median of the unmasked reference pixels. Candidate and baseline similarities are computed against the same reference region.

Let a_{jx} and b_{jx} be the candidate and baseline SSIM-map values at unmasked pixel x in region j, after averaging color channels and clipping to $[0, 1]$. Define

$$g_{jx} = \min\{\max(a_{jx} - b_{jx}, 0), 1 - b_{jx}\}, \qquad h_{jx} = 1 - b_{jx}. \tag{4}$$

The checkpoint similarity before rounding is

$$\widetilde{v} = \begin{cases} \dfrac{\sum_j \sum_{x \in A_j} g_{jx}}{\sum_j \sum_{x \in A_j} h_{jx}}, & \displaystyle\sum_j \sum_{x \in A_j} h_{jx} > 10^{-12}, \\ \dfrac{\sum_j \sum_{x \in A_j} a_{jx}}{\sum_j |A_j|}, & \text{otherwise}, \end{cases} \tag{5}$$

where A_j contains the unmasked pixels in region j. Thus aggregation is weighted by the possible improvement over the baseline; rather than generally an area-weighted mean of region-level normalized scores. The second branch uses raw area-weighted SSIM when the reference has negligible improvement available over its baseline.

SSIM is computed with an odd window size of at most seven pixels, reduced when necessary to fit the evaluated region. Regions smaller than three pixels in either dimension use the comparator's small-region image-moment calculation. Pixel intensities are represented on the 0–255 scale, and the checkpoint score v is rounded to ten decimal places before thresholding. We retain the unrounded area-weighted SSIM and per-region scores for diagnostic analysis.

D.4 Binary Credit and Diagnostic Values

$$p_i = F_i \prod_{k \in \mathcal{V}_i} \mathbf{1}[v_{ik} \geq \tau], \qquad S_{c,s,r} = \frac{100}{N_s} \sum_{i=1}^{N_s} p_i, \qquad P_{c,s,r} = q_{c,s,r} S_{c,s,r}. \tag{6}$$

Here N_s is the number of tests for site s, c identifies the agent configuration, and r identifies the trial. The set $\mathcal{V}_i$ contains the visual checkpoints of test i; its empty product is one. The indicator $q \in \{0, 1\}$ is zero for submissions disqualified for unauthorized copying and one otherwise. The current shared evaluator uses $\tau = 0.75$. Each reported trial records the threshold used by its evaluator version, including any scoring code specific to that task.

Equation 6 gives equal weight to each test across all tiers and levels. Every visual checkpoint of a test has to meet τ; averaging checkpoints cannot compensate for a failed checkpoint. A checkpoint value below the threshold provides no fraction of a test point, although it remains available for error analysis. Functional and visual conditions are combined at the test level: a functional failure gives that test zero credit regardless of its screenshot similarity.

Table 8: Score components and diagnostic measures. Every test has equal weight.

Quantity	Definition	Interpretation
Functional result F_i	Equation 3	Direct-test success or success by both journey executors.
Checkpoint similarity v_{ik}	Equation 5, rounded to ten decimals	Continuous similarity; must meet the threshold for credit.
Test point p_i	$F_i \prod_k \mathbf{1}[v_{ik} \geq \tau]$	One or zero, including T1 and T3.
Trial score S	$100 \sum_i p_i / N_s$	Percentage of declared tests passed.
Trial score after disqualification P	qS	Zero for a submission disqualified for copying.
Configuration score M_c	Equation 7	Percentage of tests passed across eligible trials.
Tier/level pass rates	Mean test points within each category	Results by test category; categories receive no additional weight.

Each test contributes equally to the primary trial score. Ties in the primary score are broken by T1 and then T3 pass rates; tests from both tiers also contribute normally to the primary score. A suspected-copy flag does not by itself set q to zero; disqualification decisions and their outcomes are recorded separately.

A test-derived trial score is computed only from a complete, valid result set. Recorded candidate failures remain in the fixed denominator and receive zero credit. The paper separately assigns zero to a documented absent submission under the accounting rule in Appendix E. Other missing or malformed records may invalidate the evaluation; they are not removed to reduce its test denominator.

Direct-test result, or success by both journey executors, $\longrightarrow F_i$. Playwright replay of specified visual checkpoints $\longrightarrow v_{ik}$.

$$p_i = F_i \prod_{k \in \mathcal{V}_i} \mathbf{1}[v_{ik} \geq \tau], \qquad S_{\%} = 100 \frac{\sum_i p_i}{N}, \qquad \text{reward} = q \frac{\sum_i p_i}{N}.$$

The visual product is one when no checkpoint is specified. A separate copying check determines q. Visual similarity and functional pass rates are also reported by test level. Aggregation across trials follows Equation 7.

Figure 7: Current execution and scoring. Visual values are normalized against a constant-color baseline and must meet the recorded threshold for test credit ($\tau = 0.75$ in the current shared evaluator). Each test contributes one binary point.

D.5 Validation of the Test Suite

D.5.1 Reviewing Assertions and Running the Reference

Assertion review checks whether the tests describe an intended behavior that agents can discover from the reference or public specification. Reference execution checks whether the implemented test completes and produces the expected result under the specified conditions. Validation records report these two checks separately.

An oracle implementation is the known reference solution used to check test execution. The current validation procedure uses one reference solution per task. A test suite qualifies for inclusion after its assertions are reviewed and a complete run of the final reference and test versions produces a raw reward of exactly one.

Across the 74 reviewed website tasks, both complete oracle runs produced one passing result for every test ID, yielding a raw reward of 1.0 in each run.

D.5.2 Sensitivity to Incorrect Reconstructions

A passing reference run verifies that the suite accepts the reference implementation, while negative controls assess whether it rejects targeted reconstruction errors. Injected defects include nonfunctional controls, disabled submission, dropped form inputs, state loss after restart, and incorrect user ownership.

Each modified implementation is evaluated against its unmodified counterpart under identical tests and execution conditions. We distinguish tests that detect the injected defect from unrelated failures, including startup errors. Defect injection, repeated reference runs, and cross-host evaluation are reported separately for the corresponding test versions.

D.5.3 Rendering and Executor Stability

We assess the visual threshold by comparing repeated reference renders with controlled visual perturbations, including missing foreground content, layout shifts, and typography changes. The resulting similarity distributions indicate whether incidental rendering variation overlaps with meaningful reconstruction errors. Threshold comparisons use identical saved outputs, or the same candidate and test set.

Executor agreement is summarized by four paired Playwright and Browser Use outcomes: both pass, Playwright only, Browser Use only, and neither passes. Disagreements are analyzed using the executed actions, reference observations, and resulting candidate outputs.

Table 9: Checks used to validate the test suite and the evidence required for each.

Check	Question	Evidence
Assertion review	Does the assertion follow from observable behavior or a public requirement?	Test definition and supporting reference observation.
Reference execution	Can the intended reference satisfy the test?	Executed test IDs, verdicts, and reference revision.
Negative controls	Does the test detect the intended defect?	Modified candidate, targeted defect, and observed verdict.
Rendering stability	Does rendering variation change the visual verdict?	Repeated captures, similarity values, and threshold crossings.
Executor agreement	Do functional executors agree on the same behavior?	Paired verdicts and inspected disagreements.

For Claude-Opus-5, we recompute performance under alternative evaluation rules while holding the eligible trials, test set, evaluator version, and copying decisions fixed. The single-executor variants evaluate journey tests using only the specified executor while preserving the original visual thresholds; direct tests are unaffected. The functional-only variant preserves all functional requirements but removes visual checks.

For the visual diagnostic, we consider only test instances that satisfy the original functional requirements and contain at least one visual checkpoint. We summarize each instance by its minimum similarity across checkpoints and report the mean of this quantity over 5,030 qualifying instances, approximately 10% of all test instances.

At the reported precision, the alternative rules leave the overall score unchanged for this configuration. The mean similarity of 0.99 describes only the functionally passing instances with visual checkpoints. Together, these diagnostics describe the effect of the scoring rules on the selected outputs; negative controls and rendering tests assess different questions about the evaluator.

E EXPERIMENTAL CONFIGURATION AND AGGREGATION

E.1 SHARED TASK INTERFACE AND AGENT SETTINGS

The main experiments use the same reference-access configuration. The browser gateway provides interactive observations and access to reference media files while filtering out JavaScript and CSS so that agents cannot inspect or retrieve these files directly. This access policy is held fixed across evaluated agents. For each agent, we record the model and reasoning setting, command-line client and harness versions, instructions, context-management policy, and resource limits applied to the run.

The generation time limit is 16 hours in all reported runs. Compilation and verification use a separate environment with their own time, CPU, and memory limits. Table 10 lists the shared Claude Code settings. Model-specific settings are recorded in the corresponding run records.

Table 10: Agent run configuration.

Group	Required field	Current value
Agent	harness name	Harbor 0.22.0; Claude Code 2.1.268
Reasoning	reasoning effort/budget	`max`
Sampling	temperature, top-p	Default temperature and top-p
Limits	maximum output tokens	No maximum output-token limit
Context	window, truncation/ compaction/summarization policy	No configured context reset or external memory. Per-model context windows and thresholds for automatic context management are unspecified.
Transport	retry policy, timeout, concurrency	1 trial per job; 1 concurrent agent. Harbor: at most 2 retries, with waits of 1–60 s.
Generation	wall clock	16 h

Table 11: Evaluation settings recorded for each experimental condition. The recorded settings identify the task and evaluator versions used for the trial.

Setting	Recorded specification
Executors	Playwright and Browser Use versions, interaction settings, and separate functional runtimes.
Rendering	Browser version, fonts, viewport, locale, time zone, clock, and animation settings.
Reference	Reference revision, capture mode, initial state, and reset procedure.
Isolation	Writable paths, runtime state, and explicitly permitted local service endpoints.
Scheduling	CPU and memory limits, number of test partitions and workers, and applied timeouts.
Scoring	Test set, scorer revision, normalization, rounding, visual threshold, and copying rule.
Failures	How candidate failures are counted, when infrastructure failures are retried, and whether a trial is valid.

E.2 FAILURE AND MISSING-RESULT ACCOUNTING

Every scheduled trial is assigned an outcome. A valid candidate may obtain either a positive score or zero. Compilation and runtime-interface failures of an attempted submission produce an all-zero result set. An untouched scaffold is classified as `CANDIDATE_ABSENT`. It produces no reward file and receives zero credit under the paper's failure-accounting rule. Under this rule, the absent submission remains an eligible trial with $P_{c,s,r} = 0$ and weight N_s in Equation 7. This assigned score is recorded separately from a complete set of executed test results. Trials invalidated by infrastructure failures are excluded from scoring. Outcome frequencies are reported over all scheduled trials.

The test-level infrastructure retry uses the same seed and does not create an additional reconstruction attempt. Job-level retry policies can instead restart a larger part of the trial and are recorded

separately. For each selected retry, analysis records retain the failed attempt and the rule that made the retry eligible.

The job summary averages outputs containing numeric rewards and counts missing or malformed outputs separately. We calculate the paper's aggregate score separately using Equation 7. If a configuration has no eligible trial for a site, its site mean is undefined. The analysis must state whether the comparison is incomplete or restricted to a common subset chosen in advance, and identify the sites included. Calculating the fraction of attempts that produce usable submissions requires the outcomes of all scheduled trials.

E.3 Uncertainty and Paired Comparisons

For configuration c, let $\mathcal{R}_{c,s}$ be the nonempty set of eligible trials on site s, and let $\mathcal{S}$ be the common 74-site set. The aggregate gives equal weight to each test result across eligible trials:

$$\overline{P}_{c,s} = \frac{1}{|\mathcal{R}_{c,s}|} \sum_{r \in \mathcal{R}_{c,s}} P_{c,s,r}, \qquad M_c = \frac{\sum_{s \in \mathcal{S}} N_s |\mathcal{R}_{c,s}| \overline{P}_{c,s}}{\sum_{s \in \mathcal{S}} N_s |\mathcal{R}_{c,s}|}. \tag{7}$$

Here N_s is the scored test count for site s. Valid zero scores remain in these means. A site with no eligible trial has no defined site mean and requires the missing-data decision described above.

All eligible tests receive equal weight in M_c, so websites with more tests have greater influence on the aggregate score. We estimate uncertainty by resampling websites and recomputing M_c for each sample. Tests from the same website are kept together because they often share functionality and reference resources.

To compare configurations c and d, we use the same set of websites for both. For each resample, we recompute $M_c - M_d$ using the corresponding results from the sampled websites. We report the number of resamples, random seed, confidence level, number of repeated trials, and any adjustment for multiple comparisons.

Preliminary uncertainty check. The available workbook provides `passrate` values and test counts for five configurations on a common subset of 36 websites. Treating `passrate` as the trial score after copying disqualification, we recomputed the aggregate in Equation 7 and resampled websites as clusters. In this subset, the estimated scores and exploratory 95% percentile bootstrap intervals are 12.46% [7.46%, 18.02%] for Claude Opus 5, 10.50% [6.29%, 15.02%] for GLM 5.3 Flash, 9.63% [6.23%, 13.23%] for Kimi K3, 9.12% [4.96%, 14.23%] for DeepSeek V4.1 Flash, and 6.84% [3.78%, 10.44%] for Qwen 3.8 Max. These calculations use 20,000 site-level resamples with random seed 20260926. They describe the 36-site subset and do not provide uncertainty for the nine-configuration, 74-site comparison in Table 1.

E.4 Evaluation Records

Each result records the agent run, reference environment, submitted candidate, test results, evaluator revision, analysis inputs, visual threshold, and browser runtime. Additional inputs, such as human traces, are recorded when used.

Reevaluation of a saved candidate and rescoring from stored test results retain the original run identity and are recorded as derived evaluations rather than new reconstruction attempts.

F Extended Results and Diagnostic Analyses

Model rankings. Table 1 reports the complete ranking across the 74-website evaluation set. Claude-Opus-5.5 scores highest at 22.5%, followed by Claude-Opus-5 at 18.7%. GLM-5.3-Flash is the best-performing evaluated open-source model at 9.7%, 12.8 percentage points below Claude-Opus-5.5. The two Claude models, two GPT models, and Grok-4.6 average 16.3%, versus 8.2% for the four open-source models. Each of these five models outperforms every evaluated open-source model in Overall and each test group.

Claude-Opus-5.5 leads Request (63.5%) and Browser (18.0%), while GPT-5.6-Sol leads System (37.4%), followed by Claude-Opus-5.5 (34.8%). Claude-Opus-5.5, Claude-Opus-5, and GPT-5.6-Sol occupy the top three positions in every group, although their order changes in System. Among the evaluated open-source models, DeepSeek-V4.1-Flash, GLM-5.3-Flash, and Kimi-K3 lead Request, Browser, and System, respectively; no single model leads all three groups.

Resource use and score-to-cost ratios. Claude-Opus-5.5 has the highest mean cost, total token usage, and turn count, at $261.1, 930.5M tokens, and 2,107.1 turns per reconstruction attempt. GPT-5.6-Luna uses the fewest tokens (14.3M) and turns (174.0), while GLM-5.3-Flash has the lowest mean cost ($2.4). Among the evaluated open-source models, GLM-5.3-Flash combines the highest Overall score (9.7%) with the lowest mean cost and token usage (37.0M).

The score-to-cost ratio divides Overall by mean cost per reconstruction attempt. GLM-5.3-Flash has the highest ratio at 4.04 percentage points per dollar, followed by GPT-5.6-Luna at 3.47; Claude-Opus-5.5 has a ratio of 0.09.

Trajectory diagnostics. Figure 8 summarizes tool activity, exploration coverage, implementation of observed requirements, and the paired human-trace comparison. Each panel uses the sample and denominator stated in its caption. Resource comparisons appear in Figure 6.

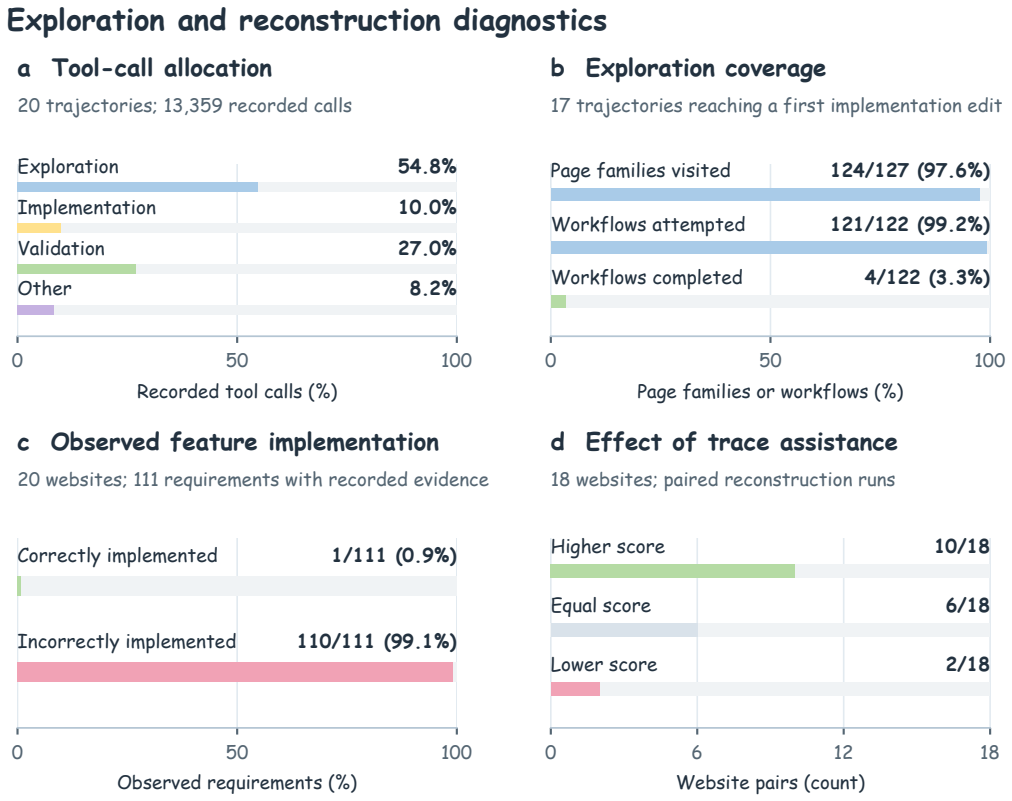


Figure 8: Exploration and reconstruction diagnostics from the analysis subsets described in Section 5. (a) Shares of 13,359 recorded tool calls across 20 trajectories. (b) Exploration coverage before the first implementation edit in each of the 17 eligible trajectories: 124 of 127 page families visited, 121 of 122 workflows attempted, and 4 of 122 workflows completed. (c) Correct implementation of 1 of 111 observed requirements with enough evidence to judge implementation across 20 websites; the remaining 110 requirements were not implemented correctly. (d) Trace-assisted runs scored higher, equally, and lower than independent runs on 10, 6, and 2 of the 18 paired websites, respectively. Percentages use the sample counts stated for each panel.

G Agent Behavior and Qualitative Analysis

G.1 Categories of Agent Behavior

Table 12: Categories for interpreting agent behavior during website reconstruction.

No.	Activity	Agent Behavior
0	**Understanding the Task**	Identify the reconstruction objective, constraints, and success criteria.
1	**Inspecting the Interface**	Identify UI elements, layout, content, and visible state from observations.
2	**Exploring the Reference**	Choose interactions to discover states, behaviors, and workflows not yet observed.
3	**Mapping Website Structure**	Describe the pages, components, navigation paths, and relationships among them.
4	**Understanding Workflows**	Infer how user actions trigger state transitions and enable workflows.
5	**Predicting Action Effects**	Infer how actions affect application state and predict their effects on related components.
6	**Planning**	Divide the reconstruction task into implementation and testing steps.
7	**Implementation**	Implement the observed structure and behavior in frontend and backend code.
8	**Testing and Revision**	Inspect local test results and revise the implementation or plan.

Table 12 describes nine categories of reconstruction behavior informed by theories of human cognition. Understanding the task relates to goal representation and problem formulation (Newell & Simon, 1972). Inspecting the interface and exploring the reference draw on theories of visual representation (Marr, 2010) and active perception (Aloimonos et al., 1988), respectively. Mapping website structure relates to program comprehension (Von Mayrhauser & Vans, 1995); understanding workflows relates to mental model theory (Johnson-Laird, 1983); and predicting action effects relates to causal reasoning (Pearl, 2000). Planning draws on human problem-solving theories (Miller et al., 1960), implementation on reflective design (Schön, 1984), and testing and revision on metacognition and cognitive monitoring (Flavell, 1979).

These nine categories guide qualitative interpretation of recorded behavior. Tool allocation is summarized separately using four activity groups: exploration, implementation, validation, and other operations. In the exploratory analysis in Section 5.2, 20 reconstruction trajectories contained 13,359 tool calls. Of these, 54.8% were classified as exploration, 10.0% as implementation, 27.0% as validation, and 8.2% as other operations. Before the first implementation edit in each of the 17 eligible trajectories, agents attempted 121 of 122 core workflows, but only 4 of 122 reached their final outcomes. The analysis counts recorded actions and outcomes using the definitions below. Figure 9 reports reference-interaction tool calls and observed edit–test–revise cycles for these 20 Grok-4.6 regular runs.

G.2 Classifying and Counting Recorded Actions

We assign the nine qualitative categories using recorded actions and files produced by the agent. Reference screenshots, reported interface state, navigation, and interactions show interface inspection and exploration. Notes about pages and components, followed by code changes, show website mapping and implementation. Explicit plans, local tests, error inspection, and revisions show planning, testing, and revision. To classify an action as understanding a workflow or predicting its effects, we require a recorded prediction or implementation of the relevant state transition or dependency. Each

record identifies whether the agent inspected the reference or its candidate. An event may receive multiple qualitative labels. The four activity shares are a separate summary of tool allocation and are not computed by summing these overlapping labels.

Tool-call coding rules. We use each recorded tool call as the counting unit and assign it to one of four mutually exclusive activity groups. Classification uses the requested operations, their targets, and available tool outputs, rather than the tool name alone.

- **Exploration.** Calls that inspect or interact with the reference website to obtain information about its appearance, content, or behavior. These include navigation, screenshots, interface-state inspection, and interactions that probe workflows.
- **Implementation.** Calls that create or modify the candidate's source code, styles, assets, or application configuration. Revisions after local testing remain implementation calls. Files generated by builds, screenshots, and test logs do not by themselves constitute implementation edits.
- **Validation.** Calls that check the candidate through builds, runtime checks, browser interactions, HTTP requests, or inspection of test results and diagnostic logs. Comparing candidate observations with the reference also counts as validation. These checks use information available during reconstruction, excluding hidden evaluation.
- **Other operations.** Calls that perform supporting operations without meeting the definitions above, including environment preparation, directory listing, and general local-file inspection. Reading candidate source files without editing them falls in this group; inspecting test results or diagnostic logs falls under validation.

Calls containing multiple activities. A call containing multiple commands or browser actions counts once. When a call meets more than one activity definition, we apply the fixed priority order: implementation, validation, exploration, then other operations. Thus, a call that edits source code and runs a local check is assigned to implementation. A call that inspects both the reference and the candidate without editing is assigned to validation. A call combining reference exploration, source editing, and local checks is assigned to implementation. This rule prevents double counting; it does not imply that the assigned activity occupies most of the call's execution time.

Counting and aggregation. Failed calls and retries are counted according to their requested operations, with each recorded retry counted separately. A missing result does not prevent classification when the request identifies the operation and its target. Calls with insufficient evidence for classification are reported separately rather than assigned to other operations. Each activity share is its call count divided by the total number of classified calls pooled across the analyzed trajectories. These shares measure call allocation, not elapsed time, successful outcomes, or the number of actions within each call.

Timestamped events identify reference exploration, implementation, candidate testing, and revision. These activities may interleave within a run. Interaction counts and completed revision rounds are summarized per run; activity shares use tool events or elapsed time as their denominator. Each summary reports its sample size and the number of missing or truncated logs. Missing observations remain separate from recorded zero activity.

The present trajectory analysis describes activity in one agent configuration; Equation 6 defines the reconstruction score.

Figure 9 uses the final regular-run trajectories from a fixed dataset version. A reference-interaction count is the number of tool calls requesting navigation, clicking, typing, or other browser interactions on the reference. Batched actions count once per tool call; failed requests are included, while observation-only calls are excluded. A completed edit–test–revise cycle requires a successful candidate source edit, a completed local browser, HTTP, or runtime check, and a later successful source edit, in strictly increasing trajectory steps. Source editing is sufficient for the implementation step; compilation is optional. Repeated checks without another edit count once. A cycle records a revision following a local check, regardless of whether the revision fixes the problem. The Petfinder and Target logs each lack the result of the final tool call used for implementation. Their observed cycle

counts are retained as lower bounds in the accompanying data and excluded from the cycle distribution, leaving 18 runs. Reference-interaction requests remain available for all 20 runs. Tool-activity shares retain the pooled denominator of 13,359 calls reported in Section 5.2.

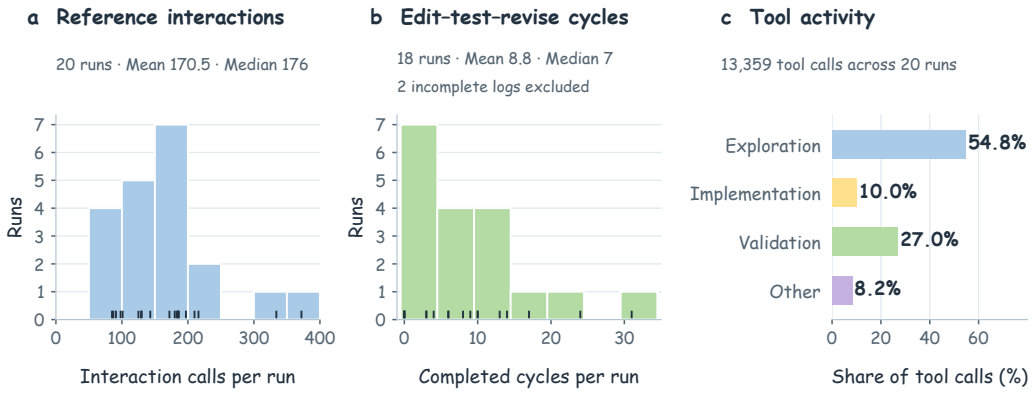


Figure 9: Recorded trajectory diagnostics. (a) Reference-interaction tool calls per run ($N = 20$); each batched call counts once. (b) Observed edit–test–revise cycles per run ($N = 18$); two logs with missing final tool results are excluded. Short marks show individual observations in (a) and (b). (c) Previously reported activity shares across 13,359 tool calls in the same 20 runs, computed by pooling tool events across runs. Local tests use only information available during reconstruction.

G.3 Qualitative Reconstruction Analysis

A reconstruction case study connects what the agent observed with what its submitted implementation did. The analysis uses the reference observations, generation trace, saved candidate, and test execution from a documented trial. It follows a particular workflow from the agent's observations to the first visible difference between reference and candidate.

The illustrative workflow in Table 2 explains test design. The recorded examples in Appendix I.2 examine agent behavior using observations, actions, candidate code, and available test records. Screenshot comparisons match route, viewport, authentication, and application state.

Failure analysis links each mismatch to the available evidence. Reference observations show what the agent had seen; candidate code and execution show what it implemented. Cases with insufficient evidence remain unresolved.

H INTERPRETATION AND REPRODUCIBILITY

The benchmark measures agreement with tested reference behaviors under a fixed observation interface and resource budget. In the main setting, agents may reuse reference media, while JavaScript and CSS remain inaccessible. Performance therefore reflects the ability to reconstruct appearance and behavior from interactive observations, permitted media, and public task specifications. Evaluation follows fixed execution and screenshot protocols; the sensitivity of results to rendering variation and test selection is examined separately.

Reproduction requires the generation instructions and tool interface, reference environment, candidate artifacts, test definitions, evaluator, and analysis inputs associated with each reported result. Released records include these identifiers, the relevant configuration, and evaluation status. Traces are released only after removing private credentials and personal data, with any redactions documented.

Analysis records identify the trials contributing to each aggregate and document all exclusions. They also distinguish original evaluation, candidate reevaluation, and offline rescoring, allowing reported statistics to be recomputed from the released data.

I SUPPLEMENTARY ANALYSES

This section reports reconstruction case studies, alongside analyses linked to the main text. The figures cover human trajectories (Figure 4), exploration and implementation gaps (Figure 5).

I.1 ANALYSIS 1: HUMAN TRAJECTORIES AND ACTIVE EXPLORATION

The analysis and its schematic are presented in Section 5.1. The paired score comparison includes 18 websites and gives each website equal weight. The exploration diagnostic in Section 5.2 instead pools workflows across 17 trajectories that reached a first implementation edit, counting observations before that edit.

I.2 ANALYSIS 2: EXPLORATION AND IMPLEMENTATION GAPS

The aggregate analysis and its schematic are presented in Section 5.2.

CASE STUDIES OF EXPLORATION AND IMPLEMENTATION GAPS

The following three examples connect reference observations, implementation decisions and recorded outcomes. All three runs used `grok-4.6`. Identifiers `sN:cK` use zero-based indices for the trajectory step and tool call; test identifiers denote separate evaluation records.

Fandango: a successful local booking check `grok-4.6`

Task. Reconstruct the reference website through independent exploration.
Reference behavior. Book three seats, E2–E4, for *Mutiny* at Regal Union Square, totaling $67.87, and reach confirmation and ticket history.

Observed actions and local test.

`s49:c1`, `s50:c0`, `s51:c0`: observe confirmation. Review the reference booking, click confirmation and inspect the resulting state. The reference confirms seats E2–E4 for $67.87.

`s224:c0`, `s228:c0`: implement the catalogue and server. Write the candidate catalogue and server, including the `/api/checkout/confirm` endpoint.

`s233:c0`, `s235:c0`: exercise the local API. The recorded test reaches confirmation with HTTP 201 and a total of 6,787 cents. The booking is stored, the selection is cleared, and the stored booking is checked.

Final state. The saved implementation contains the confirmation endpoint and the client code that calls it. The server saves the booking and clears the selection. The client connects its confirmation button to the API and redirects on success. The recorded local API check completed successfully.

Interpretation. The candidate reproduced the observed booking outcome in the local API test: it stored the booking and cleared the selection.

AMC Theatres: the URL query does not select the tab `grok-4.6`

Task. Reconstruct the reference website through independent exploration.
Reference behavior. The reference exposes a Coming Soon navigation state and movie list.

Observed actions and query handling.

`s64:c0`: observe the Movies page. Inspect the Coming Soon navigation, filters and a 23-movie listing.

`s100:c0`, `s101:c0`: probe the Coming Soon URL. Open `/movies?tab=coming-soon` and capture the resulting page.

`s218:c0`, `s221:c0`: implement the catalogue and server. Add the movie catalogue and the request handler for the Movies page.

`s233:c2`: implement tab switching. Modify the frontend to read `data-movies-tab` and apply the selected tab.

Final state. The candidate's `page_movies` implementation does not parse the `tab` query parameter, and inspection found no parser or alias for `movie-list`. The evaluation records report failed checks for this interface.

Interpretation. The catalogue and client-side tab switching were implemented, but opening a URL with a tab query did not select the requested state. Reproducing the visible controls therefore left the observed URL behavior unimplemented.

PetSmart: correct labels with incorrect images `grok-4.6`

Task. Reconstruct the reference website through independent exploration.
Reference behavior. Fish, Bird and Small Pet category cards show the corresponding animals.

Observed categories and image selection.

`s8:c0`: observe the reference homepage. The screenshot pairs the category labels with the corresponding animal artwork before implementation.

`s280:c1`: write the catalogue. Assign specific asset files to the category keys.

`s283:c0`: write the renderer. Construct category cards whose labels select images through `img(key)`.

`s324:c0`: inspect the candidate homepage. The last captured homepage shows dog artwork under Fish, Bird and Small Pet.

`s325:c0--c2`, `s326:c0`: finish local work. Inspect the source and rebuild the files needed to run the website. No subsequent repair of the category-image mapping is recorded.

Final state. The homepage connects each label to its category key, but the keys select unrelated assets: two dog "Deals" images and a Dalmatian grooming image. The lookup helper prefixes each selected filename with `/static/source/`. The incorrect category-to-file assignments account for the images seen in the candidate screenshot.

Interpretation. The reference screenshot showed the correct category images before the agent assigned the candidate assets. The recorded mismatch therefore arose in the implementation. The evaluation record does not link this defect to specific tests.

Takeaway. Checking each rendered image against its category label would expose the incorrect asset assignments.

I.3 Analysis 3: Which Websites Are Hardest to Rebuild?

We compare reconstruction scores across the 20 websites analyzed in Section 5.2. Scores describe performance under the fixed Grok-4.6 configuration, including its exploration, implementation, and runtime delivery. Compilation and runtime-contract failures receive zero under the failure-accounting rule and can prevent functional evaluation. The all-site summaries below include zero scores; the category means and rank correlations are restricted to websites with nonzero scores.

The selected 20 sites are classified into categories based on their properties: eight shopping, ten service, and two websites included in both categories. Under the fixed agent configuration, reconstruction scores were low across website categories. Among websites with nonzero scores, the two shopping websites had a mean score of 6.70% and the seven service websites 6.73%. The difference of 0.04 percentage points is smaller than either group's sample standard deviation, 0.98 and 6.20 percentage points respectively. These results suggest that there is no clear relationship between website category and reconstruction difficulty.

Page count showed no consistent relationship with score among the websites with nonzero scores. Therefore, we define two structural measures for each reference site and examine difficulty from both angles. Depth is the number of steps a typical workflow on the site takes; breadth is the number of distinct pages and workflows the site has. Both are counted from the pages and workflows for Section 5.2. Depth ranges from 2 to 7.5 steps, breadth from 10 to 32 pages and workflows, and the two are only weakly related. Deeper sites score lower. Mean score is 5.46% for the five shallowest sites, 2.75% for the seven in the middle, and 1.00% for the eight deepest, and among the eight sites with nonzero scores depth and score are negatively rank-correlated (Spearman $\rho{=}-0.81$, $p{=}0.016$). Breadth shows no such relationship ($\rho{=}-0.14$). A site is hard to rebuild not because it does many things but because each thing takes many steps. This is the same pattern as in Section 5.2, seen from the site's side. Agents seldom follow a workflow to its end, and the sites with the longest workflows are the ones they rebuild worst.

Through inspecting the trajectories and failed tests, we identified several common failure modes. Runtime delivery failures prevented functional evaluation when candidates failed compilation or did not satisfy the required runtime contract. These failures should be considered separately because they concern packaging and execution rather than the website's functional behavior. Workflow breaks interrupted continuous sequences of user actions. Content mismatches affected page appearance, text, product details, or element locators, while interface mismatches involved incorrect status codes, response fields, or state updates.

The failure examples reveal recurring difficulties in maintaining state throughout complete workflows. Fandango's candidate returned 404 for both the ticket submission and the seat-selection route. Medium's candidate had no locatable email input, sent no verification email, and failed the session checks that followed. Tripadvisor's candidate failed the review workflow at submission, at readback of the submitted review, and again when the review had to survive a restart. These examples show why reproducing individual pages does not ensure successful reconstruction. Later interactions depend on choices and application state established earlier.