WEBMOCKGYM: BENCHMARKING WEB RECONSTRUCTION WITH AUTONOMOUS INFORMATION GATHERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous software development requires agents to determine what they need to know, acquire and organize relevant information, and use it to produce reliable implementations. Evaluating end-to-end development calls for tasks that leave exploration and implementation decisions to agents while providing clear criteria for judging the final result. Existing web-development benchmarks often supply screenshots or interaction recordings, offering limited evidence about development when agents must gather reference information themselves. We introduce **WebMockGym**, a benchmark of 50 websites and 50 web apps for studying autonomous development through website reconstruction. Agents explore a frozen reference website and build a separately runnable reconstruction of its entry page and captured first-hop pages. The reference provides an observable target, while agents decide what to inspect and how to implement and check their work. Fixed, hidden tests assess content, images, visual appearance, and cross-page navigation. The same reference-defined requirements apply to every reconstruction of a task, including requirements an agent may overlook. Experiments with six coding-agent systems reveal frequent navigation failures, and no system passes every applicable test on any scored task in the website track. Trajectory analysis associates earlier writing and more frequent inspection of the reconstruction followed by further writing with higher Web scores. WebMockGym provides a concrete setting for studying autonomous development through the quality of independently runnable reconstructions.

1 INTRODUCTION

Autonomous software development requires agents to take responsibility for the process from understanding a task to delivering a working implementation. They must obtain the information needed for development, decide how to implement the solution, and check whether the result meets the goal. Evaluating this capability therefore calls for tasks that let agents make these decisions while assessing their final outputs against fixed requirements, including those they may overlook during development.

Existing benchmarks such as Design2Code (Si et al., 2025), IWR-Bench (Chen et al., 2026b), and Interaction2Code (Xiao et al., 2025) evaluate interface reconstruction from supplied screenshots, interaction videos, or annotated interaction prototypes. These settings assess how well agents implement interfaces from provided reference information (Table 1). WebArena (Zhou et al., 2024) and VisualWebArena (Koh et al., 2024) evaluate agents interacting with existing websites. These settings assess implementation from supplied observations or task execution within an existing environment. We therefore ask: *how well can agents deliver implementations that satisfy fixed requirements when they must manage information gathering, coding, and verification themselves?*

To study this question, we introduce **WebMockGym**, which uses website reconstruction as a concrete setting for evaluating end-to-end development. WebMockGym comprises 50 websites and 50 web apps, covering over 4,500 pages. As shown in Figure 1, agents explore a frozen reference and write and run code to build an independently runnable reconstruction. They can revisit the reference and use local previews and execution feedback to check and revise their work. To assess the

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.

Table 1: Comparison with web reconstruction and generation benchmarks. ● full support, ◐ partial, ○ none, – not applicable. *Self-acq.*: the agent decides what reference information to collect by browsing; *Hidden reqs.*: some requirements (e.g., links behind menus) are revealed only through interaction; *Determ.*: executable tests without LLM/agent judges; *Validated*: both human agreement and controlled-defect studies.

Benchmark	Self-acq.	Hidden reqs.	Multi-page	Determ.	Tests hidden	Validated	Agentic	Stateful	#Tasks	#Pages	#Tests
Design2Code (2025)	○	○	○	●	–	◐	○	○	484	484	–
Interaction2Code (2025)	○	○	○	◐	–	◐	○	●	127	127	374
IWR-Bench (2026b)	○	○	◐	○	–	◐	○	●	113	100	1,001
UI2App (2026a)	○	○	●	◐	–	◐	○	●	45	327	–
WebGen-Bench (2025)	○	○	◐	○	●	◐	●	●	101	–	647
WebCoderBench (2026)	○	○	◐	◐	●	◐	◐	◐	1,572	–	–
Vision2Web (2026)	○	○	●	○	●	◐	●	●	193	918	1,255
Cloning Bench (2026)	◐	○	◐	◐	○	○	●	●	1	–	33
WebMockGym	●	●	●	●	●	●	●	◐	**100**	**4,564**	**504K**

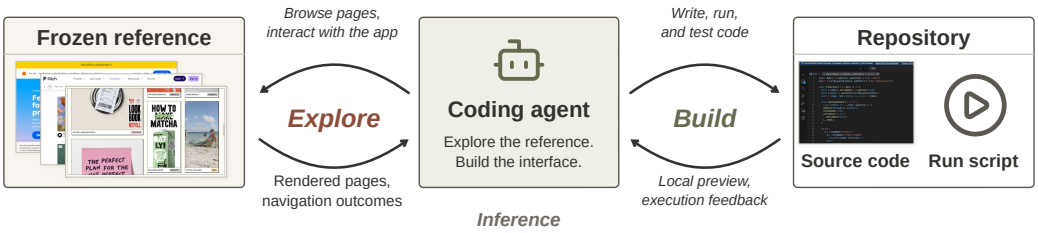


Figure 1: Development workflow in WebMockGym. Agents explore a frozen reference and iteratively build and check an independently runnable reconstruction.

resulting implementations, we construct fixed tests from the frozen references before development. Approximately 500,000 tests across the benchmark assess content, images, visual appearance, and cross-page navigation. These tests remain hidden during development, and every reconstruction of a task is evaluated against the same requirements, including those an agent may overlook during exploration or implementation.

All six coding-agent systems achieve higher overall scores than a baseline (wget), which mirrors websites by downloading pages and resources. On Web, agent scores range from 15.8 to 41.7 out of 100, compared with 13.3 for wget, while agent scores are more closely grouped on App. Navigation is the lowest-scoring family for most systems, and only 3.9% of reconstructions pass every test on App tasks that include navigation checks. Navigation requires the rebuilt pages and their links to work together, making cross-page completeness a priority for improving reconstruction (Section 3.2).

Trajectory analysis identifies development patterns associated with stronger reconstructions. On Web tasks, earlier writing and more frequent inspection of the reconstruction followed by further writing are associated with higher scores across page quality and navigation. On App tasks, the association with inspection followed by writing is clearest for the layout and position checks within Perception (Section 3.3).

Our contributions are:

- We introduce WebMockGym, a benchmark of 100 website and web-app reconstruction tasks in which agents gather reference information through browser interaction and use it to build separately runnable implementations.
- We construct fixed, hidden tests of reconstructed pages and navigation, including links revealed by interaction. We validate the evaluator through human judgments and controlled changes to reconstructed websites.
- We evaluate six coding-agent systems and identify development patterns associated with higher reconstruction quality on Web tasks and better layout and positioning on App tasks.

Table 2: Dataset composition (100 tasks and four types per track).Minimum, median, and maximum page counts are per task. Task types group websites by their public-facing pages and apps by their primary interactions.

Web type	Tasks	App type	Tasks	Pages	Web	App
Product platforms	22	Data transforms	16	Min.	17	1
Consumer services	11	Forms/workflows	14	Median	76.5	6
Workflow tools	9	Content editing	10	Max.	298	80
Brands/catalogs	8	Browse/explore	10	Total	3,826	738

2 WEBMOCKGYM

2.1 TASK FORMULATION

Each task asks an agent to explore a frozen reference website and build a local reconstruction that runs independently of it. We denote the starting page, or entry page, by p_0. The first-hop pages $\mathcal{P}_1$ are the saved pages on the same site reachable by following one link from p_0. The reconstruction scope is

$$\mathcal{P} = \{p_0\} \cup \mathcal{P}_1. \tag{1}$$

Same-site scope includes first-party subdomains, and revealing a link within p_0 does not add a page navigation.

The page set $\mathcal{P}$ is fixed before development, and the agent discovers its pages through browser interaction. It can revisit the reference while implementing the pages and their connections, with all evaluation tests and pass conditions hidden throughout development. The final site must serve its own pages and resources and is checked against tests derived from the reference. Section 3.1 describes the development conditions.

2.2 DATASET CONSTRUCTION

We select reference sites to vary the content and page structures that agents must reconstruct. The collection contains 50 public websites and 50 web apps; Table 2 summarizes their types and captured page counts.

Web selection. We select public websites with usable captures of page content and resource responses, allowing their retained pages to be browsed through local replay. Within this eligible pool, we seek coverage across functional domains and website types. We limit repeated examples from overrepresented categories. The 50 selected websites span 23 functional domains. For workflow-oriented products, we capture their public website pages.

App selection. We select web applications to cover different functions and interaction structures. Selection prioritizes less represented categories and limits repeated source projects, retaining multiple tasks from the same project when they provide distinct interaction types. The resulting 50 tasks use the applications' captured interfaces and navigation as reconstruction targets.

Freezing the reference. We freeze the captured pages and their network responses so that agents encounter the same reference content across runs. Each page is saved as rendered HTML, with its network requests and responses recorded in an HTTP Archive (HAR). The capture also pairs link labels with destinations and records the actions that reveal initially hidden links. A local replay server serves the saved HTML and recorded resource responses, keeping the reference content independent of later changes to the live website. We include a task in the benchmark once its saved reference passes every applicable test. The reference and test requirements are then fixed for all submissions.

2.3 TEST CONSTRUCTION

We derive a fixed test suite from the saved reference to evaluate reconstructed pages and the navigation between them (Figure 2). Table 3 lists the test types and counts.

Table 3: Dataset statistics and test counts by scoring family. Page load and Local links are reported separately as dataset statistics. Each test is counted once per task.

	Dataset statistics		Evidence	Image	Perception				Navigation	Summary		
Track	Page load	Local links	Element presence	Images	Position	Color	Layout	Screenshot	Navigation	Total	Median / task	Min–max / task
Web	3,744	3,744	141,468	23,896	153,439	18,301	44,928	3,744	48,340	**441,604**	8,488.5	1,755–34,365
App	728	728	19,624	1,382	21,869	2,854	8,736	728	5,983	**62,632**	308.5	20–7,917

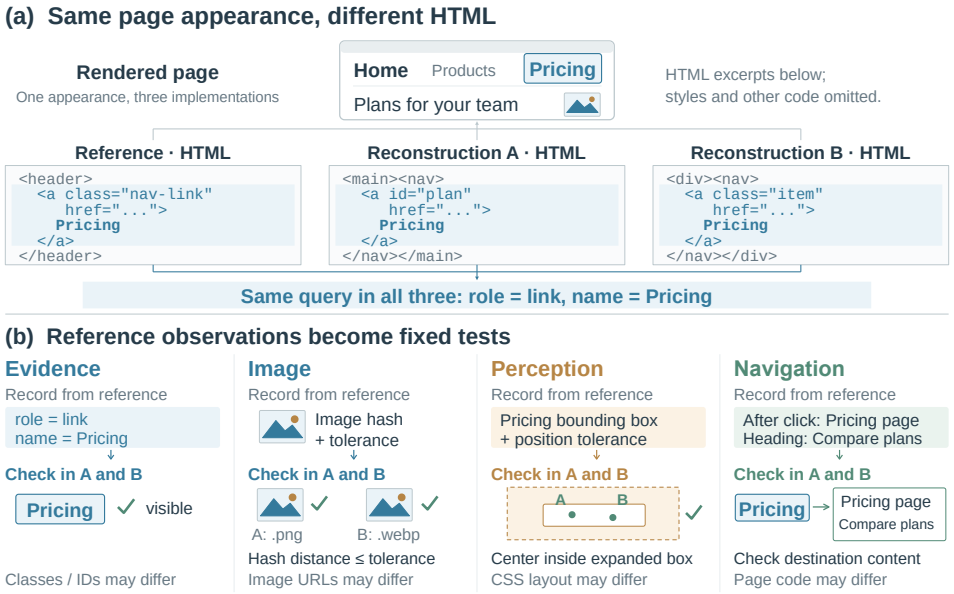


Figure 2: Schematic examples of fixed tests across different website implementations. (a) Three HTML implementations can produce the illustrated page appearance; styles and other code are omitted. Pricing is the visible label of the clickable link. The query role = link, name = Pricing finds it by component type and label. (b) Each column pairs a reference observation with a check applied unchanged to reconstructions A and B. The dashed region expands the reference rectangle by the position tolerance; green dots mark reconstructed link centers. Navigation checks that the link opens the intended page and displays its distinctive content.

Page checks. Page checks compare the reconstructed content and appearance with observations recorded from the reference. Evidence checks whether required elements are present, while Image compares fingerprints of rendered images. Perception checks element positions and compares the page's appearance with the reference. Core heading and link checks identify elements by their function and browser-reported label (role and accessible name), allowing different code structures.

Navigation checks. Navigation checks preserve the actions needed to reach a captured destination within $\mathcal{P}$. For an initially hidden link, the test performs the recorded revealing action before locating and activating the link. It then checks that the corresponding local page opens and displays the recorded distinctive content. These checks form the Navigation family in both tracks.

Reference validation. We calibrate the generated suite on the saved reference before fixing its tests and pass conditions for evaluation. Tasks enter the benchmark after passing the reference validation described in Section 2.2. Appendix A.3 gives the capture and matching rules, and Appendix A.2 illustrates the resulting checks.

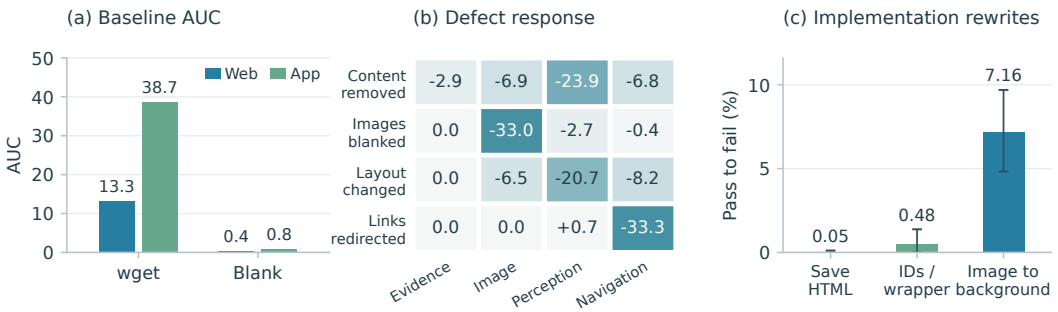


Figure 3: Evaluator responses to degraded pages and code changes. (a) AUC for wget and blank pages on Web and App. (b) Changes in family scores in percentage points. Rows identify the introduced defects and columns identify the scoring families. (c) Percentage of initially passing checks that fail after code rewrites. Error bars show 95% confidence intervals.

2.4 Evaluation metrics

We report scores for four dimensions: Evidence, Image, Perception, and Navigation. We compute each family's pass rate per task and average it across tasks where that family applies. Missing required pages or elements count as failures.

Our primary metric, AUC, rewards reconstructions that perform well in every applicable family, reflecting the need for faithful pages and working navigation. We average each task's lowest applicable family score, so a task with entirely broken navigation scores zero even if its pages look accurate. This equals the area under the curve of task pass rate against the threshold required in every applicable family; scores range from 0 to 100.

All-pass reports the percentage of tasks passing every applicable test. Entry and FirstHop apply AUC to the starting page and captured pages one link away, respectively. Appendices B.2 and B.5 give the calculation and page-scope details.

2.5 Evaluator validation

AUC rankings show positive agreement with human judgments, and controlled image and link defects reduce the corresponding family scores.

Human evaluation. AUC rankings show positive agreement with expert judgments of reconstruction quality. Three PhD participants each ranked six reconstructions on 40 tasks. Mean Kendall's τ_b is 0.705 between automated and expert rankings, while agreement between experts is 0.710. Appendix C.2 reports the aggregation details and additional agreement measures.

Response to controlled defects. Controlled image and link defects reduce the scores of the corresponding test families. The paired study modifies 39 pages across 20 tasks and compares each modified version with an intact control under the same tests. Medium image and link defects reduce Image and Navigation scores by 33.0 and 33.3 percentage points (Figure 3b). Blank pages also receive near-zero AUC scores on both tracks (Figure 3a).

Tolerance to implementation changes. HTML reserialization and conservative structural edits preserve almost all initially passing checks. Among pages passing the text and screenshot screen, parsing and saving HTML yields a task-mean pass-to-fail rate of 0.05%, and renaming identifiers or adding wrappers yields 0.48% (Figure 3c). Replacing image tags with same-size CSS backgrounds yields a 7.16% rate, with failures concentrated in image checks. These failures identify image representation as a source of sensitivity. Appendix C.1 specifies the rewrite operations and screening criteria.

Table 4: Reconstruction quality and per-task cost on Web and App. GLM costs exclude cache discounts; DeepSeek and other costs include them (Appendix B.4).

Configuration	Evidence	Image	Perception	Navigation	AUC	*All-pass*	Entry	FirstHop	Cost (USD)
Web-50									
wget	56.9	22.3	33.3	28.9	13.3	0.0	8.7	13.7	–
GLM-5.3-Flash	83.0	51.5	63.8	46.5	33.6	0.0	25.3	33.5	2.23
DeepSeek V4 Flash	82.0	53.3	65.2	43.7	33.4	0.0	29.5	33.2	0.97
Claude Opus 5	**84.8**	**61.5**	**68.4**	**53.3**	**41.7**	0.0	**36.5**	**41.7**	25.64
Kimi K3	79.6	45.4	54.3	39.8	27.0	0.0	23.5	27.0	11.43
GPT-5.6-sol	60.5	28.3	41.2	37.9	20.7	0.0	19.6	20.3	4.32
Claude Sonnet 5	61.3	25.2	35.9	31.6	15.8	0.0	14.4	15.7	5.69
App-50									
wget	60.7	47.2	57.0	43.7	38.7	4.0	40.6	39.0	–
GLM-5.3-Flash	75.5	56.9	76.3	40.5	45.0	10.0	39.0	43.1	1.45
DeepSeek V4 Flash	74.1	62.9	74.9	50.2	**51.9**	12.0	38.8	46.6	0.27
Claude Opus 5	74.2	61.8	75.4	44.9	49.7	**16.0**	38.6	**48.3**	7.37
Kimi K3	**76.5**	**64.5**	**77.2**	47.2	48.2	14.0	38.6	44.8	5.14
GPT-5.6-sol	71.6	58.6	74.6	**52.2**	50.8	12.0	**41.8**	48.2	3.17
Claude Sonnet 5	72.0	60.2	71.4	49.7	48.4	14.0	40.7	46.9	4.52

Takeaway. Rankings from the hidden tests agree with expert rankings (mean Kendall's $\tau_b = 0.705$), close to the agreement between experts (0.710). Controlled image and link defects reduce the corresponding scores by about 33 points. Among rewrites that pass the text and screenshot screen, HTML reserialization and identifier or wrapper edits produce task-mean pass-to-fail rates below 0.5%. Replacing image elements with CSS backgrounds produces a higher rate of about 7%, with failures concentrated in image checks.

3 Evaluation

3.1 Experimental setup

We evaluate how well coding agents reconstruct websites when they gather the reference information themselves. All agents and the wget baseline are evaluated on the same 50 Web and 50 App tasks.

Agents and baseline. We compare six coding-agent configurations with wget, a baseline that mirrors websites by downloading pages and resources (Table 4). GPT-5.6-sol run in Codex, while the other five use Claude Code. These coding environments provide browser and shell tools for exploration and implementation.

Development conditions. Each task starts from an empty workspace with browser access to the frozen reference website. Agents choose when to consult the reference and when to inspect their local implementation during development. They may reuse accessible content and assets, while the evaluation tests and pass conditions remain hidden. The task instruction asks agents to finish once they judge that the required pages have been faithfully reconstructed (Appendix A). Recorded development time and token use are reported in Appendix B.3, with costs in Appendix B.4.

Evaluating final reconstructions. We score one final implementation per agent–task pair by running the hidden tests on its local deployment with JavaScript enabled. Every reconstruction of a task is evaluated with the same checks and pass conditions, including the wget baseline. Each check starts with a 1.5-second timeout and can extend to 5 seconds while the requested navigation or an operation on a located target remains unfinished. Checks are not retried. Scores follow the definitions in Section 2.4; Appendix B.1 gives the execution details.

3.2 Main results

Agents improve on mirroring and differ most on Web. All six agents outperform wget on Image and Perception in both tracks (Table 4). The agents' AUC scores span 15.8–41.7 on Web and

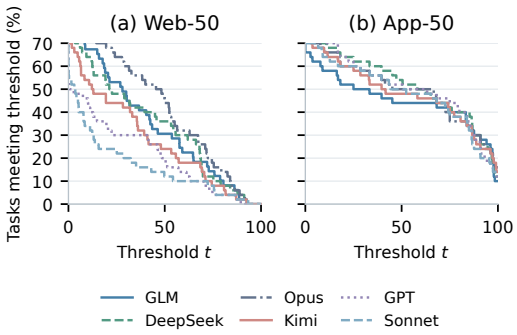


Figure 4: Tasks meeting threshold t in every applicable test family. At $t = 100$, the rate equals All-pass.

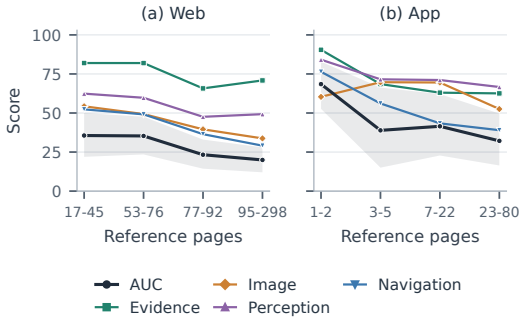


Figure 5: Scores by reference page count. Lines connect group means; shading shows pointwise 95% confidence intervals for AUC.

45.0–51.9 on App, showing substantially greater separation between systems on the website tasks. Requiring at least 80% in every applicable test family leaves 4–14% of Web tasks and 36–42% of App tasks passing (Figure 4). These thresholds expose the remaining gap between improved individual family scores and meeting the requirements of an entire reconstruction.

The entry page remains a substantial source of error. Every agent scores lower on the starting page (Entry) than on pages one link away (FirstHop) in both tracks (Table 4). When the comparison uses test families present in both scopes, mean entry-minus-first-hop gaps remain -5.8 points on Web and -10.1 points on App. Navigation checks count toward the page containing the link, so Entry also measures whether the starting page connects correctly to its destinations. The lower Entry scores highlight the need to verify the content and outgoing links of the page available from the start of development. Appendix B.5 gives the detailed comparisons.

Scores generally fall as reference size grows. On Web, Image and Navigation decline most as reference size grows. Mean AUC falls from 35.6 in the 17–45-page group to 19.9 in the 95–298-page group (Figure 5). Image falls from 54.3 to 33.7 and Navigation from 52.3 to 29.1, with both declining in every successive group. On App, AUC falls from 68.5 for 1–2-page tasks to 32.2 for 23–80-page tasks. Smaller App tasks also often lack image or navigation checks, so this trend combines changes in test coverage and page count. Appendix B.5 gives the grouping procedure and the comparison restricted to tasks with all four test families.

Takeaway. Complete reconstruction remains difficult across the required pages: the best Web AUC is 41.7, and no Web output passes every test. Mean Web AUC also falls from 35.6 to 19.9 between the smallest and largest reference-size groups. These results motivate checking content and images across all required pages during development, including the entry page.

Navigation limits complete reconstruction. Navigation is the lowest-scoring family for four Web systems and all six App systems; Image is the lowest-scoring family for the other two Web systems. A navigation check requires the reconstructed control to reach the right local page and display its expected content, making success depend on both the source page and its destination. Agents therefore need to verify each reconstructed link together with the content at its destination.

App completion is concentrated in tasks without navigation checks. Of the 39 App reconstructions that pass every applicable test, 31 come from tasks without navigation checks. Among App tasks with navigation checks, 3.9% of reconstructions pass every test, compared with 32.3% on tasks without navigation checks. The higher App completion rate therefore comes largely from tasks whose evaluated requirements cover individual pages, making Navigation central to interpreting the overall result.

Page and link guidance. We test whether specifying the pages and links to reconstruct improves navigation. For five websites, an added prompt paragraph lists all page URLs in $\mathcal{P}$ and the actions that reveal hidden links. Agents use this guidance to browse the reference and build the local site, with all other development and evaluation settings held fixed. All six systems improve their mean Navigation scores on these websites by 22.80–53.02 percentage points (Table 5; Appendix B.6).

Table 5: Navigation scores on five websites (%).

System	Main	Guided
GLM-5.3-Flash	31.28	84.30
DeepSeek V4 Flash	46.55	83.90
Claude Opus 5	46.20	85.61
Kimi K3	23.72	67.61
GPT-5.6-sol	46.19	68.99
Claude Sonnet 5	28.43	60.13

Takeaway. Working page connections are central to complete reconstruction: App all-pass rates are 3.9% on tasks with navigation checks and 32.3% on tasks without them. Supplying page URLs and the actions that reveal hidden links raises Navigation by 22.80–53.02 percentage points across six systems in the five-site study. Agents should track which pages and links they have implemented and verify the content at each destination.

3.3 Agent actions and reconstruction quality

On Web tasks, higher reconstruction scores are associated with earlier file writes and more frequent inspection of the reconstruction followed by further writing. We study these relationships using 594 development runs from the same 50 Web and 49 App tasks across six agents, matching action sequences to final test scores. The comparisons account for differences between tasks and agents and adjust for the total number of tool calls. An inspection–write sequence consists of a successful browser inspection of the reconstruction followed by a file write; its frequency is the number of such sequences per 100 tool calls.

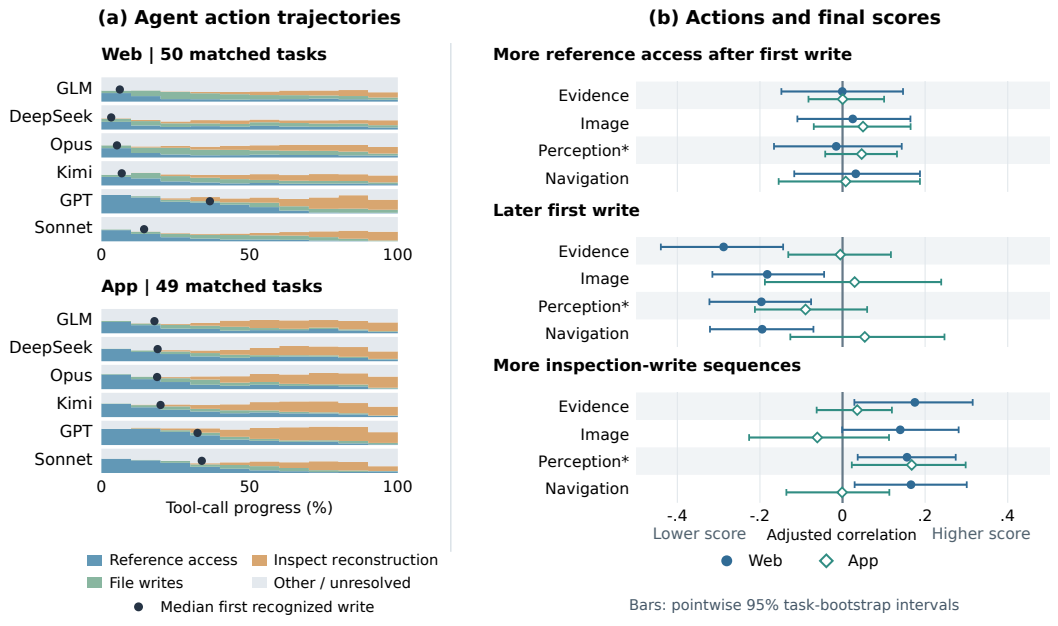


Figure 6: Development trajectories and their associations with reconstruction quality. (a) Action proportions over tool-call progress; dots mark the median first write. (b) Adjusted correlations with final scores. Evidence checks required elements and text; Perception* uses only the layout and position checks within Perception. Inspection–write frequency counts sequences per 100 tool calls. Positive values indicate higher scores as the named action measure increases. Bars show 95% confidence intervals.

Earlier writing and repeated inspection accompany higher Web scores. The associations with earlier writing and repeated inspection extend across all displayed scores (Figure 6(b)). Later first writes have negative adjusted correlations across all four displayed scores (-0.288 to -0.182), and more frequent inspection–write sequences have positive correlations (0.140 to 0.175). Local inspection gives agents a concrete implementation to assess and can guide what they revise next.

On App, inspection is most strongly associated with layout and position. On App, inspection–write frequency has an adjusted correlation of 0.17 with the layout and position checks within Perception and a near-zero correlation with Navigation (Figure 6(b)). First-write timing also has no consistent direction across App score dimensions. These differences motivate examining which requirements an agent checks and subsequently changes, especially how it verifies the connections between pages. Appendix B.7 defines the action measures and reports the statistical details.

Takeaway. On Web, later first writes correlate with lower scores ($r = -0.29$ to -0.18), while more frequent inspection–write sequences correlate with higher scores ($r = 0.14$ to 0.18). On App, inspection followed by writing has its clearest association with layout and position ($r = 0.17$). These associations motivate testing agent workflows that implement observed requirements early and use browser checks of pages and links to guide revisions.

4 Related work

Visual interface generation. Pix2code generates code from screenshots, while Design2Code benchmarks screenshot-based reconstruction (Beltramelli, 2018; Si et al., 2025). WebSight supplies synthetic screenshot–HTML pairs, and Web2Code combines webpage understanding with HTML generation (Laurençon et al., 2024; Yun et al., 2024). UI2Code^N uses rendered feedback for iterative refinement (Yang et al., 2025).

Interactive and multi-page website development. Interactive reconstruction uses videos in IWR-Bench and WebVR or before-and-after UI views in Interaction2Code (Chen et al., 2026b; Dai et al., 2026; Xiao et al., 2025). MRWeb supplies multi-page screenshots and resource lists; WebCoderBench uses text (Wan et al., 2024; Liu et al., 2026). Cloning Bench exposes recorded flows and their test harness during development (Vibrant Labs Research Team, 2026). Vision2Web and WebGen-Bench use supplied visual or textual requirements (He et al., 2026; Lu et al., 2025).

Coding and browser agents. SWE-agent supports repository work, while SWE-bench tests issue resolution (Yang et al., 2024; Jimenez et al., 2024). WebArena and VisualWebArena test browsing on existing websites (Zhou et al., 2024; Koh et al., 2024). ProgramBench evaluates reconstruction from executables and documentation (Yang et al., 2026).

Evaluation of reconstructed websites. WebVR uses human-aligned visual rubrics, WebGen-Bench uses browser-agent judgments, and Vision2Web combines workflow-constrained GUI verification with VLM visual judgments (Dai et al., 2026; Lu et al., 2025; He et al., 2026). FrontendBench uses predefined executable tests (Zhu et al., 2025). WebMockGym checks pages and navigation, not general full-stack behavior, and validates these reference-derived checks with controlled defects and human judgments.

5 Conclusion

WebMockGym gives agents responsibility for deciding what to inspect in a reference website and evaluates their reconstructions with fixed, hidden tests. Across six coding-agent systems, the highest Web AUC is 41.7, and no Web reconstruction passes every applicable test. Navigation is the weakest test family for all six systems on App and four on Web, highlighting the difficulty of connecting reconstructed pages into a working site. These findings motivate agent workflows that turn observations into implementations early and use browser checks of complete navigation paths to guide revisions. Benchmarks should connect records of discovered and implemented requirements with tests that cover the full task, including requirements an agent overlooks.

ETHICAL CONSIDERATIONS

Data sources and privacy. WebMockGym draws on publicly accessible websites and web application interfaces, with source URLs recorded for every task. The Web track captures public-facing pages, including product pages for commercial services. The App track uses public applications and project demonstrations, such as the Quill editor and TodoMVC, and records their source websites or repositories. For each reference page, we save rendered HTML and the network responses needed to replay it locally.

Before releasing the reference captures, we will remove cookies and authentication credentials from their network records. This cleanup covers stored cookie fields and cookie headers, along with authentication tokens carried in request headers or URLs. The captured pages and resources remain subject to the copyright and license terms of their original owners. Redistributed material will retain source attribution and applicable license notices.

Research use. WebMockGym supports controlled research on owner-authorized front-end reimplementation. Agents browse frozen references and build independently running local reconstructions, allowing repeated experiments with fewer visits to the original websites. Website reconstruction can support interface migration and can also be used to impersonate services or create phishing pages. Reconstructions shared outside the evaluation environment should be clearly labeled as research artifacts so that users can distinguish them from the original services.

Human evaluation. We report human judgments using participant identifiers. Participants compare reconstructed interfaces with the reference, and the reported results summarize their rankings and agreement.

REPRODUCIBILITY STATEMENT

Our anonymous submission repository is available at `https://anonymous.4open.science/r/test-generation-freezing-E61D`. Appendix A provides the agent instructions and representative tests. Appendix B specifies the evaluation procedures and score calculations.

AI USE STATEMENT

The research team prepared the data and conducted the core experiments. AI tools were used primarily for manuscript drafting and language polishing. The authors remain responsible for the research findings and the final manuscript.

REFERENCES

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2018. URL https://arxiv.org/abs/1705.07962.

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, 2026a. URL https://arxiv.org/abs/2607.06306.

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. URL https://arxiv.org/abs/2509.24709.

Jacob Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220, 1968. doi: 10.1037/h0026256.

Yuhong Dai, Yanlin Lai, Mitt Huang, Hangyu Guo, Dingming Li, Hongbo Peng, Haodong Li, Yingxiu Zhao, Haoran Lyu, Zheng Ge, Xiangyu Zhang, and Daxin Jiang. WebVR: Benchmarking

multimodal LLMs for webpage recreation from videos via human-aligned visual rubrics. *arXiv preprint arXiv:2603.13391*, 2026. URL https://arxiv.org/abs/2603.13391.

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. *arXiv preprint arXiv:2603.26648*, 2026. URL https://arxiv.org/abs/2603.26648.

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*, 2024. URL https://arxiv.org/abs/2310.06770.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. URL https://arxiv.org/abs/2401.13649.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024. URL https://arxiv.org/abs/2403.09029.

Chenxu Liu, Yingjie Fu, Wei Yang, Ying Zhang, and Tao Xie. Webcoderbench: Benchmarking web application generation with comprehensive and interpretable evaluation metrics. *arXiv preprint arXiv:2601.02430*, 2026. URL https://arxiv.org/abs/2601.02430.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. WebGen-Bench: Evaluating LLMs on generating interactive and functional websites from scratch. *arXiv preprint arXiv:2505.03733*, 2025. URL https://arxiv.org/abs/2505.03733.

Kenneth O. McGraw and S. P. Wong. Forming inferences about some intraclass correlation coefficients. *Psychological Methods*, 1(1):30–46, 1996. doi: 10.1037/1082-989X.1.1.30.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *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, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.199. URL https://aclanthology.org/2025.naacl-long.199/.

Vibrant Labs Research Team. Cloning Bench: Evaluating AI agents on visual website cloning. Research report, March 2026. URL https://vibrantlabs.com/research/cloning-bench.

Yuxuan Wan, Yi Dong, Jingyu Xiao, Yintong Huo, Wenxuan Wang, and Michael R. Lyu. MRWeb: An exploration of generating multi-page resource-aware web code from UI designs. *arXiv preprint arXiv:2412.15310*, 2024. URL https://arxiv.org/abs/2412.15310.

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, 2025. URL https://arxiv.org/abs/2411.03292.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, et al. Programbench: Can language models rebuild programs from scratch? *arXiv preprint arXiv:2605.03546*, 2026. URL https://arxiv.org/abs/2605.03546.

Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiale Cheng, Xiaotao Gu, and Jie Tang. UI2Code^N: Ui-to-code generation as interactive visual optimization. *arXiv preprint arXiv:2511.08195*, 2025. URL https://arxiv.org/abs/2511.08195.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *arXiv preprint arXiv:2406.20098*, 2024. URL https://arxiv.org/abs/2406.20098.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

Hongda Zhu, Yiwen Zhang, Bing Zhao, Jingzhe Ding, Siyao Liu, Tong Liu, Dandan Wang, Yanan Liu, and Zhaojian Li. FrontendBench: A benchmark for evaluating LLMs on front-end development via automatic evaluation. *arXiv preprint arXiv:2506.13832*, 2025. URL https://arxiv.org/abs/2506.13832.

A TASK PROMPT AND TEST EXAMPLES

A.1 AGENT-FACING INSTRUCTION

The following prompt specifies the reconstruction task given to Web agents. Workspace and website details are replaced by placeholders. The harness supplies its own additional instructions.

Task prompt: website reconstruction

```
You are in a blank clone workspace:
<CLONE_WORKSPACE>

Clone the frozen website for site: <SITE_ID>

The source website is already running. Do not start it yourself.
SOURCE_URL:
<SOURCE_URL>

Playwright browser tools are available for exploring and interacting with the
source website. Use them as you see fit.

Read, Write, Edit, Glob, Grep, and Bash are available for authoring, downloading
public source assets, building, previewing, and testing inside the blank clone
workspace.

If you choose to start a local preview server, use the reserved preview URL so
concurrent benchmark jobs do not conflict:
<PREVIEW_URL>

Recreate the entry page and every first-party HTML page that loads successfully
from the frozen source and is reachable from the entry page by one direct page
navigation through its links, menus, buttons, or other visible navigation
controls. First-party subdomains are in scope; external sites, downloads, and
fragment-only changes are not. Opening a menu or revealing a link does not add a
hop; the hop is the resulting page navigation. Discover these pages yourself. Do
not crawl beyond one page navigation from the entry page.

Create the clone inside this blank workspace, using index.html as the entry
point. Preserve the content, appearance, navigation, and interactions you judge
important.

The entry page and every in-scope one-hop page must be an actual page from the
original first-party website, recreated as a local page in the clone. Do not
satisfy this requirement by linking, redirecting, or navigating back to the
original domain or to SOURCE_URL. Every in-scope first-party navigation target
exposed by the cloned entry page must resolve to its corresponding local clone
page; navigation within the clone must remain entirely inside the local clone.

The finished clone must keep working after the frozen source server is stopped.
Do not leave HTML, CSS, or JavaScript dependent on SOURCE_URL, localhost replay
paths, or the original first-party site for pages or runtime resources. Before
finishing, verify that the delivered clone contains no such dependencies.

Do not directly access, copy, parse, or reuse benchmark-private tests,
capability manifests, HARs, recordings, reference artifacts, scoring outputs,
or prior clone outputs. These are evaluation data, not source website evidence.

Finish when you believe the entry page and its one-hop website have been
faithfully recreated.
```

The benchmark instruction was passed to `claude -p` or `codex exec`. An App prompt variant additionally requires a successful browser observation through Playwright before completion. Interrupted runs can receive a continuation prompt.

A.2 FROM REFERENCE EXPLORATION TO EXECUTABLE CHECKS

Recording the reference website. During reference collection, the recorder explores the original website to discover content hidden behind controls. It uses the reference page's structure to find interactive controls, then hovers, clicks, or scrolls to expose additional links. For each discovered link, it records its label and destination together with the action that revealed it. The recorder revisits available controls after interactions until it exhausts eligible controls or reaches its configured limits. Opening a menu occurs within the source page, so following a link from that menu still counts as a first-hop page transition.

Testing the reconstructed website. During evaluation, each test searches the reconstructed page for its target link by browser-reported role and accessible name, using `getByRole('link', {name: label})`. This matches the link's function and label without requiring the reference link's class names or DOM nesting. When the link needs to be exposed first, the test attempts the recorded opening action before selecting the visible link. Test 9 below illustrates a menu trigger located by button role and name.

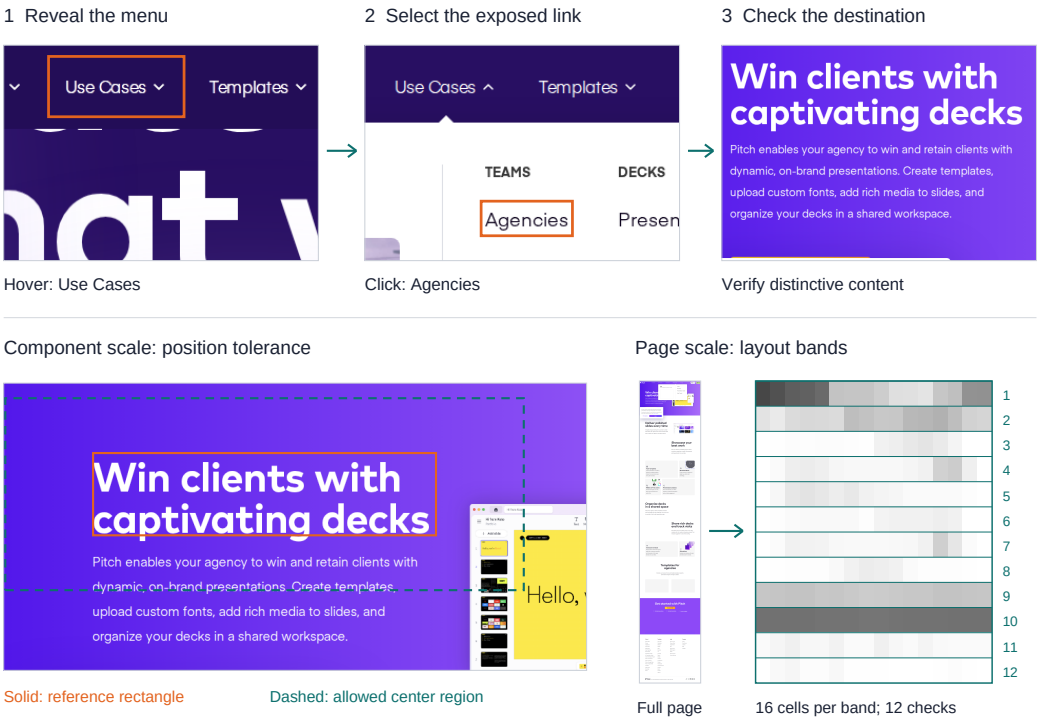


Figure 7: Reference-derived test construction on Pitch. Top: the *Use Cases* menu exposes the *Agencies* link, whose destination contains the heading *Win clients with captivating decks*. Bottom: a component rectangle and its allowed position region complement a coarse grid over the full destination page. Screenshots are rendered from the frozen reference; outlines and the grid are explanatory overlays.

Navigation example: reveal, select, and verify

The Pitch navigation check follows the path from opening the menu to verifying the destination page. Its construction record contains the source page, the *Use Cases* hover action, the link label *Agencies*, and the captured destination with its distinctive heading. The generated navigation check first attempts the revealing action, then requires a visible matching link and activates it. It checks the landing page against the recorded destination and its visible content. Here, *Agencies* is the accessible name used to find the destination link on the reconstructed page; the reference screenshot supplies the example's content and action sequence.

Perception example: component position and page coverage

For a selected heading, the position check records its rectangle and accepts a reconstructed center within the expanded region shown in Figure 7. Coordinates include page scroll offsets, so a component below the first viewport is measured in document space. Separately, the layout check partitions a full-page screenshot into a coarse grayscale grid and compares each horizontal band with its reference counterpart. The two checks address different errors: an individual component can be misplaced, or a page can reproduce its opening section while omitting much of what follows.

A.3 Capture scope and test matching

The recorder retains destinations under each site's capture rules and limits, defining the saved first-hop set $\mathcal{P}_1$.

Selecting content checks. The generator selects ordinary links by their labels and page regions. It merges repeated labels within each region and filters generic labels such as "Learn more", along with labels containing dates or changing counts. These rules determine which links are tested on the unchanged reference page. During evaluation, an ordinary-link presence check selects the first visible match. A missing or hidden target fails the check; its destination is assessed separately by navigation tests.

Matching images and positions. Image and position checks compare rendered properties recorded from the reference. Image tests use perceptual hashes of rendered images, allowing filenames and resource addresses to differ. A position check accepts an element's center within the reference rectangle expanded by a fixed fraction of viewport width and height. The reference rectangle uses the same matching rule, and coordinates include scroll offsets. Position checks select the first role-and-name match, so reordering identically named elements can change which one is measured.

Comparing whole-page appearance. Whole-page checks measure the appearance surrounding the individually tested elements. Color checks compare the proportions of selected colors, while screenshot checks compare pixels within fixed tolerances. Layout checks reduce a full-page screenshot to a coarse grayscale grid and compare each horizontal band with its reference counterpart. Checking bands separately allows missing lower-page sections to be detected alongside errors near the top.

Matching navigation destinations. Navigation checks require the corresponding local destination and its recorded distinctive content. Destination matching normalizes trailing slashes and HTML suffixes. It also accepts index filenames while preserving query strings and page identity. A failure to complete the required navigation counts as a failed test. In both tracks, navigation tests perform recorded control actions to reveal links and verify that their destinations open with the expected content.

A.4 Representative test assertions

The examples below show how generated tests check the reconstructed pages. The page-load, heading, and layout-band examples use named variables for reference-derived values. Tests 3–7 are abbreviated excerpts from generated tests for an EnergyHub page; shared setup and helper definitions are omitted. Test 9 shows the generator's button role-and-name branch for exposing a menu link. These examples illustrate test construction; Appendix B.1 specifies the evaluation protocol.

Test 1. Page-loading check

The requested page must pass the response-status check and have a nonempty title and body. The test fixture supplies `pageUrl` from the captured page.

```
const response = await page.goto(pageUrl, {
  waitUntil: 'domcontentloaded'
});
expect(response?.status() ?? 0).toBeLessThan(400);
await expect(page).not.toHaveTitle('');
await expect(page.locator('body')).not.toBeEmpty();
```

Test 2. Evidence: heading text

The test looks for a visible heading with the reference text. The generator obtains `headingText` from a heading that it has verified on the captured page.

```
await expect(
```

Test 2. Evidence: heading text (continued)

```
  page.getByRole('heading', { name: headingText }).first()
).toBeVisible();
```

Test 3. Evidence: link visibility

The *Edge DERMS Platform* link is located by its browser-reported role and accessible name.

```
const link = page.getByRole('link', { name: "Edge DERMS Platform" })
  .filter({ visible: true }).first();
await expect(link).toBeVisible({ timeout: 500 });
```

Test 4. Image: rendered-image fingerprint

The test for the *Alarm.com logo* compares hashes of images in the reconstruction with the stored reference hash `REAL`. The smallest Hamming distance must be at most 8 bits out of 64.

```
const best = hashes.reduce((m, h) => Math.min(m, hamming(REAL, h)),
64);
expect(best).toBeLessThanOrEqual(8);
```

Test 5. Perception: element position

The heading's center must lie inside its captured rectangle, expanded by 128 pixels horizontally and 80 pixels vertically.

```
const { cx, cy } = await absCenter(el);
expect(cx).toBeGreaterThanOrEqual(-68);
expect(cx).toBeLessThanOrEqual(758);
expect(cy).toBeGreaterThanOrEqual(163);
expect(cy).toBeLessThanOrEqual(587);
```

Test 6. Perception: color coverage

A quantized RGB bucket must occupy the permitted fraction of the screenshot. Here, the bucket `240-240-240` must cover 1.88–7.54% of its pixels.

```
const got = __PAL.get("240-240-240") || 0;
expect(got).toBeGreaterThanOrEqual(0.0188);
expect(got).toBeLessThanOrEqual(0.0754);
```

Test 7. Navigation: link navigation

The generated test activates the visible *Edge DERMS Platform* link and checks the landing URL.

```
const link = page.getByRole('link', { name: "Edge DERMS Platform" })
  .filter({ visible: true }).first();
await link.evaluate((el: HTMLElement) => el.click());
const after = landingPage.url().split('#')[0];
expect(after).toContain("/edge-derms-platform/platform-overview");
```

The navigation test also checks that the URL belongs to the captured destination's local route and that its distinctive text is present.

Test 8. Perception: full-page layout bands

The layout generator reduces a full-page screenshot to a 16×12 grid of grayscale averages. Each horizontal band receives its own check, using the mean absolute cell difference normalized by 255. The following generator-derived excerpt uses `band` for the selected row and `REAL` for the stored reference grid; capture setup and the averaging helper are omitted.

Test 8. Perception: full-page layout bands (continued)

```
const grid = gridGray(await page.screenshot({ fullPage: true }), 16,
12);
const similarity = rowSim(grid[band], REAL[band]);
expect(similarity).toBeGreaterThanOrEqual(0.78);
```

The fixed grid normalizes page dimensions, while separate band checks locate differences in the distribution of light and dark regions.

Test 9. Navigation: open a menu and select its link

The recorder supplies `menuName` and `linkName` from the reference website. The following generator-derived example illustrates a button-triggered menu on the reconstructed website. When the record has no explicit trigger selector, the test locates the menu button by role and accessible name. It opens the menu and finds the exposed link by the same semantic matching rule. Shared timing and destination-verification helpers are omitted.

```
const menu = page.getByRole('button', { name: menuName })
  .filter({ visible: true }).first();
await menu.scrollIntoViewIfNeeded().catch(() => {});
await menu.click({ force: true }).catch(() => {});

const link = page.getByRole('link', { name: linkName })
  .filter({ visible: true }).first();
await expect(link).toBeVisible();
await link.evaluate((el: HTMLElement) => el.click());
```

For a hover-opened menu, the opening action is `menu.hover()`. After selecting the link, the navigation test checks the recorded destination and its distinctive visible content.

B Evaluation details

B.1 Final-reconstruction protocol

Each reported score is computed from tests on the final submitted reconstruction for an agent–task pair. Development can include continuations and retries before the final implementation is scored. We record each test outcome and its execution time for that implementation. Unavailable requested pages remain test failures.

Each agent reconstruction is evaluated with four parallel workers and no per-assertion retries. An assertion begins with 1.5 seconds and may extend to a total of 5 seconds when recorded evidence shows that its requested navigation or positively located target operation is still unfinished. A missing target on a loaded page does not qualify for an extension, and neither does an already completed operation or unrelated resource activity. Preparation and cleanup have separate timings. Each assertion receives a fresh browser context and page with an independently initialized deployment of the reconstruction.

B.2 Score calculation

For task s, let G_s contain the reference-applicable families and let $S_{s,g}$ be the percentage of tests passed in family g. Write $b_s = \min_{g \in G_s} S_{s,g}$ for the largest threshold that all its applicable families meet. For N tasks in a track, the pass fraction and its area are

$$R(t) = \frac{1}{N} \sum_{s=1}^{N} \mathbf{1}[b_s \geq t], \qquad \text{AUC} = \int_0^{100} R(t)\, dt = \frac{1}{N} \sum_{s=1}^{N} b_s. \tag{2}$$

The last equality follows because task s contributes to the curve from threshold 0 through b_s. We sum the exact areas under the empirical steps using unrounded task scores, giving every task and threshold equal weight. Figure 4 displays $100R(t)$ as a percentage; AUC includes the full curve even where the displayed vertical range ends at 70%. At $t = 100$, $100R(t)$ equals All-pass. Entry and FirstHop repeat this calculation using each scope's applicable tests on tasks with load checks in

both scopes. Within each scope, test counts are pooled by family before computing the score, and navigation checks belong to the page containing the link.

Table 4 highlights each track's highest displayed score per column, including ties; all-zero columns and Cost are unranked. A dash in the cost column denotes an unreported baseline cost.

B.3 Trajectory length, model turns, and token usage

Table 6 reports the recorded reconstruction effort for each of the six models on each track. We sum the recorded continuations and retries for a task before taking the task mean.

Table 6: Mean recorded reconstruction resources per task. Time is in minutes, trajectory length in tool calls, and input/output/total usage in thousands of tokens (k). A dash denotes unavailable timing.

Model	Time (min)	Length (calls)	Turns	Input incl. cache (k)	Output (k)	Total (k)
Web-50						
GLM-5.3-Flash	173.8	314.8	301.1	31099.7	212.8	31312.5
DeepSeek V4 Flash	92.8	608.9	405.9	52196.5	219.6	52416.1
Claude Opus 5	96.8	299.6	294.9	32873.5	228.7	33102.3
Kimi K3[†]	61.5	266.9	261.9	20411.7	56.3	20467.9
GPT-5.6-sol	14.9	56.1	66.9	4113.6	20.9	4134.5
Claude Sonnet 5	–	205.0	154.1	18034.1	102.6	18136.7
App-50						
GLM-5.3-Flash	39.9	168.6	164.6	20498.7	65.4	20564.1
DeepSeek V4 Flash	15.2	149.9	146.6	14366.0	64.5	14430.5
Claude Opus 5	22.3	95.3	95.4	10856.2	40.2	10896.4
Kimi K3	43.0	140.9	135.5	12163.5	42.3	12205.8
GPT-5.6-sol	8.6	74.4	56.2	3989.8	19.5	4009.4
Claude Sonnet 5	23.3	88.5	90.2	9631.3	47.5	9678.7

[†]Kimi K3 Web token and cost estimates use six runs with complete request accounting. Timing and trajectory measurements use a separate set of recorded runs.

Time and trajectory length. Time is the sum of recorded generation durations, including model computation, tool execution, and waiting within each run. Claude supplies invocation durations; Codex supplies intervals bounded by recorded task start and end or abort events.

Trajectory length counts the deduplicated tool calls in each action sequence, including unsuccessful calls. The granularity of a call depends on the agent's tool interface, which can bundle several operations into one call.

Model turns. A recorded model turn is an assistant response, which can invoke several tools. For Claude-based runs, we count distinct assistant-message identifiers with recorded usage, removing duplicate identifiers across saved files. For Codex-based runs, we count distinct nonzero request-usage updates after deduplicating cumulative counters within each session.

Token accounting and cost. The token columns include recorded cache tokens. Appendix B.4 describes the cost estimates. We partition inclusive input counters into mutually exclusive ordinary-input, cache-read, and cache-write categories before summing them into Input. Adding Output gives Total, which is computed before rounding.

Usage is accumulated across requests and recorded continuations for each task. Complete terminal model-usage totals include recorded internal calls and take precedence over partial message-level totals. Token means use tasks with complete accounting: 41–50 tasks per model–track group, except for the six Kimi Web runs marked above. Timing and trajectory measurements use their respective recorded samples, which can differ from the token samples.

B.4 GENERATION COST

Table 7 reports generation token-cost estimates in USD per task. The estimates use complete token-accounting records as described in Appendix B.3.

Accounting. The estimate includes ordinary input, cache reads, cache writes, and output:

$$C = (p_{\text{in}} I + p_{\text{read}} R + p_{\text{write}} W + p_{\text{out}} O)/10^6.$$

Here I, R, and W are mutually exclusive token categories, and prices are in USD per million tokens. For inclusive input counters, we subtract cache reads and writes before pricing ordinary input. We deduplicate cumulative session counters and include recorded continuations and retries. Complete Claude `modelUsage` totals take precedence over alternative summaries of the same usage. Reasoning tokens are included in output.

Price basis. We estimate token costs from recorded usage and model prices in the gateway catalog saved on September 19, 2026. Prices follow each request's context tier. GLM-5.3-Flash costs exclude cache discounts: ordinary and cached input both cost \$0.07 per million tokens, and output costs \$0.25 per million tokens. DeepSeek V4 Flash costs include cache discounts: ordinary input, cache reads, and output cost \$0.44, \$0.01, and \$1.32 per million tokens, respectively. Other models also use cache-inclusive pricing. These are the saved gateway rates, not current list prices. The estimates are not verified invoices.

The GLM estimates are \$2.23 per Web task and \$1.45 per App task; the DeepSeek estimates are \$0.97 and \$0.27, respectively.

Table 7: Generation token-cost estimates per task in USD. GLM excludes cache discounts; DeepSeek and other models use cache-inclusive pricing.

Configuration	Web	App
GLM-5.3-Flash	2.23	1.45
DeepSeek V4 Flash	0.97	0.27
Claude Opus 5	25.64	7.37
Kimi K3	11.43	5.14
GPT-5.6-sol	4.32	3.17
Claude Sonnet 5	5.69	4.52

B.5 PAGE SCOPE AND REFERENCE SIZE

Both entry and first-hop load checks apply to all 50 Web tasks and 39 App tasks. Ten App tasks have no saved first-hop page; one more has a saved first-hop page without tests. First-hop scores for these 11 tasks are reported as N/A. The paired App entry-minus-first-hop AUC gap changes from -6.7 to -10.1 points when restricted to families applicable in both scopes.

Entry denotes the recorded starting page, which need not be the site's root homepage. Assertions are assigned to the corresponding reference pages, with navigation attributed to its source page and missing reconstructed pages counted as failures. Tasks without first-hop checks remain in the family scores and threshold curves but do not enter the paired comparison. The entry-minus-first-hop gap ranges from -9.6 to -4.1 points across App configurations and from -8.3 to -0.8 on Web. Averaging available configurations within each task, entry scores exceed first-hop scores on 20/50 Web tasks and 16/39 paired App tasks. The applicable test families can differ between the entry and first-hop scopes.

Reference size analysis. Figure 5 compares reconstruction quality across groups defined by captured reference page count. We use each track's page-count quartiles and keep tasks with equal page counts in the same group. For each metric, a task contributes when all six agents have defined scores; we average those scores within each task, then weight tasks equally within each group. Each reconstruction contributes its lowest applicable family score to AUC (Section 2.4). The shaded regions show pointwise 95% intervals from 5,000 task-bootstrap resamples, using seed 20260925.

The App comparison also spans changes in test applicability. Small App tasks often lack image or navigation checks, and the families contributing to AUC therefore vary across the size groups. Among the 25 tasks with all four families applicable, regression of AUC on $\log_2$ page count gives -7.6 points per doubling, with a 95% task-bootstrap interval of $[-14.7, 0.9]$. The interval spans zero, leaving the size association uncertain within this group.

B.6 Five-website guidance study

We examine how page and link guidance affects navigation reconstruction on five public websites (Table 8). The guided condition adds a prompt paragraph listing the URLs of all pages in $\mathcal{P}$ and the actions needed to reveal hidden links. Agents use this guidance to inspect the reference's content and appearance in the browser and build a local reconstruction.

Table 8: Websites in the page and link guidance study.

Website	Purpose
Todoist	Website for a task-management and to-do-list tool.
Biogen	Corporate website of a biopharmaceutical company.
GoTo	Enterprise communications, video meetings, and remote-support products.
EnergyCAP	Enterprise energy-management and carbon-accounting software.
commercetools	Commerce infrastructure for enterprise customers.

Both conditions use the same frozen references and the development and evaluation settings in Section 3.1. The applicable tests and their pass conditions are identical and remain hidden throughout development. Table 5 reports each system's Navigation score averaged equally over the five websites, comparing the standard prompt (Main) with the prompt containing page and link instructions (Guided).

B.7 Recorded behavior and final quality

The behavior analysis links the reconstructions evaluated in Table 4 with their development trajectories. We match trajectories and final reports for the same 50 Web and 49 App tasks across six agents, giving 594 runs. Figure 6(a) divides each run into ten equal bins of tool-call order, then averages action proportions equally across tasks. Dots mark the median first recognized write. GPT's median first write on Web occurs at 36.6% of tool-call progress, compared with 3.3–6.9% for DeepSeek, Opus, GLM, and Kimi. Those four agents continue reference access after writing begins, with median counts of 22.5–58.0; GPT's median is 0.5. On App, inspection of the reconstruction is followed by another write in 43/49 GPT runs and 16/49 Sonnet runs.

We relate reference access and writing patterns to final reconstruction scores. This analysis uses the main evaluation's score definitions, with Perception restricted to layout and position checks. Reference-access frequency divides successful reference-access calls by all calls remaining after the first write. Inspection–write frequency counts sequences of browser inspection followed by a file write per 100 tool calls. Multiple inspections before the next write count as one sequence. First-write timing uses normalized tool-call order. These measures count recorded access and write events, including writes to helper scripts; their granularity depends on the agent's tool interface. Each comparison uses all 50 Web tasks and 31–49 App tasks with applicable scores and action measures for all six agents. App navigation associations retain 33 tasks with applicable navigation checks and complete trajectories.

The 24 exploratory comparisons relate the recorded action measures to final reconstruction scores. We remove additive task and agent means from both variables, then remove their linear association with similarly adjusted $\log(1 + \text{total call count})$. Figure 6(b) plots the residual correlations and pointwise 95% intervals from 5,000 complete-task bootstrap samples, refitting the adjustment in each sample. We assess sensitivity to individual agents by repeating the analysis with each agent omitted in turn.

Later reference-access frequency has correlations between -0.015 and 0.032 on Web and between 0.000 and 0.050 on App, with every interval crossing zero. Web first-write timing has negative

correlations (-0.288 to -0.182), while inspection–write frequency has positive correlations (0.140 to 0.175); App associations are less consistent.

C Evaluator evidence

C.1 Controlled checks of the evaluator

The controlled study measures how introduced defects change family scores and how code rewrites affect individual test outcomes. We apply graded defects to 39 pages across 20 selected tasks. Targets are selected from the reference before scoring. Each modified page is paired with an intact parsed-HTML control under the same test inventory; unaffected pages remain reference copies. The interventions remove nested fractions of text or images, alter layout, redirect captured links, or retain progressively less document content. The paired score changes measure responses to the introduced defects. Figure 3(b) reports mean family-score changes under medium-severity defects relative to the intact control. Blank-page controls cover 50 tasks per track and measure responses to missing content and appearance. These experiments use the same 1.5–5-second timeout rule as the main evaluation (Appendix B.1). Figure 3 reports control scores, defect responses, and rewrite pass-to-fail rates.

The intervention sample spans reference sizes and interface types. The rewrite screen requires unchanged normalized text and a screenshot difference below 0.08. Rewrite rates are averaged across tasks, with 95% confidence intervals obtained by resampling tasks. Identifier/wrapper edits rename selected identifiers and add neutral wrappers. The Flex-to-Grid probe changes seven containers on four pages. Image-to-background rewrites expose a dependence on image-element representation.

C.2 Human evaluation

Agreement with the automated ranking is positive for all three participants. Three PhD participants each ranked six reconstructions on the same 40 tasks (20 Web and 20 App), giving 120 task-level rankings. The tasks were selected by a random split. Table 9 summarizes agreement between automated and expert rankings and between experts.

Table 9: Agreement between automated and expert rankings and between experts.

	Human–machine		Human–human	
Measure	Agreement	Tasks	Agreement	Tasks
Kendall's τ_b	0.705	40	0.710	40
ICC(A,1)	0.832	40	0.825	40
Weighted κ	0.640	40	0.650	40

Human–machine Kendall's τ_b, ICC and weighted κ retain all 40 tasks.

Ranking and aggregation. The calculation retains score ties and gives each task equal weight. The automated rankings use AUC: each reconstruction's score is the minimum of its applicable family scores (Appendix B.2). Each task's six reconstructions are ranked using their full-precision scores. Exact score ties receive competition ranks.

Mean Kendall's τ_b summarizes agreement across tasks after averaging the participant comparisons within each task. For human–machine agreement, we average the three participant–machine coefficients per task; for human–human agreement, we average the three participant-pair coefficients. We then average the task means over the applicable tasks listed in Table 9.

ICC and weighted kappa. The additional coefficients quantify agreement on numerical rank positions using the same rank vectors for the six reconstructions. We calculate the two-way mixed-effects, absolute-agreement, single-rating ICC(A,1) (McGraw & Wong, 1996), which treats numerical ranks as equally spaced. Linearly weighted Cohen's κ (Cohen, 1968) uses fixed rank categories $1, \ldots, 6$ with agreement weights $w_{ij} = 1 - |i - j|/5$. For each measure, we average the three pair coefficients within each task, then the 40 task means equally, giving 120 pair–task comparisons for each agreement type.

The ranking interface lets participants inspect the reference and six reconstructions before assigning ranks, with ties allowed (Figure 8).

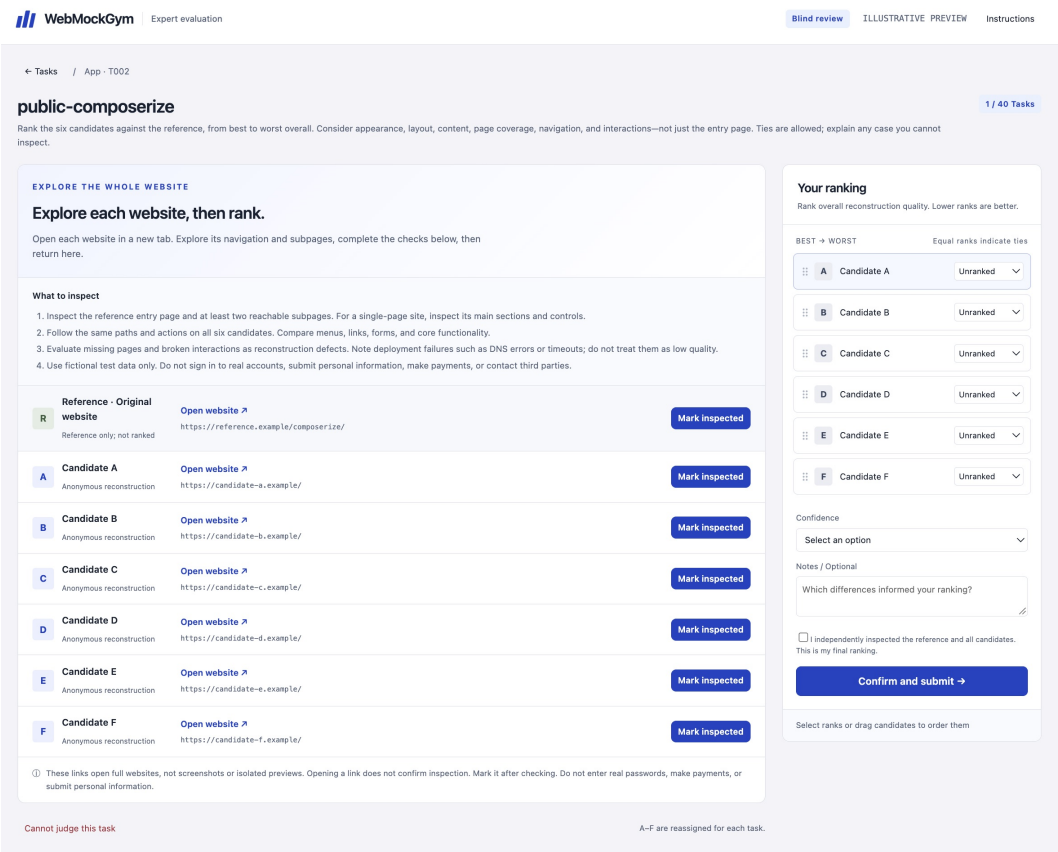


Figure 8: Illustrative English-language view of the PhD expert-ranking interface, with placeholder links. Reviewers inspect the reference and six anonymous reconstructions, then assign ranks with ties allowed. The panel also records confidence, optional notes, and confirmation of independent review.

D TASK CATEGORIES AND THEIR SUBTYPES

Every task asks an agent to reconstruct a captured website or web app. The categories below describe the kinds of source interfaces that must be reconstructed. Tables 10 and 11 expand all eight categories in the dataset overview into their constituent types. Each task belongs to one category within its track.

D.1 WEB-50: WEBSITE SURFACES

Web categories describe how a public website presents its content. *Product platforms* present business products and technical platforms; *Consumer services* present services or transactions for individual customers. *Workflow tools* are public sites for products used to carry out a workflow, while *Brands & catalogs* present a company, brand, or product catalog. Table 10 lists the recorded domain types within each category. A domain can occur in more than one category because domain and website surface describe different properties.

Table 10: Web-50 categories and all domain subtypes represented within them. Counts in parentheses partition each category's task count; each website is counted once.

Category	Tasks	Included subtypes (number of tasks)
Product platforms	22	Agricultural technology (1); Automotive and mobility (1); Commerce and marketing (2); Communications platforms (2); Developer products (4); Energy and utilities (2); Business and financial technology (1); Industrial and manufacturing technology (2); Insurance (1); Property and field-service software (1); Security and privacy (2); Travel and hospitality (2); Workforce software (1).
Consumer services	11	Consumer health (2); General consumer services (1); Energy and utilities (1); Business and financial technology (1); Health education (1); Healthcare services (2); Insurance (2); Travel and hospitality (1).
Workflow tools	9	Design and creative tools (2); Developer products (2); Productivity and collaboration (4); Visual-media tools (1).
Brands & catalogs	8	Agricultural technology (1); Automotive and mobility (3); Consumer retail (1); Life sciences (1); Consumer telecommunications (1); Travel and hospitality (1).
Total	50	

D.2 APP-50: PRIMARY INTERACTIONS

App categories describe the source application's primary interaction. *Data transformation* maps supplied inputs or parameters to computed, converted, or generated output. *Forms & workflows* organize choices, validation, submissions, and state changes through controls. *Content editing* lets users author or directly manipulate an artifact, while *Browse & explore* lets them inspect existing content and collections. Table 11 spells out the recorded interaction types within each group. Editing followed by a transformed preview occurs in both data-transformation tools and a content editor; the latter is assigned to Content editing using its recorded product family. The App track evaluates how faithfully agents reproduce application interfaces and the navigation paths through which users access their content.

Table 11: App-50 categories and all interaction subtypes represented within them. The descriptions expand the recorded interaction labels; counts in parentheses sum to the category total.

Category	Tasks	Included subtypes (number of tasks)
Data transformation	16	Author input and convert it to output (1); Configure, generate, and copy output (1); Edit input and preview its transformation (2); Interpret an expression as it changes (1); Configure and generate output (2); Calculate a result from supplied input (1); Convert supplied input to another representation (1); Format input as it is entered (1); Generate and preview output from input (1); Adjust controls and update calculations (2); Change parameters and recompute results (2); Configure and run a query or calculation (1).
Forms & workflows	14	Add, toggle, and delete task items (1); Select and update a date range (1); Select and update a date (1); Fill a form and inspect submission validation (2); Validate and submit forms (2); Complete and submit a guided form (1); Select multiple items and submit a bulk update (1); Open, confirm, and dismiss dialogs (1); Open a record, edit it, and save or cancel (1); Search and select an option (1); Select options and submit a checkout form (1); Start, advance, and dismiss a guided tour (1).
Content editing	10	Add, move, and resize layout elements (1); Author, edit, and transform structured content (2); Author content and edit its visualization (1); Author and format rich text (1); Author content, preview it, and export it (1); Compose citations (1); Edit input and preview its transformation (1); Crop and transform images (1); Edit and navigate a timeline (1).
Browse & explore	10	Search and navigate articles (1); Navigate a calendar (1); Inspect a dashboard and drill into details (1); Filter, sort, and navigate collections (2); Open, navigate, and close a media gallery (1); Change parameters and update a visualization (1); Browse a storefront (2); Explore a visualization (1).
Total	50	

E EXAMPLES OF RECONSTRUCTED PAGES

We show the entry page of the GEICO insurance website. The figure places the saved reference on the left and six agents' final reconstructions in two columns on the right. All screenshots use a 1280×800 viewport and a five-second wait after the initial document loads, without manual scrolling or interaction. The example illustrates differences in appearance and visible content; navigation is evaluated separately by the benchmark tests.

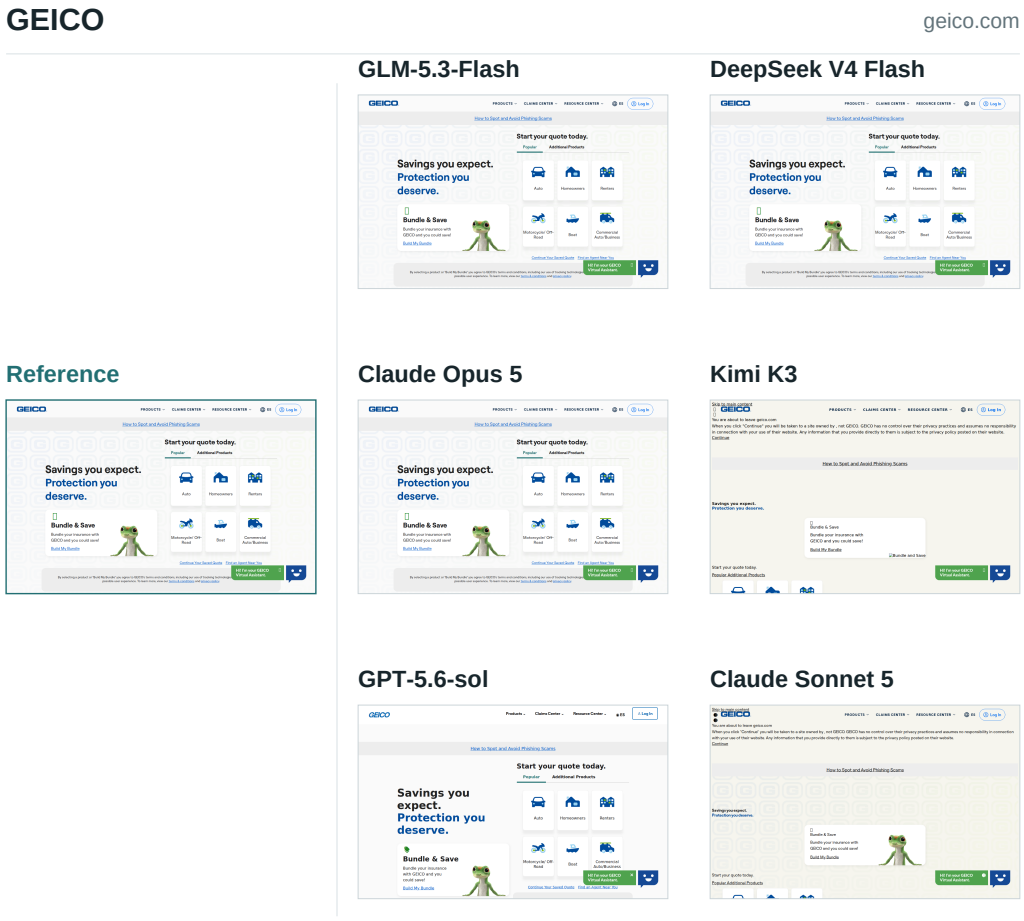


Figure 9: GEICO. Compare the two-column layout, product cards, and text placement. The reconstructions differ in spacing and in how much of the page fits within the same viewport.