MOCKWEB-BENCH: EVALUATING HYBRID AGENTS THROUGH WEBSITE RECONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Reconstructing a complete website from a running reference requires discovering its appearance and behavior across pages, interaction states, and screen sizes, then reproducing both in an independent implementation. Unlike fixed screenshots or recorded interactions, a running reference lets agents decide what to inspect as they build. We introduce MockWeb-Bench, which evaluates hybrid graphical user interface (GUI) and coding agents on 50 offline reference websites with 4,101 manually verified evaluation pages and paired desktop and mobile captures. Each task provides an entry URL, a starter workspace, and a six-hour initial budget; the list of scored pages stays hidden. Functional tests are validated on the reference and frozen before candidate generation, then applied to each reconstruction to measure behavioral fidelity. A fixed vision-language model (VLM) checks candidate screenshots against visual requirements extracted from reference screenshots, while separate checks assess quality and structure. Across ten model–agent configurations, GPT-6 Astra with Codex achieves the highest mean final score of 56.26%. All ten model–agent systems score lower on interaction tests than on general content and navigation tests. Beyond final scores, trajectory case studies reveal both useful strategies and attempts to take shortcuts. Agents write code to compute pixel differences between reference and candidate screenshots, using the results to make reconstructed pages look much closer to the reference. Some systems also try to read protected files or copy reference code, showing why evaluations need to restrict access to reference files and check submissions for copying. MockWeb-Bench supports studying both the websites hybrid agents build and the strategies they use to build them. Our code and dataset are available at `https://anonymous.4open.science/r/mockweb-bench-208C/README.md`.

1 INTRODUCTION

Real-world software work often spans code changes, program execution, and interaction with user-facing applications (Xie et al., 2024). Command-line interfaces (CLIs) and application APIs give agents structured access to files and supported operations, but their coverage depends on what an application exposes. Visual layouts, control states, and some user-facing actions may have no corresponding command or application API. Graphical user interfaces (GUIs) provide access to these views and controls. Relying entirely on GUI controls, however, can add navigation and input steps to code changes, file operations, and builds that direct editing and execution tools can perform programmatically.

Hybrid agents can improve adaptability and efficiency in everyday office work by combining GUI interaction with programmatic tools. Routine office tasks alone, however, do not fully test the sustained coordination needed for long and complex development. Software reconstruction therefore offers a demanding setting for evaluating hybrid agents: they must use GUI observations to guide coding and repeatedly check the rebuilt application against the original. We focus on complete website reconstruction across pages and interaction states, evaluating both appearance and supported behavior. The reconstruction task differs from addressing a specified software issue in SWE-bench Live (Zhang et al., 2025) or completing goals within existing applications in WebArena and OS-World (Zhou et al., 2024; Xie et al., 2024).

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.

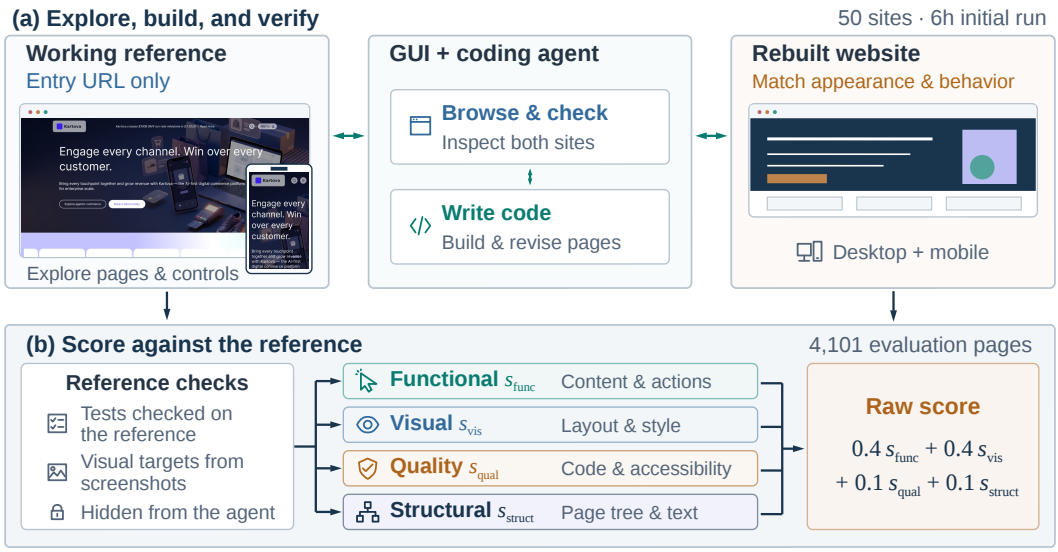


Figure 1: MockWeb-Bench workflow. (a) Agents use GUI and coding tools to explore the reference and iteratively build and check a candidate website. (b) Evaluation scores the candidate against fixed reference requirements. Reference images are from Kartova; the candidate preview is schematic.

A running reference website exposes requirements through its pages, responsive layouts, and responses to user actions. Prior reconstruction benchmarks supply screenshots, resource lists, or recorded interactions (Si et al., 2025; Wan et al., 2024; Xiao et al., 2025; Chen et al., 2025), selecting the available evidence before implementation begins. In contrast, an executable reference lets agents gather additional evidence as they implement the site, while also serving as a common target for evaluating the delivered website.

We therefore introduce MockWeb-Bench to evaluate hybrid GUI and coding agents through complete website reconstruction (Figure 1). The benchmark contains 50 offline reference websites and 4,101 manually verified evaluation pages with desktop and mobile captures. Each task supplies an entry URL, a starter workspace, and a six-hour initial budget, while withholding the scored-page inventory. Agents choose how to distribute their effort across the site's pages and interaction states. Functional tests are constructed from reference content and interactions, validated by execution on the reference, and frozen before candidate generation. The same tests are then applied to each reconstruction. A fixed vision-language model (VLM) judges appearance against requirements drawn from reference screenshots; additional checks measure page quality and structure.

Using this protocol, we evaluate ten model–agent configurations, with GPT-6 Astra with Codex achieving the highest mean final score of 56.26%. All ten model–agent systems score lower on interaction tests than on general tests of page content and navigation, highlighting the challenge of turning interface evidence into reliable implementations. At the process level, trajectory profiles reveal distinct mixes of GUI and coding work among different systems, including renewed interface use after a concentrated writing phase.

Contributions. Our contributions include the following: (i) A benchmark for hybrid agents that use GUI access to discover a website's appearance and behavior and coding tools to reconstruct it, with 50 executable references and desktop and mobile targets. (ii) An evaluation protocol that makes website reconstruction verifiable against an executable reference, using validated functional tests, fixed visual requirements, and quality and structural checks. (iii) An empirical study of ten model–agent configurations that identifies a consistent interaction-fidelity gap and characterizes how systems allocate work across GUI exploration, coding, and subsequent refinement.

2 Related Work

Website generation and reconstruction. Existing work gives agents different descriptions of the website to build. Screenshot-to-code work uses images as reconstruction targets (Si et al., 2025;

Laurençon et al., 2024; Xiao et al., 2026), supported by paired webpage and code data (Yun et al., 2024; Gui et al., 2025). Other benchmarks add resource lists specifying images and link destinations (Wan et al., 2024), or recorded states and videos showing interactions (Xiao et al., 2025; Chen et al., 2025). Text-based benchmarks describe application requirements (Lu et al., 2025b; Tran et al., 2026), while VISTA, Vision2Web, and WebCompass support multiple input formats (Guo et al., 2026; He et al., 2026; Lei et al., 2026). Vibe Code Bench accepts different designs that meet the requested behavior, while reconstruction targets the reference's appearance and behavior. MockWeb-Bench makes the working reference available throughout development, so agents choose which pages and controls to inspect as they build and check their implementation. Table 1 compares benchmark inputs and evaluation coverage.

Table 1: Benchmark inputs and evaluation. N/A: no reference views; –: no dedicated score. User workflows are multi-step interactions; † marks image-conditioned visual scoring.

	Target specification		Evaluation	
Benchmark	Given input	Observations	Visual match	Functionality tested
Design2Code	Screenshot	Fixed	✓	–
MRWeb	Screenshots + resources	Fixed	✓	Links and resources
Interaction2Code	Before/after screenshots	Fixed	✓	UI interactions
IWR-Bench	Interaction video + assets	Fixed	✓	User workflows
WebGen-Bench	Text specification	N/A	–	UI interactions
Vibe Code Bench	Text specification	N/A	–	User workflows
VISTA	Text + optional screenshots/Figma	Fixed / N/A	✓	UI interactions
Vision2Web	Prototypes + specs + assets	Fixed	✓	User workflows
WebCompass (gen.)	Text / screenshots / video	Fixed / N/A	✓†	User workflows
MockWeb-Bench	Executable reference URL	**Agent-selected**	✓	Page content, navigation, interactions

GUI agents and interactive environments. GUI-agent research studies how agents understand and use interfaces. Benchmarks measure task completion in web (Zhou et al., 2024; Koh et al., 2024; Drouin et al., 2024) and desktop applications (Xie et al., 2024). Mind2Web provides demonstrations across real websites (Deng et al., 2023), while WebCanvas tracks intermediate actions and states during online evaluation (Pan et al., 2024). WARC-Bench and WebForge address reproducible web environments (Srivastava et al., 2026; Yuan et al., 2026). Qwen-CUA trains agents to use computers through screenshots, keyboard and mouse actions, and rewards based on verifiable outcomes (Lu et al., 2026). Spider2-V combines GUI control and coding for professional data workflows (Cao et al., 2024). In MockWeb-Bench, interface interaction supplies information for a coding task. Agents use the reference to infer requirements and the candidate to inspect their implementation, while programming tools support code changes and execution.

Website evaluation and copying detection. Assessing a generated website requires checking both its appearance and what users can do. FrontendBench uses functional tests (Zhu et al., 2025), while IWR-Bench combines browser interactions with visual judgments (Chen et al., 2025). Agentic testing uses code and browser exploration to find bugs (Hong et al., 2026). Feedback on appearance and interactions also guides model training in ReLook, WebGrader, and WebGen-Agent (Li et al., 2025; Chen et al., 2026; Lu et al., 2025a). Work on benchmark reliability and reward hacking studies cases where high automated scores do not reflect the intended task (Yu et al., 2026; Gabor et al., 2026). MockWeb-Bench pairs appearance and behavior scores with checks for copied reference code.

3 MOCKWEB-BENCH

3.1 TASK SETUP

MockWeb-Bench asks an agent to reconstruct the appearance and supported behavior of a complete website. The locally served site it can explore is the reference S; the agent's new implementation is the candidate $\hat{S}$. The original site's server-side code, database operations, and live services such as login verification are outside the task. Each task is $\mathcal{T} = (S, u_0, W, \tau)$, where u_0 is the starting URL on the reference, W is a writable project directory initialized with an application template, and $\tau = 6$ hours is the initial time budget (Appendix D.1). Starting at u_0, the agent discovers pages and interactions as it develops $\hat{S}$ in W. The page list used for scoring, the reference project files, and the prepared evaluation tests and screenshots are withheld from the agent.

Browser tools let the agent inspect and operate both sites. Playwright automates browser operations, and the Model Context Protocol (MCP) exposes them as tools the agent can call. These tools provide screenshots and text descriptions of page elements, and support clicking, scrolling, and typing. On the reference, opening an expandable section reveals content to implement; on the candidate, the same action checks whether that content appears. Command-line and editor tools let the agent install the libraries used by the project, edit code, build the site, and run it locally for testing. The agent can revisit either site while developing the candidate.

The template uses React for page components, TypeScript for application code, and Tailwind CSS for styling; Vite builds the deliverable. The agent submits source code and `output/index.html`, which embeds the application logic (JavaScript) and styles (CSS) in a single HTML page. This entry point uses the reference's URL paths to display the corresponding reconstructed pages; each subpage must work when opened directly. The required desktop and mobile viewports (the browser's visible page area) are 1920×1080 and 375×812 CSS pixels, respectively. Agents may reuse local media such as images and fonts, visible text, and individual design values such as colors or font sizes, under the task rules and applicable licenses. They must write their own HTML structure and CSS styles, including the class names that associate styles with page elements. Appendix G.1 provides the task and workspace instructions.

3.2 DATASET CONSTRUCTION

We adapt captured public websites into offline references, preserving their responsive layouts and organization across pages. Six are modified copies that retain permitted source material and attribution while replacing selected branding and media; the other 44 are synthetic variants of distinct source sites with fictional identities. Models support these adaptations by rewriting text and generating replacement images. To make the resulting references run locally, we use scripts to package pages and media and rewrite internal links, with additional repairs to resolve offline rendering issues.

Reviewers inspect reference pages for rendering problems. For synthetic variants, they also check for remaining real names, logos, and protected imagery (Appendix A.1.1). For each reference S, the reviewed page list defines $P^* \subseteq \text{pages}(S)$, the pages on which its reconstructions are scored. Across the 50 references, this gives 4,101 evaluation pages, with 20–158 per site (median 79). We store a desktop and a mobile screenshot of each selected page as visual targets, giving 8,202 screenshots (capture settings in Appendix A.1). Figure 2 summarizes the site categories and shows desktop/mobile examples; Appendix A provides the full site inventory and construction details.

These sites, page lists, and screenshots are released with the evaluation checks described in §3.3, plus license notices for source material, benchmark modifications, and bundled components.

3.3 TEST CONSTRUCTION

We construct evaluation checks for four dimensions: Functional, Visual, Quality, and Structural. Table 2 shows the reference information and representative checks for each dimension.

Functional checks. Our scripts generate Playwright checks from reference-page titles, headings, and links. During dataset preparation, a separate test-generation agent explores each reference and adds two groups of tests: general functional checks for content and navigation, and dedicated interaction checks for responses to user actions. The scripted checks and both agent-assisted groups

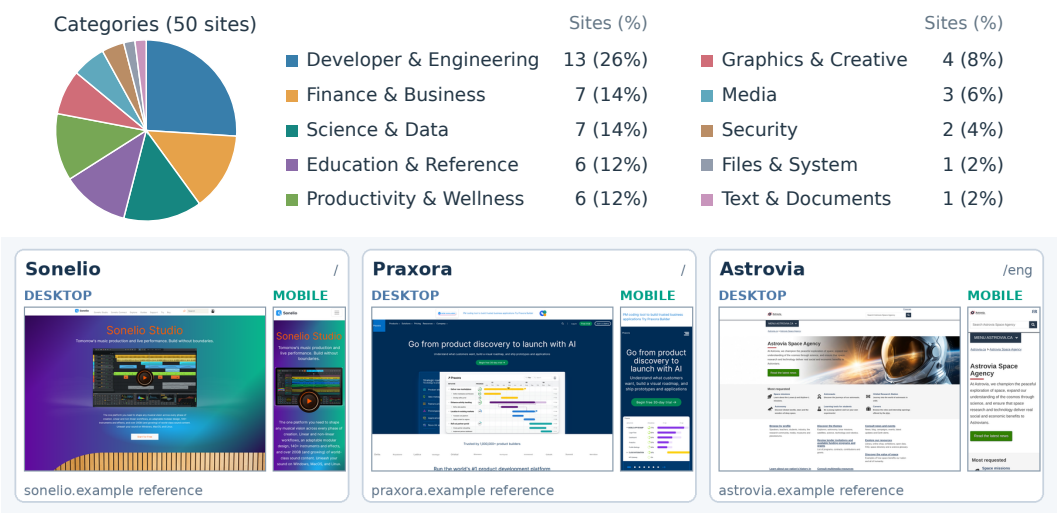


Figure 2: Site categories (top) and desktop/mobile crops from three synthetic references (bottom).

Table 2: Reference information and representative checks for the four evaluation dimensions.

Dimension	Reference information	Representative checks
Functional	Rendered text and links; browser actions and observed outcomes	Page loading, titles and headings, and navigation; dropdowns, tabs, expandable sections, dialogs, mobile menus, and outcomes after actions.
Visual	Desktop and mobile reference screenshots	Element placement, section colors, text size and style, and reference imagery.
Quality	Rendered pages and HTML properties	Console errors, image loading, text presence, alternative text for images, and document language.
Structural	Page structure and desktop/mobile layouts	Header and navigation regions; heading order; content fitting the mobile viewport width.

contribute to the Functional score defined in §4. The test-generation agent receives the evaluation page list and metadata describing page titles, URL paths, and links between pages; these evaluation inputs are withheld from the reconstruction agent.

Each interaction test specifies a starting page, browser actions, and an expected change in content or control state supported by the offline reference. To support independent implementations, we instruct the test-generation agent to favor visible text, roles such as `button` or `link`, and accessible labels, and to avoid CSS class selectors tied to the reference. These selectors identify controls without requiring the same class names or element nesting. For example, a test on the Vellora reference at the 375×812 mobile viewport checks that the "Careers" menu link is hidden, clicks the menu button, and requires "Careers" and "Integrations" to be visible. Other cases check revealed content, whether a control is marked as open in the HTML, and values entered into search fields. Navigation arrival checks require content specific to the destination page to become newly visible after the action; a successful click alone does not satisfy the check.

To validate the generated checks, we first run the full functional suite, including the dedicated interaction tests, on the locally served reference website. After repairing or removing failing checks and discarding duplicates or checks for invalid pages, we freeze the validated suite before reconstruction begins. We then replay the same tests on each candidate website, keeping the starting pages, browser actions, and expected outcomes fixed. Appendix A.3 documents this construction and validation procedure and illustrates a test with two successive navigation steps. To further assess reliability on independently built websites, we also check whether the tests detect deliberately introduced navigation failures and audit a stratified sample of candidate test results (Appendix D.4).

Visual checks. To specify how reconstructed pages should look, a vision-language model (VLM) reads desktop and mobile reference screenshots and generates visual assertions. Each describes one

aspect of placement, color, text style, or imagery—for example, that a page background is near-black. Visible text identifies the element being checked; image descriptions follow the prepared reference, including its replacement artwork. To focus on page-specific content, the prompt allows at most two assertions per capture for shared headers, navigation, search bars, and footers combined. We remove assertions repeated across many pages and freeze the lists before reconstruction. During evaluation, a VLM judge checks candidate screenshots against these fixed lists (§4). Appendices A.3 and G detail the generation constraints, filtering, and prompts.

Quality checks. Scripts generate checks for basic rendering and accessibility properties, including image loading, console errors, and text alternatives for images.

Structural checks. The browser represents HTML as a document object model (DOM), a tree of page elements. Our scripts use the reference's DOM and rendered pages to generate checks for page organization, such as header and navigation regions (Table 2).

3.4 Runtime isolation

To keep reconstruction based on the reference interface, we control which files an agent can read and which services it can contact during each run. The agent can browse the local reference while the reference's project files and evaluation materials remain protected on disk.

Filesystem and information isolation. The agent runs under a user account that can write to its development workspace. File permissions prevent that account from reading the reference project, evaluation tests, prepared reference screenshots, visual assertions, and scoring code. The reference server runs separately and retains read access to the reference files so it can serve the site over local HTTP. The runner, the program that starts and coordinates each task, also removes the scoring page list and related hints, such as page-count limits, from the agent's instructions and workspace files.

Network isolation. Agent processes run in a network namespace, an isolated network environment with no direct route to external services. Local relays forward requests across this boundary to the reference server and model services. Direct access to the original public websites and other external sources of implementation code is blocked. The runner stops if it cannot establish the requested isolation. Appendix F.4 documents an earlier run with external access that downloaded and reused original article source despite protected reference files.

Because the browser receives HTML and CSS to render reference pages, we also audit submitted code for copied reference content (§4). Appendix C details the runtime controls and copying audits. Appendices E.4 and F.3 document attempted access to protected files and copying of reference code.

4 Evaluation Protocol

The reference supplies the user-facing requirements for reconstruction and the target for evaluation. We evaluate each candidate on P^* using the fixed evaluation checks from §3.3. Functional and visual scores measure how well the candidate preserves the reference's behavior and appearance; quality and structural scores measure document and code properties. Each component ranges from 0 to 1. We combine them into a Raw score before applying the copying penalty:

$$\text{Raw} = 0.40\, s_{\text{func}} + 0.40\, s_{\text{vis}} + 0.10\, s_{\text{qual}} + 0.10\, s_{\text{struct}}. \tag{1}$$

We report all four components alongside Raw and Final, the score after any copying penalty. The functional score s_{func} is the weighted pass rate of functional tests on the candidate. Content and interaction checks receive greater weight than basic loading and link checks.

Reference and candidate screenshots use the same capture procedure (Appendix A.1). To score appearance, a fixed VLM receives the candidate screenshot and the assertion list created from the reference. Every system is thus checked against the same requirements for each page and viewport. The judge returns pass or fail for each requirement's appearance and location. The viewport score is the weighted assertion pass rate; all released assertions have unit weight. Page scores combine desktop and mobile results with weights 0.60 and 0.40; s_{vis} averages the resulting page scores.

Quality checks cover form labels, image alternative text, semantic markup, and contrast in sampled text. Continuous measurements give partial credit for these properties, while executable tests check conditions such as image loading and console errors. The structural score compares the reference and candidate DOMs after normalization, measuring their tree structure, text, and page regions such as headers and navigation. Structural tests check properties such as heading order and horizontal overflow on mobile. We combine continuous measurements with test pass rates at $0.60 : 0.40$ for s_{qual} and $0.70 : 0.30$ for s_{struct}. Appendix B.4 specifies the measurements and aggregation.

Copying audits and score enforcement. The copying cap enforces the requirement for an independently authored implementation. Transferring reference HTML/CSS can yield high functional and visual scores while bypassing this requirement. The task prompt states the 10% cap before reconstruction begins (Appendix G.1). We audit submitted source and build files for evidence of copied reference code, including large matching HTML blocks. Each evaluation includes the audit evidence and verdict. A copying verdict (`SCRAPE`) caps that candidate's Final score at 10%:

$$\text{Final} = \begin{cases} \min(\text{Raw}, 0.10), & \text{if the copying audit returns } \texttt{SCRAPE}, \\ \text{Raw}, & \text{otherwise.} \end{cases} \tag{2}$$

Other detectors record information for analysis without changing the score. Appendix C describes the audit rules.

5 Experiments

Our experiments measure how faithfully agents turn a running reference into a new implementation and how much elapsed time this requires. We then examine how they distribute work between interface observation, GUI interaction, and code development.

5.1 Experimental setup

We evaluate ten model–agent combinations from the GPT, Claude, Gemini, GLM, Qwen, and Grok families on the 50 websites (Table 3). The two GPT configurations use Codex; the remaining configurations use Claude Code. All systems share the task prompt, React workspace, and Playwright browser backend. Image reading is enabled except for GLM-5.3, which uses textual browser snapshots and page-inspection tools. Each system consists of its model, agent software, and observation settings.

Runs follow the task setup in §3.1. During the initial six-hour run, agents explore the reference, write code, and test the candidate. They can revise their implementations using browser observations, screenshots, and build feedback.

We score the delivered website on the four dimensions in §4, with Qwen3.7-Plus as the visual judge for all candidates. Evaluation results are not returned to the agent for further revision. Appendix D.1 gives inference settings, time limits, and the conditions for resuming agents that return without source changes. Appendix G provides the prompt templates.

5.2 Reconstruction performance and efficiency

Table 3 compares the ten systems, ordered by mean Final score. GPT-6 Astra has the highest mean Final score at 56.26%, followed by GLM-5.3 at 53.27%. Across 4 alternative weight settings, rank correlations with the standard score are $\rho = 0.988$ (Appendix D.8). The Raw–Final gap is largest for Gemini 3.7 Flash, which is capped on 25 of 50 websites. We therefore reviewed capped and uncapped submissions to check for false-positive penalties and missed copying (Appendix C.3).

The component scores separate basic implementation quality from reconstruction fidelity. Systems score highly on quality checks, while their functional and visual scores are lower. Following common document and code conventions does not by itself reproduce the reference's content, appearance, and interactions.

Table 3: MockWeb-Bench results (%). Final includes copying penalties; Capped counts penalized sites.

System	Final	Raw	Func.	Visual	Qual.	Struct.	Capped
GPT-6 Astra	56.26	58.12	47.18	59.02	92.12	64.31	2
GLM-5.3	53.27	57.10	51.30	51.40	93.81	66.33	3
Claude Opus 5	52.75	56.11	49.02	52.51	90.54	64.39	3
Grok-4.6	46.27	46.27	45.15	33.52	90.67	57.32	0
GPT-5.6 Sol	44.27	45.23	30.68	45.53	89.77	57.74	1
Qwen3.8-27B	43.96	44.59	35.22	39.15	91.13	57.23	1
Qwen3.8-Max	40.15	40.15	30.86	33.17	89.63	55.72	0
Claude Opus 4.8	33.89	34.37	21.73	27.33	91.47	55.96	1
Qwen3.7-Plus	21.69	21.69	10.00	10.38	85.55	49.88	0
Gemini 3.7 Flash	17.35	36.00	30.30	23.72	84.89	59.01	25

Table 4 compares the weighted pass rates of the two agent-assisted functional suites. Dedicated interaction tests score below general content and navigation tests in 10 of 10 systems. Appendix D.3 reports per-system test-group scores. The same interaction can be implemented with different HTML structures. To check whether this affects the comparison, we repeat it on retained runs using a subset of interaction tests with semantic role or text locators, excluding CSS selectors and assertions about DOM classes, attributes, or styles. Interaction scores remain lower for all ten systems under this restriction (Appendix D.4). Additional analyses examine site scale (Appendix D.6) and the effect of providing evaluated page URLs, which reveal which pages to build and give clues about their content (Appendix D.7).

Table 4: Agent-assisted test-group scores (%). Equal-weight system means.

Test group	Mean score
General functional	51.57
Dedicated interaction	22.29
Gap (pp)	29.28

Reconstruction time. We measure elapsed time from the start of the agent stage until the runner takes over the final build, including time spent waiting for models and tools. Following Tran et al. (2026), Figure 3 pairs each system's median time with its mean Final score in Table 3. The dashed line connects systems for which no other tested system has both a shorter median time and a higher mean score. Qwen3.8-27B and GPT-5.6 Sol score 43.96% and 44.27% at median times of 113.4 and 396.9 minutes. GPT-6 Astra reaches the highest score, 56.26%, at 136.1 minutes. Appendix D.5 gives the measurement details.

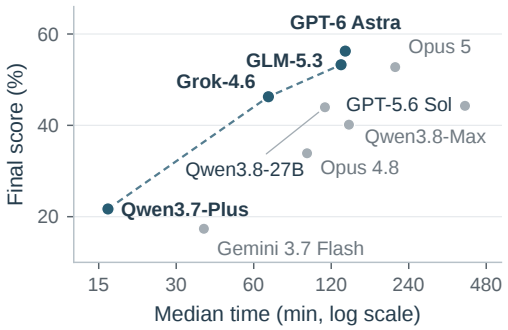


Figure 3: Time and reconstruction score.

5.3 Analysis

Table 4 shows that all ten systems score lower on interaction tests than on content and navigation tests. These aggregate results do not show which behaviors remain missing or how agents develop and check their reconstructions. We examine patterns of GUI and coding activity, interaction failures in visually faithful outputs, and the feedback agents use during refinement. Measurement details and additional trajectory analyses appear in Appendix E.1.

Systems combine GUI and coding work differently. Astra uses 52.8% of its actions to observe interfaces and 13.8% to interact with them (Figure 4a). Reading and writing together account for 53.2% of Opus 5's actions and 46.4% of GLM-5.3's, compared with 15.1% for Astra. These activity profiles describe how agents work; the output scores show whether their implementations reproduce the reference's behavior. Among outputs with recorded interaction scores, 35 achieve Visual scores of at least 80% without a copying penalty, yet 16 (45.7%) score below 50% on dedicated interaction tests (Appendix E.3). The interaction gap therefore persists within visually strong reconstructions.

GPT-5.6 Sol's Corravale candidate illustrates what can remain missing: it scores 87.69% on Visual and 8.20% on dedicated interaction tests. Source inspection finds a privacy notice without expansion logic or content and an unnamed search input whose form has no submit handler. A local browser replay confirms that clicking the notice fails to reveal its content and submitting the search drops the entered keyword from the URL (Appendix F.1). These omissions concern the behavior behind visible controls: the state changes, revealed content, and submitted values that a reconstruction must preserve. By checking the outcomes of actions against an executable reference, MockWeb-Bench exposes behavioral gaps that a high visual score alone does not rule out.

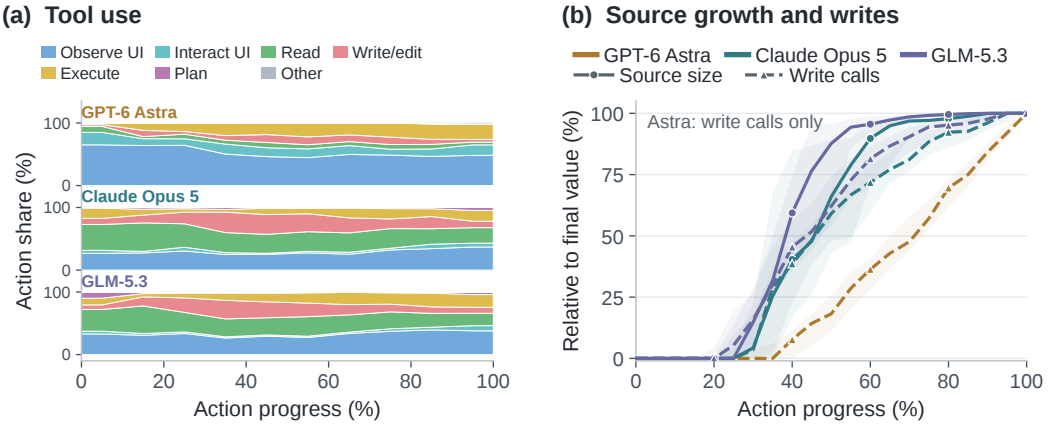


Figure 4: Tool use and implementation dynamics. (a) Mean per-run action shares across ten progress bins. (b) Median source size and cumulative source-write calls, normalized by each run's final values. Shading: 95% site-bootstrap intervals. Appendix E.1 gives definitions and sample counts.

Development shifts toward inspection and revision. For Opus 5 and GLM-5.3, the share of writing actions peaks at 30–40% progress, then declines as browser activity increases (Figure 4a). Source growth slows while writing continues (Figure 4b). Across eight Claude Code configurations with recoverable edit histories, a median of 26.4% of source-write calls occur after source size first reaches 90% of its final value. Together, the panels suggest a shift from code expansion toward inspection and revision.

Agents write code to guide visual refinement. GPT-6 Astra writes Python scripts that compute mean absolute error (MAE) between the RGB pixel values of reference and candidate screenshots. For Vellora, it uses screenshot comparisons and measurements of typography and spacing to revise responsive layouts and mobile controls. The mean recorded MAE across eight mobile routes falls from 22.40 to 8.09 after this refinement round. In a separate Aperlio run, MAE on the same Photos page decreases from 44.67 to 11.98 to 5.39 across successive checks. These cases illustrate how coding turns GUI observations into quantitative feedback for visual refinement (Appendix F.2).

6 Limitations and Broader Impact

Long runtimes and high costs limit evaluation to one run per system–site pair, leaving run-to-run variability unmeasured. Future work could enhance reconstruction and verification on backend logic, databases, and APIs. Reconstruction could support interface migration but also facilitate unauthorized cloning and phishing. Offline execution and fictionalized references reduce exposure to real websites and identities during evaluation, but cannot prevent downstream misuse.

7 Conclusion

MockWeb-Bench evaluates hybrid GUI and coding agents through complete website front-end reconstruction. Agents discover requirements through browser observation and interaction, then implement and refine them with programming tools. Evaluation checks behavior and appearance against the reference, alongside quality and structure. Across ten model–agent combinations, interaction scores remain below content and navigation scores. Trajectory analysis reveals different tool allocations and a shift toward inspection and revision; case studies show agents using pixel differences for visual refinement. The benchmark supports research on agents that discover, implement, and verify requirements throughout development.

AI USE STATEMENT

Generative AI assisted with synthesizing reference websites, generating tests and visual assertions, and evaluating visual appearance, as described in Sections 3.2, 3.3, and 4. It also assisted with manuscript polishing. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Under the protected protocol, reconstruction takes place on local reference and candidate sites, without actions on live public websites or their users' accounts. Synthetic references use fictional identities and undergo manual screening for recognizable real branding, personal likenesses, and protected visual material (Appendix A.1.1). Per-site license notices record modifications, attribution, and component-specific terms. These measures are intended to respect the rights of source-site owners and avoid implying affiliation or endorsement. Filesystem and network restrictions protect reference and evaluation materials, while audits check for copying and answer leakage (Appendix C).

REPRODUCIBILITY STATEMENT

The code and dataset are available through our anonymous repository: `https://anonymous.4open.science/r/mockweb-bench-208C/README.md`. The code release includes the React template, runner, prompts, screenshot and test-generation tools, evaluator, copying and access detectors, and tests. The dataset release includes reference sites, metadata, screenshots, visual assertions, test suites, and validation reports. Appendix A describes reference construction, manual page review, and test generation. Appendix D gives the run settings and rules for handling failures.

REFERENCES

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, Victor Zhong, Lu Chen, Kai Yu, and Tao Yu. Spider2-v: How far are multimodal agents from automating data science and engineering workflows?, 2024. URL https://arxiv.org/abs/2407.10956.

Boshui Chen, Huiping Liu, and Shaolei Zhang. Webgrader: Training llms for web development with self-evolving programmatic grader, 2026. URL https://arxiv.org/abs/2608.06474.

Yang Chen, Minghao Liu, Yufan Shen, Yunwen Li, Tianyuan Huang, Xinyu Fang, Tianyu Zheng, Wenxuan Huang, Cheng Yang, Daocheng Fu, Jianbiao Mei, Rong Wu, Yunfei Zhao, Licheng Wen, Xuemeng Yang, Song Mao, Qunshu Lin, Zhi Yu, Yongliang Shen, Yu Qiao, and Botian Shi. Iwr-bench: Can lvlms reconstruct interactive webpage from a user interaction video?, 2025. URL https://arxiv.org/abs/2509.24709.

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, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024. URL https://arxiv.org/abs/2403.07718.

Jonathan Gabor, Jayson Lynch, and Jonathan Rosenfeld. Evilgenie: A reward hacking benchmark, 2026. URL https://arxiv.org/abs/2511.21654.

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. ACM, April 2025. doi: 10.1145/3696410.3714889. URL http://dx.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.

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.

Bin Hong, Zhenchao Zhang, Jiyuan He, Kai Zhang, and Zhenya Huang. Framework and benchmark for code-driven agentic testing in web development, 2026. URL https://arxiv.org/abs/2609.00081.

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, 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, 2024. URL https://arxiv.org/abs/2403.09029.

Xinping Lei, Xinyu Che, Junqi Xiong, Chenchen Zhang, Yukai Huang, Chenyu Zhou, Haoyang Huang, Minghao Liu, Letian Zhu, Hongyi Ye, Jinhua Hao, Ken Deng, Zizheng Zhan, Han Li, Dailin Li, Yifan Yao, Ming Sun, Zhaoxiang Zhang, and Jiaheng Liu. WebCompass: Towards multimodal web coding evaluation for code language models, 2026. URL https://arxiv.org/abs/2604.18224.

Yuhang Li, Chenchen Zhang, Ruilin Lv, Ao Liu, Ken Deng, Yuanxing Zhang, Jiaheng Liu, Wiggin Zhou, and Bo Zhou. Relook: Vision-grounded rl with a multimodal llm critic for agentic web coding, 2025. URL https://arxiv.org/abs/2510.11498.

Dunjie Lu, Shuai Bai, Tianyi Bai, Sicheng Fan, Chang Gao, Jian Guan, Feng Hu, Mianqiu Huang, Xingyang Huang, Yizhen Jiang, Yuheng Jing, Dehui Kong, Ning Li, Dayiheng Liu, Shixuan Liu, Zheng Liu, Que Shen, Bowen Wang, Junli Wang, Chencan Wu, Rui Xie, Tianbao Xie, Zhihui Xie, Haiyang Xu, An Yang, Tao Yu, Wenzhen Yuan, Xi Zhang, Zhenru Zhang, Mingkang Zhu, Zhaoqing Zhu, Yizhong Cao, Kai Dang, Binyuan Hui, Kaixin Li, Junyang Lin, Haiquan Wang, Zekun Wang, Yiheng Xu, Fan Yan, Mengqi Yuan, Danyang Zhang, Jiajun Zhang, Zhipeng Zhang, Fan Zhou, and Fan Zhou. Qwen-cua: Native computer use for (almost) everything, 2026. URL https://arxiv.org/abs/2608.02352.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning, 2025a. URL https://arxiv.org/abs/2509.22644.

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, 2025b. URL https://arxiv.org/abs/2505.03733.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. Webcanvas: Benchmarking web agents in online environments, 2024. URL https://arxiv.org/abs/2406.12373.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering, 2025. URL https://arxiv.org/abs/2403.03163.

Sanjari Srivastava, Gang Li, Cheng Chang, Rishu Garg, Manpreet Kaur, Charlene Y. Lee, Yuezhang Li, Yining Mao, Ignacio Cases, Yanan Xie, and Peng Qi. Warc-bench: Web archive based benchmark for gui subtask executions, 2026. URL https://arxiv.org/abs/2510.09872.

Hung Tran, Langston Nashold, Rayan Krishnan, Antoine Bigeard, and Alex Gu. Vibe code bench: Evaluating ai models on end-to-end web application development. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pp. 514–536. ACM, May 2026. doi: 10.1145/3786335.3813180. URL http://dx.doi.org/10.1145/3786335.3813180.

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, 2024. URL https://arxiv.org/abs/2412.15310.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

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.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R. Lyu. Designbench: A comprehensive benchmark for mllm-based front-end code generation, 2026. URL https://arxiv.org/abs/2506.06251.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

Boxi Yu, Yang Cao, Yuzhong Zhang, Liting Lin, Junjielong Xu, Zhiqing Zhong, Qinghua Xu, Guancheng Wang, Jialun Cao, Shing-Chi Cheung, Pinjia He, and Lionel Briand. Swe-abs: Adversarial benchmark strengthening exposes inflated success rates on test-based benchmark, 2026. URL https://arxiv.org/abs/2603.00520.

Peng Yuan, Yuyang Yin, Yuxuan Cai, and Zheng Wei. Webforge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark, 2026. URL https://arxiv.org/abs/2604.10988.

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, 2024. URL https://arxiv.org/abs/2406.20098.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. Swe-bench goes live!, 2025. URL https://arxiv.org/abs/2505.23419.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018. URL https://arxiv.org/abs/1801.03924.

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, 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, 2025. URL https://arxiv.org/abs/2506.13832.

A Dataset and Test Construction

A.1 Reference construction

Reference selection also removes duplicate language or regional versions, overly simple pages, and pages that display raw source code or data.

Creating fictional site content. The brand template takes source-site titles, descriptions, headings, navigation, and sample text as input and produces a fictional name, domain, tagline, industry, and writing style. Text rewriting uses this identity and the text fields on each page to produce prose of similar length, preserving technical terms, API names, code, paths, and versions. Media prompts combine each source image with its role and dimensions to preserve the subject, composition, and broad visual style while replacing branding and identifiable people. Image generation is followed by site assembly, checks, and reference capture.

Evaluation capture settings. Reference and candidate screenshots for evaluation use 1920×1080 desktop and 375×812 mobile viewports, in CSS pixels, as specified in §3.1. We capture each page in a fresh browser context, wait for navigation and content, scroll to trigger lazy loading, and wait for fonts. We suppress animations and autoplay and fix the seed for JavaScript random numbers. Mobile capture enables touch and mobile emulation with a default device scale factor of three. Long pages are checked against capture-size limits and recaptured at lower resolution when needed. Layout snapshots and desktop semantic-region crops accompany the screenshots.

A.1.1 Manual review of reference pages

Manual review takes place during reference preparation. For each synthetic target page, reviewers inspect full-page captures at 1280×900 (desktop) and 375×812 (mobile) CSS pixels, with device scale factors of one and three, respectively. These sizes apply to manual review; evaluation screenshots use the capture sizes specified above. Reviewers first practice applying the criteria to ten common pages, then review complete assigned sites, using each site's fictional identity as the reference for content checks. Modified copies retain the rendering review performed in the original collection.

Table 5 gives the rendering criteria (R1–R6) and the additional content criteria for synthetic references (C1–C5). Reviewers inspect both viewports, including footers, logo walls, captions, contact details, and text embedded in images. A page receives `verified` only when both captures pass every criterion; a failure or unresolved case in either viewport receives `rejected` and is returned for repair or exclusion.

Table 5: Rejection criteria for manual review of synthetic reference pages.

Code	Rejection condition
R1	More than one third of content images are broken or missing, or a key hero, product, or article image is broken or replaced by an empty placeholder.
R2	Missing CSS or an unstyled page.
R3	Blank main content or empty central cards, lists, or grids.
R4	An obstructing popup covering more than 30% of the viewport, a login wall, CAPTCHA, or an error page.
R5	Overlapping, displaced, or clipped content; mobile horizontal overflow; or blank video and media placeholders.
R6	Legible gibberish or placeholder text in images, missing or unreadable diagram labels, or a severe mismatch between an image and its context.
C1	Recognizable real brand or product names, including text embedded in images.
C2	Real logos or trademark graphics presented as brand identifiers.
C3	Recognizable real people's names or likenesses, including team and testimonial images.
C4	Visible real domains, URLs, corporate email addresses, or contact details.
C5	Recognizable protected visual material, such as product screenshots, map tiles, book covers, film posters, artworks, or characters.

Accepted content includes the assigned fictional brand and `.example` domain, generic technical terms, unbranded functional icons, and generic imagery suited to the page. Minor nonessential gaps and capture-related whitespace at the bottom of very long mobile pages are accepted when the main content is complete.

Each site's manual review report records the reviewer, date, page identifier, route, and verdict. Rejections record the criterion, viewport, location, and observed issue; rendering issues are marked `critical` and content issues `blocker`. Report totals are checked against the page inventory.

A.2 SITE INVENTORY

Tables 6 and 7 list reference sites, construction types, and evaluated page counts. Synthetic variants contribute 3,574 evaluated pages and modified copies contribute 527.

Table 6: Reference sites, construction types, and evaluated page counts (1/2).

Reference site	Construction type	Pages
ablira.example	Synthetic variant	38
aperlio.example	Synthetic variant	135
astrovia.example	Synthetic variant	27
borvia.example	Synthetic variant	69
bramble.example	Synthetic variant	20
breezel.example	Synthetic variant	107
buildara.example	Synthetic variant	54
cascora.example	Synthetic variant	58
climatewatch.org	Modified copy	29
corravale.example	Synthetic variant	20
cresvia.example	Synthetic variant	22
detox.github.io	Modified copy	110
fluvica.example	Synthetic variant	60
formora.example	Synthetic variant	138
fynora.example	Synthetic variant	54
glyphel.example	Synthetic variant	144
gradyn.example	Synthetic variant	81
haulvio.example	Synthetic variant	87
inclura.example	Synthetic variant	77
kartova.example	Synthetic variant	57
keycloak.org	Modified copy	104
klaravo.example	Synthetic variant	73
klavrium.example	Synthetic variant	107
mendova.example	Synthetic variant	66
nimbric.example	Synthetic variant	36

Table 7: Reference site inventory (2/2). The total covers both tables.

Reference site	Construction type	Pages
obseron.example	Synthetic variant	130
postgresql.org	Modified copy	91
praxide.example	Synthetic variant	126
praxora.example	Synthetic variant	27
quorvex.example	Synthetic variant	36
rankspire.example	Synthetic variant	116
relvyn.example	Synthetic variant	77
rituvia.example	Synthetic variant	134
rust-lang.org	Modified copy	90
sentorra.example	Synthetic variant	158
sonelio.example	Synthetic variant	59
sonivex.example	Synthetic variant	85
streamora.example	Synthetic variant	131
talvio.example	Synthetic variant	137
terravyn.example	Synthetic variant	72
vaultera.example	Synthetic variant	134
vellora.example	Synthetic variant	102
veranthe.example	Synthetic variant	22
verendale.example	Synthetic variant	56
verentia.example	Synthetic variant	63
veridara.example	Synthetic variant	81
verrindale.example	Synthetic variant	94
vizmara.example	Synthetic variant	147
weavora.example	Synthetic variant	57
webpack.js.org	Modified copy	103
Total: 50 sites (44 synthetic variants; six modified copies)		4,101

A.3 TEST GENERATION AND FILTERING

Functional checks: interaction generation. As in §3.3, Functional combines scripted checks with agent-assisted general and interaction suites. The interaction-test generator explores four kinds of behavior: navigation across pages, dropdown menus, changes within a page, and mobile navigation. Scripts add arrival checks from the navigation graph, pairing links or selection controls with text specific to the destination. Each case records its interaction depth, the number of steps across page or interface-state boundaries, and an outcome type (`reach`, `transition`, or `value`).

Selectors and observable outcomes. To make functional checks reusable across independent implementations, the generation prompts favor semantic selectors and prohibit CSS class selectors (Appendix G.3). The scripted navigation generator uses roles and labels as well: for example, `getByRole('link', {name: 'Docs'})` matches a link labelled "Docs" despite changes to its CSS classes or enclosing elements. Dropdown options are selected by their displayed labels. For restored controls, the test templates include standard HTML selectors as alternatives to attributes added during reference preparation. They check changes in control state or visible content instead of unconditionally requiring those attributes. Arrival checks likewise require destination-specific content to appear after a real click; shared navigation or footer text is excluded from the arrival target.

Reference execution and filtering. We replay the dedicated interaction suite on the locally served reference with Playwright. Only cases with a passing execution result are retained; failures, timeouts, and skipped cases are pruned before the suite is frozen. This gate checks the generated action sequences and assertions against the reference's behavior before the same tests are applied to candidates.

Two-hop interaction example. The Vellora case starts at `/wireframes/`, clicks the link to `/ai/`, and checks that "Flowchart AI" newly appears. It then clicks "Mind map AI" and checks the new heading "Free AI mind map creator" at `/ai/ai-mind-maps/`. A test helper records visible elements and their roles before and after each action; each step passes when a new role–text pair contains the destination text. This case has depth two and the `value` label; Figure 5 shows its local replay.

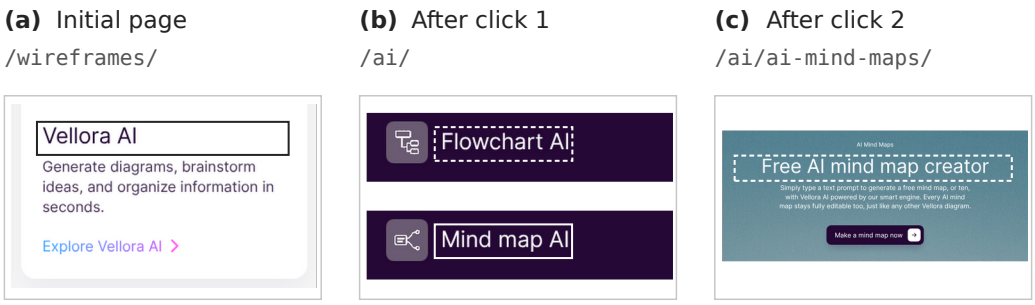


Figure 5: Crops from the Vellora two-hop replay. Solid outlines: click targets; dashed: newly matched text. All three checks pass.

Test inventory. Table 8 inventories executable test files across the 50 sites, grouped by evaluation dimension and file group. General and interaction checks both contribute to Functional. The reported Visual score uses the VLM assertions described in Appendix B.2; legacy Visual test files are listed separately from those assertions. The dedicated interaction suite contains 3,874 `reach`, 931 `transition`, and 2,027 `value` cases, plus one unclassified case. Of these, 2,803 have depth at least two. There are 42–299 interaction tests per site.

Table 8: Playwright test declarations before evaluator filtering.

Dimension / file group	Scripted	Agent-assisted
Functional: general	29,056	2,591
Functional: interaction	—	6,833
Visual: legacy files	8,186	1,120
Quality	15,683	1,333
Structural	11,368	1,401

Cross-page visual filtering. The cross-page filter lowercases assertion text, extracts alphabetic tokens, removes stopwords and tokens of at most two characters, and greedily clusters token sets at

Jaccard similarity ≥ 0.60. Candidate clusters are retrieved using each assertion's two rarest tokens. A cluster is removed when it spans at least $\max(2, \mathrm{round}(N/3))$ distinct pages, where N is the number of pages represented in the generated lists; desktop and mobile views of one page count together.

For the six modified copies, an additional filter removes assertions combining a logo, mascot, or related artwork term with a configured brand or mascot name, keeping visual requirements tied to the replacement artwork. The generation and judge prompts are provided in Appendix G.

B Score Computation

We detail the Functional, Visual, Quality, and Structural dimensions used in §4. Their weighted combination gives Raw, and the copying cap gives Final (Equations 1 and 2). The optional pixel and layout metrics below are separate from the Visual score used in the reported results.

B.1 Functional scoring

The evaluator assigns tiers from test-title patterns and applies the tier multipliers gate 0.10, struct 0.15, content 1.00, routing 0.40, interaction 0.40, and other 0.30. Thus $s_{\mathrm{func}} = \sum_i w_i \mathbf{1}[i \text{ passes}] / \sum_i w_i$. Gate checks cover load or title conditions, content checks include heading reproduction, the `struct` tier includes link checks, and routing checks cover navigation. These are tiers within Functional; the Structural dimension is defined separately in Appendix B.4. The dedicated interaction prefix takes precedence over other patterns.

B.2 Visual scoring

The release stores 206,949 assertions in 8,201 page–viewport sets. For page p and viewport v, let J_{pv} index the scored assertions, z_{pvj} be their verdicts, and w_{pvj} their weights. Let ρ_v denote the viewport weights in §4, and let V_p contain the scored viewports for page p. The aggregation is

$$q_{pv} = \frac{\sum_{j \in J_{pv}} w_{pvj} z_{pvj}}{\sum_{j \in J_{pv}} w_{pvj}}, \qquad q_p = \frac{\sum_{v \in V_p} \rho_v q_{pv}}{\sum_{v \in V_p} \rho_v}, \qquad s_{\mathrm{vis}} = \frac{1}{|P_{\mathrm{vis}}|} \sum_{p \in P_{\mathrm{vis}}} q_p. \tag{3}$$

Here P_{vis} is the set of pages contributing viewport scores. All released assertions have unit weight, $w_{pvj} = 1$. The desktop and mobile weights are $\rho_{\mathrm{desktop}} = 0.60$ and $\rho_{\mathrm{mobile}} = 0.40$, matching §4.

B.3 Optional visual diagnostics

The evaluation toolkit provides pixel and layout scoring as an optional alternative to VLM assertion scoring. All reported benchmark results use the VLM assertion score defined in Appendix B.2 for s_{vis}.

For page p and viewport v, pixel similarity P_{pv} combines structural similarity (SSIM) (Wang et al., 2004) and learned perceptual similarity ($1 - \mathrm{LPIPS}$) (Zhang et al., 2018). Layout overlap and relative page height then give the full-page score F_{pv}:

$$P_{pv} = G(\mathrm{SSIM}_{pv}, 1 - \mathrm{LPIPS}_{pv}), \qquad F_{pv} = G\big(P_{pv}, \mathrm{LayoutIoU}_{pv}\big) \cdot \frac{\min(h_{pv}^{\mathrm{ref}}, h_{pv}^{\mathrm{cand}})}{\max(h_{pv}^{\mathrm{ref}}, h_{pv}^{\mathrm{cand}})}. \tag{4}$$

Here h_{pv}^{ref} and h_{pv}^{cand} are the reference and candidate heights in CSS pixels. G denotes the geometric mean; the two means are nested. Screenshot pairs are resized to a common width and compared over their shared height. SSIM uses RGB channels; for tall images it takes the 10th percentile over eight horizontal bands, emphasizing poorly reconstructed portions. LPIPS averages distances over sampled square patches. Layout-IoU matches bounding boxes within semantic role groups after normalizing coordinates by page width and height. The coverage factor uses screenshot heights converted to CSS pixels, accounting for device scale. It penalizes both truncated pages and excess height beyond the shared comparison area.

Full-page scores combine desktop and mobile views with weights 0.60 and 0.40. When reference region crops exist, the page score blends this result with a region score at $0.80 : 0.20$. Regions are

captured on desktop and use semantic selectors for header, main content, and footer. Each region uses the pixel score, and the 25th percentile across regions forms the region score. The site score averages the resulting scores over evaluation pages.

B.4 Quality and Structural scores

The continuous measurements below are combined with executable-test pass rates to obtain the Quality and Structural scores in §4.

B.4.1 Continuous quality score

For page p, the continuous quality score combines agreement on accessibility features A_p, HTML implementation conventions H_p, and sampled text contrast C_p:

$$Q_p = 0.55A_p + 0.25H_p + 0.20C_p, \qquad A_p = 0.60A_p^{\text{ref}} + 0.40A_p^{\text{live}}. \tag{5}$$

The site score is the arithmetic mean of evaluated page scores, computed from unrounded page values and reported to four decimal places.

A_p^{ref} is the mean of applicable checks on the captured reference and candidate DOM body subtrees. Let $\mathcal{L} = \{\texttt{header}, \texttt{nav}, \texttt{main}, \texttt{section}, \texttt{article}, \texttt{aside}, \texttt{footer}\}$, and let $\mathcal{S}$ add `figure`, `figcaption`, and `h1`–`h6` to $\mathcal{L}$. The checks are:

1. Landmark agreement: $2|L_r \cap L_c|/(|L_r| + |L_c|)$, where L_r, L_c are the sets of tags from $\mathcal{L}$ present in each body; included only if L_r is nonempty.
2. Heading outline: Python `difflib.SequenceMatcher` ratio, with its default settings, between the DOM-order sequences of heading levels; included only if the reference has headings.
3. Single-heading agreement: one if the statements "exactly one `h1` exists" agree between reference and candidate, zero otherwise; included whenever the reference has any heading.
4. Semantic distribution: $1 - \frac{1}{2}\sum_{t\in\mathcal{S}} |f_r(t) - f_c(t)|$, where f normalizes tag counts over $\mathcal{S}$. Included only if the reference has such tags; a candidate with none receives zero.
5. Image alternative-text coverage: $\min(1, a_c/a_r)$ when reference coverage $a_r > 0$. Coverage is the fraction of `img` nodes with an `alt` attribute, including empty attributes. Candidate coverage is zero if it has no images.

The live Quality probes use a 1366×900 viewport with reduced motion. This viewport applies to those probes; Visual evaluation uses the screenshot capture settings in Appendix A.1. A_p^{live} equally averages four candidate-side checks: nonempty `html[lang]`, presence of `meta[name="viewport"]`, descriptive-link fraction, and labeled-form-control fraction. A link with a nonempty `href` passes if it has at least two trimmed text characters, a descendant image, or a nonempty `aria-label` or `title`. A non-hidden `input`, `select`, or `textarea` passes if it has a matching `label[for]`, an enclosing label, or a nonempty `aria-label`, `aria-labelledby`, or `placeholder`.

The HTML term is $H_p = 0.60h_p^{\text{sem}} + 0.40h^{\text{inline}}$. If $n_{\mathcal{S}}$ and n_{div} count candidate-body semantic tags and `div` nodes, respectively, then

$$h_p^{\text{sem}} = \min\left(1, \frac{n_{\mathcal{S}}/(n_{\mathcal{S}} + n_{\text{div}})}{0.15}\right), \qquad h^{\text{inline}} = 1 - \min\left(1, \frac{d_{\text{inline}}}{0.03}\right). \tag{6}$$

For the inline term, the scorer concatenates up to the first 50 `*.html` files from recursive traversal of the candidate output directory. Density d_{inline} is the fraction of concatenated characters inside quoted, lowercase `style=` attribute values. This output-wide term is reused across pages.

Contrast C_p is the passing fraction among measurable text samples. In document order, the scorer selects the first 40 elements matching `p, span, a, li, h1, h2, h3, h4, h5, h6, button, td, th, label` that have a non-null `offsetParent` and a direct text child containing more than two trimmed characters. Comparisons use parsed `rgb`/`rgba` foreground values and the first background on the element or its ancestors whose alpha exceeds 0.1, with white as the default background. Relative luminance is $Y = 0.2126g(R/255) + 0.7152g(G/255) +$

$0.0722g(B/255)$, with $g(u) = u/12.92$ for $u \le 0.03928$ and $g(u) = ((u+0.055)/1.055)^{2.4}$ otherwise. The ratio $(\max(Y_f, Y_b) + 0.05)/(\min(Y_f, Y_b) + 0.05)$ must be at least 3.0 for text of at least 24 CSS pixels, or at least 18.66 pixels with numeric font weight at least 700; other text requires 4.5.

We use $h_p^{\text{sem}} = 0.5$ when $n_{\mathcal{S}} + n_{\text{div}} = 0$ and $d_{\text{inline}} = 0$ for empty collected content. Link, form, and contrast pass fractions are one for empty sample sets.

B.4.2 Continuous structural score

We compare the body subtrees of paired reference and candidate DOM snapshots, normalizing both bodies before computing page score d_p:

$$d_p = 0.30R_p + 0.45T_p + 0.25M_p, \qquad R_p = 0.60E_p + 0.40L_p, \qquad T_p = 0.60F_p^{\text{bag}} + 0.40F_p^{\text{order}}. \tag{7}$$

Normalization discards subtrees rooted at `script`, `style`, `noscript`, `template`, `svg`, `link`, `meta`, or `iframe`. It also removes nodes with IDs `onetrust-consent-sdk`, `onetrust-banner-sdk`, `__NEXT_DATA__`, `google_tag_manager`, `fb-root`, or `CybotCookiebotDialog`, and nodes with any class beginning `otFlat`, `ot-sdk`, `gtm-`, `cookie`, or `consent`. It strips class names beginning `sc-`, `css-`, `svelte-`, or `styled-` followed by an ASCII alphanumeric character, and names consisting of an underscore followed by at least five ASCII alphanumeric characters. Nonempty text children are retained after trimming. A `div` without an ID or retained class is collapsed if its sole remaining child is an element. The similarity terms compare tags and text from the normalized trees.

E_p approximates tree structure with sequence edit distance on element-only preorder traversals, recording each node's tag, parent tag, and depth. For depth z, define $w(z) = \max(0.25, (1 + 0.2z)^{-1})$. Deleting a reference node or inserting a candidate node costs its depth weight. Substituting nodes i, j costs

$$(\mathbf{1}[t_i \neq t_j] + 0.5\mathbf{1}[\pi_i \neq \pi_j])\, w\left(\frac{z_i + z_j}{2}\right), \qquad E_p = \max\left(0, 1 - \frac{D_p}{W_p}\right), \tag{8}$$

where t and π are node and parent tags (the root's parent tag is empty), D_p is the minimum insertion/deletion/substitution cost from the usual sequence dynamic program, and W_p is the sum of depth weights in the traversal with more nodes, choosing the reference on a count tie. When either traversal exceeds 5,000 nodes, we approximate the distance as $D_p = \max(n_r, n_c) - \ell(t_r, t_c)$, retaining the same weighted denominator.

Here ℓ denotes the implementation's longest-common-subsequence (LCS) routine. It is exact when the product of sequence lengths is at most 200,000. Above that threshold, it scans the reference sequence and greedily matches each item to the earliest equal candidate item after the previous match. The routine is used for the large-DOM distance, tag sequence match, and ordered text score. Tag match is $L_p = \ell(t_r, t_c)/\max(n_r, n_c)$. Text nodes are lowercased and split with Python's `\w+` regular expression, then concatenated in DOM order. If m_r, m_c are token counts, b is the multiset intersection count, and ℓ_x is the token LCS length, then $F_p^{\text{bag}} = 2b/(m_r + m_c)$ and $F_p^{\text{order}} = 2\ell_x/(m_r + m_c)$. For E_p, L_p, and both text F1 terms, two empty sequences score one and exactly one empty sequence scores zero.

Semantic agreement uses the landmark tag set $\mathcal{L}$ defined above. With U_r, U_c the sets present after normalization,

$$M_p = \frac{|U_r \cap U_c|}{|U_r|}\,(0.70 + 0.30P_p)\,. \tag{9}$$

For matched `header`, `nav`, and `footer`, P_p is the fraction whose first candidate occurrence has zero-based preorder index divided by candidate element count at most 0.3, at most 0.4, and at least 0.7, respectively. These thresholds apply to normalized DOM preorder positions. We use $M_p = 1$ for $U_r = \varnothing$ and $P_p = 1$ when no position check applies.

The site score is the arithmetic mean of evaluated page scores, computed from unrounded page values and reported to four decimal places.

B.4.3 Reported Quality and Structural scores

Let $\bar{Q}$ and $\bar{d}$ denote the site-level continuous scores defined above, and let r_{qual} and r_{struct} be the corresponding executable-test pass rates. The dimension scores used in Equation 1 are

$$s_{\text{qual}} = 0.60\,\bar{Q} + 0.40\,r_{\text{qual}}, \qquad s_{\text{struct}} = 0.70\,\bar{d} + 0.30\,r_{\text{struct}}.$$

C Isolation and Copying Audits

C.1 Runtime isolation

The runner applies a privilege-reduction wrapper to initial rollouts, continuations, and builds. The reference server runs outside this wrapper. Network relays use Unix-domain sockets; the final build performed by the runner uses no relays. The deployment configures filesystem permissions and protects the isolation-status file. If requested network isolation cannot be established, the runner stops the task as described in §3.4.

C.2 Copying detection

The task permits reuse of binary media and individual style values, such as colors or spacing, while requiring an independently authored implementation (Appendix G.1). Reusing these permitted materials, matching the reference's appearance, or using an HTML-injection call alone is not sufficient evidence of copied implementation. The audit inspects submitted source and build files for copied reference code. It checks repeated build markers together with copied content or injected HTML, stored copies of the reference DOM, large blocks of raw HTML, and reuse of distinctive reference class names. Build-marker detection requires at least 20 occurrences plus evidence from content size, source origin, or HTML injection. Class-name matching ignores shared utility classes. HTML-injection checks combine the code pattern with evidence of its source. Appendix E.4 reports access-attempt counts. A `SCRAPE` verdict applies the Final-score cap in Equation 2. Access-attempt detections are recorded for analysis and do not by themselves change the score.

C.3 Source review results

Source review supports all 25 copying caps applied to Gemini 3.7 Flash, the system with the largest Raw–Final gap in Table 3. Every flagged Gemini submission contains copied reference code in its archived source or build, supporting these penalties under the independent-authorship requirement.

We reviewed all 36 automatically capped submissions in the 10×50 evaluation and 50 uncapped submissions. The uncapped group comprises five randomly sampled submissions per system and the two Opus 5 submissions completed after that sample was drawn. Reviewers compared archived candidate files with the released reference implementations, distinguishing transferred HTML, CSS, or layout data from permitted text, media, and individual design values. Initial judgments were frozen before comparison with automatic labels; every disagreement received an additional source review with the automatic label withheld.

Table 9: Source review of copying decisions. All capped outputs; sampled uncapped outputs.

Submission group	Reviewed	Copying identified	No copying identified
Gemini 3.7 Flash, capped	**25**	**25**	**0**
Other systems, capped	11	11	0
Uncapped sample	52	0	52

Across the reviewed submissions, automatic copying decisions agree with source-level evidence: all 36 capped submissions contain transferred implementation code, while no such transfer is identified in the uncapped sample. This agreement supports the application of copying penalties under the independent-authorship requirement. In particular, confirmation of all 25 Gemini 3.7 Flash cases provides direct evidence for the copying decisions label underlying its large Raw–Final gap.

Freezing initial judgments before comparison and withholding labels during additional reviews help reduce confirmation bias.

D Experimental Details

D.1 Model and inference configuration

Table 10 lists the model identifiers and inference settings.

Table 10: Model-specific inference settings. CC denotes Claude Code; Image denotes permission to read image pixels. The 1M flag and numerical window override are independent settings. Output limits apply per response. Token counts use $\mathrm{k} = 1{,}000$; – means no explicit setting at launch.

Model	Agent	Image	1M flag	Window	Output	Effort	Think budget
`claude-opus-4-8`	CC	Yes	True	–	128k	max	adaptive
`claude-opus-5`	CC	Yes	True	–	128k	max	adaptive
`gpt-5.6-sol`	Codex	Yes	True	–	128k	max	–
`azure.gpt-6-astra`	Codex	Yes	True	–	128k	max	–
`gemini-3.7-flash`	CC	Yes	True	–	64k	high	–
`glm-5.3`	CC	No	True	–	128k	max	–
`dashscope.qwen3.7-plus`	CC	Yes	–	256k	32k	enabled	16k
`dashscope.qwen3.8-27b`	CC	Yes	–	256k	32k	xhigh	–
`dashscope.qwen3.8-max`	CC	Yes	–	256k	32k	xhigh	–
`grok-4.6`	CC	Yes	–	–	128k	xhigh	–

The Opus configurations use the native Anthropic interface with adaptive thinking and no temperature setting. Gemini has thinking enabled.

Workspace and judge settings. The workspace uses React 19, Vite 7, Tailwind CSS 4, and TypeScript. The initial agent call has a six-hour budget, as in §3.1. The overall job limit is twelve hours and covers preparation, agent calls, builds, and evaluation; five jobs have a fourteen-hour cap.

The visual judge, Qwen3.7-Plus, scores the fixed visual assertions. Judge request settings are a 4,096-token output limit, temperature zero, and concurrency eight.

Resuming agents that return without code. If the initial call ends without timing out and the agent has neither added nor modified any source file under `src/`, the runner resumes the same session with a prompt to begin implementation. The prompts address refusals over copyright concerns and ask the agent to write its own implementation of the reference website. The runner allows up to five follow-up calls, checking for source changes before each one. Each call is limited to 30 minutes, and all calls remain within the overall job time limit. Appendix G.2 provides these prompts and the separate prompt used to resume Claude Code after an error.

D.2 Run accounting

System scores average scored sites with equal weight, including recorded zeros. Each model–site run is counted once, including any retries.

D.3 Functional test-group results

Table 11 compares the weighted pass rates of the two agent-assisted groups within Functional: general content and navigation tests, and dedicated interaction tests. For each system, both means and their difference use its n sites with both group scores. Table 4 averages these unrounded system means with equal weight. Paired bootstrap draws resample the same sites for both groups.

Table 11: Agent-assisted functional scores (%). gap: general minus interaction (pp, before rounding). Intervals: 95% paired-site bootstrap CIs.

System	General functional	Dedicated interaction	Gap	95% CI
GPT-6 Astra	58.96	34.07	24.89	[18.71, 31.02]
GLM-5.3	67.97	34.07	33.90	[27.32, 40.29]
Claude Opus 5	63.15	31.89	31.26	[23.84, 38.70]
Grok-4.6	67.68	25.18	42.51	[36.88, 48.24]
GPT-5.6 Sol	47.49	18.18	29.31	[22.75, 35.52]
Qwen3.8-27B	56.36	23.37	32.99	[26.58, 39.33]
Qwen3.8-Max	49.05	16.78	32.27	[26.65, 37.82]
Claude Opus 4.8	46.21	12.75	33.46	[28.39, 38.50]
Qwen3.7-Plus	17.66	3.65	14.01	[11.29, 16.95]
Gemini 3.7 Flash	41.15	22.94	18.21	[11.67, 24.57]

D.4 Functional test reliability

The reliability checks show reproducible outcomes and sensitivity to missing navigation. The general–interaction score gap also persists in a subset that excludes CSS selectors and checks on DOM classes, attributes, or styles.

Paired defect controls. We use all 12 passing interaction cases in the audit sample, spanning seven independently generated implementations. For each case, we disable the first navigation action used by the test while leaving the page, visible control, and test assertions unchanged. A matched control installs the same instrumentation without disabling navigation. All 12 controls pass, and all 12 injected navigation faults are detected (Table 12). The checks therefore respond to a missing navigation outcome even when its page and clickable control remain present.

Table 12: Replay and paired defect checks. The fault intervention changes one required navigation action; the original test assertions are retained.

Validation check	Result
Reference cases passing	75 / 75
Historical outcomes reproduced	75 / 75
Matched controls passing	12 / 12
Injected navigation faults detected	12 / 12

Robustness to selector choice. We define a conservative *direct role/text-based subset* from the available test source, fixing the rules before inspecting its outcomes. Eligible cases use semantic roles, visible text, labels, or placeholders directly; CSS selectors, class/attribute/style assertions, and custom source-level helpers are excluded. The benchmark's fixed visible-state helper remains unchanged. This retains 5,399 interaction-test results across 91 implementations with both general and subset scores. Using the same runs for each paired comparison, interaction performance remains lower in all 10 systems. The equal-system mean general–interaction gap is 24.15 percentage points. Thus, the ordering persists without those reference-specific DOM constraints.

Stratified replay audit. From 124 retained implementations, we sample two per system, then up to three cases per system and declared-depth group (one, two, or at least three steps), without selecting on pass or failure. The resulting 77 cases cover 20 implementations and 17 sites. We replay the 75 cases with matched test source against the reference and candidate using the benchmark's static server and original assertions and time limits. All pass on the reference, and every candidate result matches its recorded outcome; Table 13 also retains the two cases without matched source.

In 48 of the 63 replayed nonpassing cases, browser and artifact inspection finds concrete reconstruction defects: missing pages, missing required content, absent navigation links, or omitted control-state transitions. Table 13 groups the findings by declared interaction depth. A timeout receives a diagnosis only when the browser or artifact evidence identifies a specific issue.

Table 13: Primary findings in the stratified audit. Counts describe this diagnostic sample. Page and content prerequisites take priority when issues co-occur; the labels describe observed reconstruction or compatibility issues.

Finding	Depth 1	Depth 2	Depth ≥ 3	Total
Original pass	8	4	0	12
Missing page or error page	7	4	3	14
Required content missing	5	5	3	13
Required navigation link missing	2	3	3	8
Control or state transition missing	3	6	4	13
Selector or state encoding mismatch	4	7	4	15
Source unavailable for replay	1	1	0	2
Total	30	30	17	77

The audit also identifies implementation-sensitive checks, including DOM IDs, button versus tab roles, and different encodings of the same selected form value. For these cases, supplementary checks follow the corresponding controls and verify the reference-supported observable outcomes. They preserve required menu steps and do not substitute unrelated links for missing controls. Original test outcomes and these supplementary behavioral checks are recorded separately.

D.5 Reconstruction time

We read the start and end times from each generation job's runner log. Timing starts when the runner activates the agent's network isolation and ends at the first build handoff after the agent stage. The wall-clock interval includes continuations, builds requested by the agent, and waiting for models and tools. It excludes workspace preparation, the runner's subsequent build, evaluation, and artifact upload.
We match each interval to the generation job and attempt that produced the scored candidate. Runner logs record both timestamps. Table 14 gives the median times plotted in Figure 3.

Table 14: Median reconstruction-stage time.

System	Median time (min)
GPT-6 Astra	136.1
GLM-5.3	131.0
Claude Opus 5	212.7
Grok-4.6	68.4
GPT-5.6 Sol	396.9
Qwen3.8-27B	113.4
Qwen3.8-Max	140.6
Claude Opus 4.8	96.7
Qwen3.7-Plus	16.3
Gemini 3.7 Flash	38.4

D.6 Site scale and reconstruction performance

We measure site scale by the number of manually checked evaluation pages in the release inventory. We use the complete 10-system, 50-site experiment, averaging each site's scores across all ten systems before comparing groups of sites with different page counts. Dedicated interaction comparisons use sites where that subscore is defined for every system.

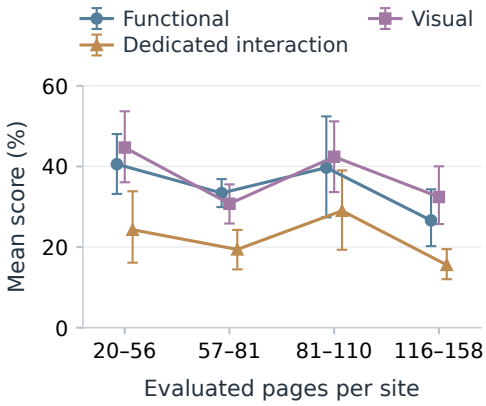


Figure 6: Reconstruction by site scale. Points: group means; bars: 95% site-bootstrap intervals. Groups: Table 15.

Table 15: Reconstruction scores by page-count stratum (10 systems; %). Cells show means and 95% site-bootstrap intervals. Strata contain 13/13/12/12 sites, sorted by page count and site name; the 81-page tie spans two strata.

Pages	Functional	Visual	Interaction	Final
20–56	40.56 [33.19, 48.04]	44.69 [36.07, 53.69]	24.30 [16.12, 33.84]	46.73 [41.55, 51.98]
57–81	33.40 [29.91, 36.86]	30.75 [25.84, 35.56]	19.41 [14.46, 24.26]	38.24 [35.26, 41.08]
81–110	39.70 [27.38, 52.43]	42.40 [33.64, 51.19]	29.00 [19.33, 39.01]	44.00 [36.19, 52.49]
116–158	26.61 [20.22, 34.32]	32.43 [25.71, 40.03]	15.57 [12.02, 19.48]	34.72 [29.12, 41.17]

Across sites, Spearman correlations with page count are functional $\rho = -0.381$, visual $\rho = -0.150$, dedicated interaction $\rho = -0.092$, Final $\rho = -0.346$.

D.7 Effect of providing page URLs

We compare reconstruction with the evaluation page URLs hidden or provided. From the fixed sample of 20 websites, we uniformly sample 15 paired sites per model without replacement (seed 20260918), using sites with verified outcomes in both conditions. Within each model, the paired runs share reference packages, agent configuration, and time budgets. Providing the URLs identifies the evaluated pages and gives clues about their content.

Table 16: Paired final scores (%) with hidden and provided evaluated URLs. Difference is provided minus hidden, in percentage points.

System	Hidden URLs	Provided URLs	Difference
GPT-6 Astra	57.37	55.57	-1.80
Claude Opus 4.8	38.72	52.50	13.78

Table 17: Raw and component score differences on the same paired sites as Table 16, in percentage points (provided minus hidden). Copy caps counts runs with the copying cap in the hidden/provided conditions.

System	Raw	Functional	Visual	Quality	Structural	Copy caps
GPT-6 Astra	2.90	6.02	0.66	0.07	2.17	0/1
Claude Opus 4.8	13.78	24.62	8.19	-0.89	7.44	0/0

For GPT-6 Astra, the raw difference of 2.90 percentage points becomes a final difference of -1.80 percentage points; copying penalties make the latter 4.70 percentage points lower.

D.8 Score reliability

Sensitivity to component weights. Table 18 holds candidate outputs and copying flags fixed while varying the four component weights. Alternative settings use equal weights $(0.25, 0.25, 0.25, 0.25)$; functional and visual scores only $(0.50, 0.50, 0, 0)$; functional emphasis $(0.50, 0.30, 0.10, 0.10)$; and visual emphasis $(0.30, 0.50, 0.10, 0.10)$. The order is functional, visual, quality, and structural.

For each alternative, we recompute Final using Equation 2 with the fixed copying verdicts, then average over sites. The analysis includes all ten systems.

Table 18: Final scores (%) under alternative component weights. ρ: Spearman rank correlation with the standard-weight ranking in Table 3.

System	Equal	Func. + vis.	Func. emph.	Vis. emph.
GPT-6 Astra	63.48	51.44	55.05	57.46
GLM-5.3	61.61	47.72	53.25	53.30
Claude Opus 5	60.52	47.58	52.36	53.15
Grok-4.6	56.67	39.33	47.43	45.10
GPT-5.6 Sol	54.80	37.26	42.82	45.72
Qwen3.8-27B	54.82	36.72	43.54	44.39
Qwen3.8-Max	52.35	32.01	39.92	40.38
Claude Opus 4.8	48.30	24.27	33.36	34.41
Qwen3.7-Plus	38.95	10.19	21.65	21.73
Gemini 3.7 Flash	25.58	11.27	17.22	17.49
Rank ρ	0.988	0.988	0.988	0.988

Human visual agreement pilot. We evaluate visual agreement on 48 reference–candidate image pairs from twelve sites and two configurations, GPT-6 Astra and Opus 4.8. The sites are sampled from a previously fixed 25-site pool, covering all four page-count groups. Each site contributes its entry page and one randomly selected non-entry page; both configurations use the same page and viewport. The sample balances 24 desktop and 24 mobile pairs. Four frozen assertions per reference yield 96 distinct reference assertions and 192 candidate assertion judgments per rater.

Two raters independently assess whether each assertion describes the reference, whether the candidate satisfies it, and overall visual fidelity on a 0–4 scale. The scale ranges from no recognizable correspondence to closely matching layout, content, typography, colors, and imagery. Four separate image pairs provide calibration, and Chinese translations accompany the English assertions. System identities, automatic scores, and judge verdicts are hidden. Reference judgments are shared across candidates for the same reference. For judge–human comparison, we rerun Qwen3.7-Plus on the retained candidate screenshots with each page's full frozen assertion list and the benchmark judge settings, then extract the four sampled verdicts. Table 19 retains all sampled assertions when comparing the judge with each rater. We also compute agreement on the subset whose reference descriptions both raters accept. Unable-to-judge responses are recorded separately.

The raters jointly accept 91 of 96 reference assertions; they jointly reject 1 and disagree on 1. On the jointly accepted subset, judge–human agreement is 85.87% for rater 1 and 96.74% for rater 2 ($n = 182$ per rater).

Table 19: Visual evaluation agreement. Assertion rows retain all sampled items and use Cohen's κ; whole-page ratings use quadratic-weighted κ.

Judgment	Comparison	Agreement (%)	κ
Reference assertions	Human–human	97.92	0.657
Candidate assertions	Human–human	89.58	0.788
Candidate assertions	Judge–rater 1	85.42	0.701
Candidate assertions	Judge–rater 2	95.83	0.914
Whole-page fidelity	Human–human	—	0.750

Visual scores and human ratings. For each rated page–viewport, we aggregate its complete frozen assertion list using the benchmark's scoring rule and compare the resulting Visual score with the mean of the two whole-page ratings. This covers 1568 judgments across 48 screenshots, including assertions the raters did not jointly accept. The visual scores correlate with mean human ratings at Spearman's $\rho = 0.812$ (95% CI $[0.634, 0.916]$) and Pearson's $r = 0.845$. Within-system Spearman correlations are 0.820 for GPT-6 Astra and 0.759 for Opus 4.8 (24 views per system). Intervals use 10,000 whole-site bootstrap draws, retaining all rated views of each sampled website. Figure 7 shows the paired scores.

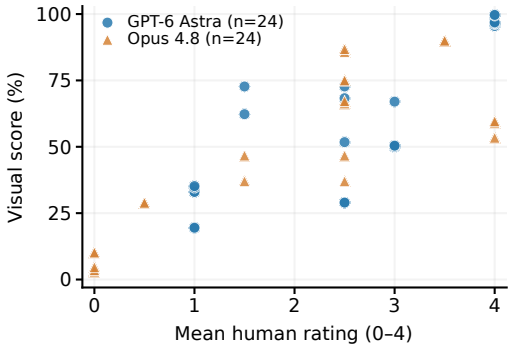


Figure 7: Visual scores from complete assertion lists and mean whole-page human fidelity ratings. Each point is one candidate screenshot; each system contributes 24 views.

E BEHAVIOR MEASUREMENTS

The experiment evaluates 10 systems on 50 websites, giving 500 model–site runs. The analyses below use evaluation outputs and recoverable tool-call records from this experiment. Each measurement applies the record requirements specified below.

E.1 TRAJECTORY DYNAMICS

Action units. We count each recorded tool call as one action. We assign main-agent actions to seven categories: view, GUI interaction, read, write, execute, plan, and other. View and GUI include browser observations, screenshots, navigation, and input actions. Read includes file and resource inspection; write includes source changes and other file-writing operations. A tool call containing a script counts as one action, classified by the code it contains. For Codex, a native patch touching multiple files is normalized to one write action per path. The corpus contains 111,819 normalized actions and 111,576 unsplit native tool calls; the difference comes from splitting these patches. Progress measures position in the recorded sequence, normalized separately for each run. Codex actions follow completed-item order; Claude Code actions follow message order.

Tool-use composition. Figure 4a shows all seven tool classes, with one stacked area plot per model and a shared progress axis. Observe UI and Interact UI correspond to view and GUI; Read and Write/Edit cover file and resource operations; Execute includes building, launching, and running programs.

We divide action progress into ten intervals. Within each run and interval, we divide each class count by the total number of actions, then average these shares within each system. Each stack sums to 100%, and each run receives equal weight. The complete-run percentages in the text are computed separately by normalizing each run's total class counts.

Source replay. We replay successful writes and recoverable replacements to reconstruct the character count of tracked source files in the delivered workspace. Each run's source-size curve is normalized by its final reconstructed size, as in §5.3. Overwrites set the tracked size to the new content length; replacements apply their change in text length. The write-call curve counts all source-write attempts with identified paths, including unsuccessful calls.

We require at least five recorded source edits and recoverable size changes for at least 70% of them. Runs containing unsuccessful shell writes to source files or edits that replace all occurrences are omitted from the pooled source-replay analysis. For source replay, action positions use $i/(N-1)$. Figure 4b shows each model's median normalized source size and cumulative source-write calls at 20 progress boundaries. Both curves use the same eligible replays within Opus 5 and GLM-5.3. For Astra, we recover source-writing calls from recorded shell arguments, including literal patches and file-write scripts, and combine them with native file-change records. Its write-call curve uses the resulting 789 identified source-writing actions. At 50% action progress, median normalized source size is 87.7% for GLM-5.3 and 66.0% for Opus 5.

For each run, we locate the first sampled boundary at which replayed size is at least 90% of the final replay, using a 5-percentage-point grid. Across eligible Claude Code replays, this threshold is first reached at a median of 62.5% action progress, with a median of 26.4% of source-write calls remaining. Table 21 checks stricter payload and shell-operation restrictions.

Context size and token use. Context size is the input-token count reported for each response, including uncached tokens, tokens read from a cache, and tokens used to create a cache. For runs with per-response records, the median per-run peak is 281,731 tokens. Per-run input totals use recorded token accounting, including repeated and cached prompt processing, with a median of 27.7 million tokens. Table 20 reports the system-level measurements. Its UI share combines Observe UI and Interact UI from Figure 4a.

Execution length and final fidelity. For each outcome we fit

$$100S_{m,s} = \alpha_m + \gamma_s + \beta \log_2 A_{m,s} + \epsilon_{m,s},$$

where A is the number of normalized actions, m identifies the system, and s the website. Component scores are measured before the copying cap. Each run has equal weight. We resample whole websites with replacement 2,000 times and refit the model for each draw; Table 22 reports 95% percentile intervals. The source-curve intervals use the same whole-site resampling scheme, recomputing each model's mean or median from its available records in the draw. A website's records are resampled together across the three displayed systems.

Table 20: Trajectory resources. Actions and token values are per-run medians; UI is the mean per-run action share.

System	Actions	UI (%)	Context (k)	Input (M)
GPT-6 Astra	107	66.6	–	15.8
Claude Opus 4.8	215	42.6	318.4	39.4
Claude Opus 5	249.5	33.1	404.7	44.1
Qwen3.7-Plus	103	37.3	99.2	5.6
Qwen3.8-27B	403	41.1	388.3	90.6
Qwen3.8-Max	194	31.9	239.3	24.7
Gemini 3.7 Flash	100.5	47.6	–	–
GLM-5.3	476	36.4	429.0	126.5
GPT-5.6 Sol	482.5	83.8	–	45.0
Grok-4.6	231	36.5	225.3	17.4

Table 21: Sensitivity of source replay. Progress is the first sampled crossing of 90% of final replayed size. Remaining writes are source-write calls after that per-run crossing. Both quantities are medians.

Replay subset	Progress (%)	Remaining writes (%)
Primary replay	62.5	26.4
At least 90% payload coverage	60.0	28.8
100% payload coverage	65.0	22.7
No source-shell writes	65.0	26.7
Initial replay set	60.0	26.7

Table 22: Execution length–score associations (pp per doubling), with 95% whole-site bootstrap intervals. All fits include system and website fixed effects.

Subset / action unit	Functional	Visual
All normalized actions	$+12.78$ $[+8.35, +17.19]$	$+13.69$ $[+9.74, +17.57]$
Unsplit native calls	$+12.83$ $[+8.38, +17.22]$	$+13.73$ $[+9.75, +17.60]$
Claude Code only	$+13.67$ $[+8.67, +18.40]$	$+14.87$ $[+10.19, +19.51]$
Excluding capped outputs	$+9.81$ $[+5.70, +13.95]$	$+11.86$ $[+8.27, +15.51]$

E.2 Browser checks

For the supplementary browser diagnostics in Figure 8b, we pair native tool calls with their results and deduplicate the main-agent transcript. These diagnostics retain native call units; the main analysis in §5.3 uses the normalized actions defined in Appendix E.1. Checks and edits use successful calls. Claude Code calls are ordered by timestamps and Codex calls by start/completion events, retaining overlapping intervals. A file-change patch counts as one tool call regardless of the number of files changed.

We distinguish reference and candidate checks using confirmed server origins (URL scheme, host, and port) and page information within each browser session. Reference proxies retain the reference role; static copies and auxiliary services are tracked separately. Reference and candidate check counts require a single identified target origin and page.

A check is an MCP navigation, snapshot, screenshot, evaluation, browser-program, or interaction call that completes without a tool execution error. We measure three behaviors:

- **Reference routes:** distinct paths and query strings among reference checks, with trailing slashes normalized and fragments omitted.
- **Check–edit–recheck:** two candidate checks must share a route and recorded viewport state, with an eligible source edit between them. The sequence stays within one browser session, with each step completed before the next begins. Eligible edits are successful writes, replacements, or patches under `src`/`public` or to specified entry/configuration files, excluding unchanged replacements and diagnostic scripts.
- **Candidate interactions:** individual input actions on the candidate, including clicks, hovering, typing, filling fields, keypresses, selections, dragging, dialog actions, and uploads. We count dedicated MCP actions and input actions identified in inline browser or terminal code. Each filled field counts separately; typing with submission additionally counts the keypress. Navigation, screenshots, and viewport changes do not count as input actions.

The median reference-route count is 8. Successful MCP checks comprise 22,822 reference checks, 12,460 candidate checks, and 3,555 checks with unresolved target roles.

Browser checks over the run. For Figure 8b, we normalize the zero-based start-order index of each paired tool call by the run's total paired-call count and use five equal-width bins with the origin roles above. In each bin, we first calculate each category's share of all paired calls within each run, then average across runs with equal weight. Every run contributes to each bin. Unresolved browser checks and other tools remain in the denominator; the figure displays reference checks, candidate checks, and source edits.

The records include candidate checks after the last explicit file edit, as well as check–edit–recheck sequences on the same route and in the same browser session.

E.3 Visual fidelity and interaction scores

Figure 8a pairs the Visual score with the dedicated interaction subscore within Functional across the 10-system, 50-site experiment. Each point represents one system–site candidate with both scores; all scores are measured before the copying cap. The five visual bins are $[0, 20)$, $[20, 40)$, $[40, 60)$, $[60, 80)$, and $[80, 100]$ in percent. Their mean interaction scores are 7.98%, 17.31%, 27.19%, 43.17%, and 53.84%. Visual and interaction scores have a Spearman correlation of 0.683.

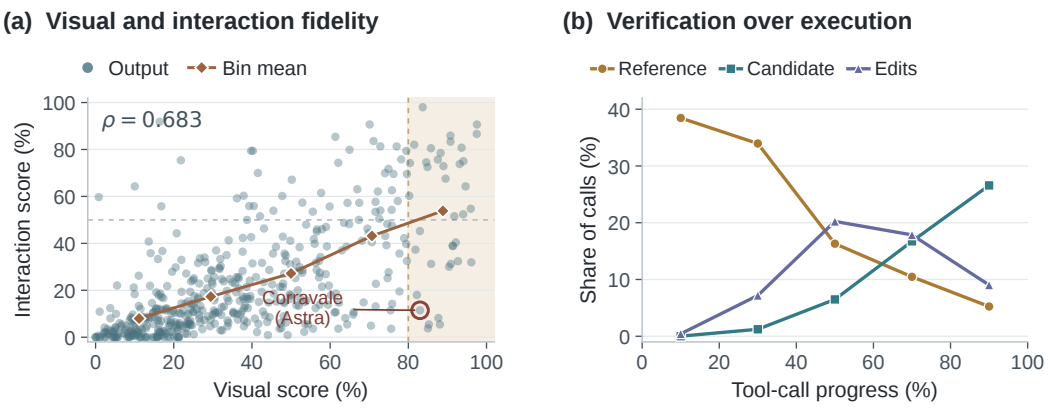


Figure 8: Interaction and verification diagnostics. (a) Visual versus interaction scores; diamonds show bin means and shading marks Visual $\geq 80\%$. (b) Mean per-run shares of reference checks, candidate checks, and source edits over progress through native calls; all calls remain in the denominator.

Table 23 checks several visual thresholds. The groups overlap: each higher threshold selects a subset of the previous group. Excluding capped outputs from the visual-at-least-80% group leaves 35 outputs with a mean interaction score of 51.34%; 16 remain below 50%. These outputs show why appearance and interaction scores need to be reported separately.

Table 23: Interaction scores for outputs meeting each visual threshold (%).

Visual score	Outputs	Mean interaction	Interaction below 50%
≥ 60	99	47.26	52
≥ 70	74	51.12	34
≥ 80	38	53.84	16
≥ 90	18	57.94	7

E.4 Access attempts

We examine access attempts in the recorded transcripts. Table 24 counts runs with at least one attempt in each category; repeated calls count once per run and category.

Counting access attempts. We classify deduplicated main-agent calls into three categories: explicit external requests from shell tools, operations on protected reference or evaluation material, and privilege-escalation commands, including capability probes. Classification uses request targets and file paths. Attempts are counted when a call is issued, regardless of its outcome.

Table 24: Runs with recorded access attempts. Each cell counts runs with at least one attempt.

System	Protected files	External requests	Privilege escalation
Claude Opus 4.8	1	0	0
Claude Opus 5	3	0	0
GPT-5.6 Sol	0	0	0
GPT-6 Astra	0	0	0
Gemini 3.7 Flash	2	0	0
GLM-5.3	1	1	0
Qwen3.7-Plus	3	1	0
Qwen3.8-27B	3	1	0
Qwen3.8-Max	2	0	0
Grok-4.6	2	0	0
Total observed	17	3	0

Observed attempts. Protected-file access calls occur 25 times in 17 runs from eight systems. Fifteen calls return explicit permission-denied errors. These attempts include traversing reference directories and reading, copying, or serving protected files. No privilege-escalation call is detected. The four explicit external-request calls occur in three runs and concern an npm installation or registry-connectivity probes. One run with a protected-file access attempt also receives a copying cap.

F CASE STUDIES

The following cases examine missing interactions despite high visual scores, image-comparison feedback, implementation copying, and retrieval of upstream source over the network.

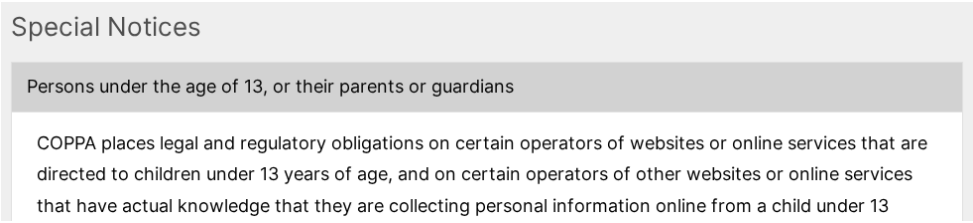
F.1 MISSING INTERACTIONS DESPITE HIGH VISUAL FIDELITY

GPT-5.6 Sol on Corravale. GPT-5.6 Sol's Corravale candidate receives 87.69% visual, 38.60% functional, and 8.20% dedicated interaction scores. Its interaction records contain 5 passes, 18 failures, and 38 timeouts across 61 tests. Table 25 maps the inspected actions to their expected content and candidate implementation. Figure 9 shows the privacy-notice replay after the same click on each site.

Table 25: Interaction omissions confirmed in Corravale's source (four failed tests; privacy notices grouped).

Action	Reference behavior	Candidate implementation
Expand Executive Offices	Reveal the Secretary of the University link.	Link to `/_404.html`; directory content absent.
Open Special Notices	Reveal the COPPA paragraph or GDPR resources link.	Static text blocks without the expected expanded content or expansion logic.
Submit a page search	Include the entered term in the URL.	Input lacks a `name`; form has no submit handler to carry the query.

(a) Reference after clicking the notice



(b) Candidate after the same click



Figure 9: Corravale privacy-notice replay at 1920×1080. Equally scaled crops: reference (top), GPT-5.6 Sol (bottom).

GPT-6 Astra's self-check on Corravale. On the same site, GPT-6 Astra's candidate receives 82.99% visual, 38.68% functional, and 11.48% dedicated interaction scores. Its final browser program traverses 13 navigation routes and checks page landmarks and horizontal overflow. At 375×812, it opens the mobile menu, expands the About page's All Colleges & Schools item, and confirms that Architecture & Urban Planning is visible.

The candidate reuses one school list across multiple expandable directories. Table 26 compares five inspected single-step expansion tests with their required entries. The About entry matches the agent's self-check and passes; the other four entries are absent from the shared list, and their tests fail.

Table 26: Five inspected single-step directory-expansion tests on Astra's Corravale candidate.

Page / directory	Required expanded entry	Result
About / All Colleges & Schools	Architecture & Urban Planning	Pass
Academics / Flagship Schools & Colleges	Informatics	Fail
Academics / Undergraduate Colleges & Schools	Commerce	Fail
Academics / Graduate Schools & Colleges	Halloran School of Graduate Studies	Fail
Research / Schools & Colleges	Architecture & Urban Studies	Fail

F.2 Image-comparison feedback

The pixel errors below are measurements computed by the agents during development. The benchmark's Visual scores use the VLM procedure in §4.

Vellora. GPT-6 Astra writes scripts that compare screenshots of its Vellora candidate with saved reference captures using the mean absolute error (MAE) of RGB values. It inspects heading typography and spacing on the reference `/mcp` page, then uses these measurements to revise mobile controls, content filtering, and responsive layouts.

The round also changes media compression and capture timing. Both checkpoints use the same saved reference screenshots. Table 27 gives the recorded MAE before and after this round for each route.

Aperlio. In a separate evaluation on Aperlio, GPT-6 Astra likewise writes Python scripts that compute RGB MAE between reference and candidate screenshots and render side-by-side comparisons. After viewing the v3 panels, it returns to the reference Photos page, inspects the grid layout and image sources, then revises image ordering and article image mappings. Photos uses the same route at all three checkpoints, and its MAE falls from 44.67 to 11.98 to 5.39 (Table 28). The two mismatched v3 pairs are marked with daggers; Figure 10 shows the Guide example. The original pairings are restored at v4.

Table 27: Agent-recorded Vellora RGB MAE before and after refinement (375×812; 0–255, lower is better).

Route	Before	After
`/pricing`	28.17	13.84
`/download`	24.03	5.17
`/blog`	31.14	8.97
`/integrations`	20.16	11.11
`/company/contact-sales`	28.19	9.20
`/learn`	11.89	5.07
`/mcp`	20.35	7.57
`/learn/contact-us`	15.25	3.77
Mean	22.40	8.09

Table 28: Agent-recorded Aperlio RGB MAE (desktop; 0–255, lower is better). †: mismatched routes at v3.

Page label	v2	v3	v4
Photos	44.67	11.98	5.39
Detail	49.31	$35.83^{\dagger}$	6.53
Guide	17.79	$45.85^{\dagger}$	9.82

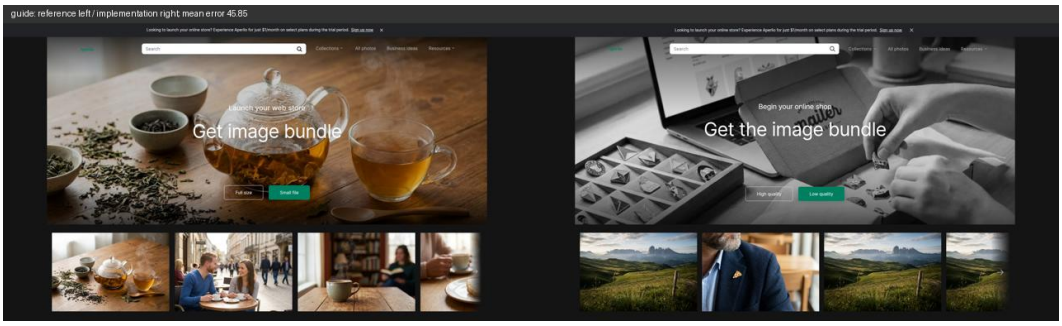


Figure 10: Mismatched Aperlio Guide captures at v3: tea-guide reference (left), enamel-pin-guide candidate (right).

F.3 Implementation copying

Nimbric. For Gemini 3.7 Flash's Nimbric candidate, `src/parsed-pages.json` contains 45 fragments matching the fetched HTML after navigation and footer removal. The stylesheet `src/extracted-site.css` matches 40 unique reference style blocks after the extraction and cleanup recorded in the trajectory. Table 29 supplies the component scores.

Table 29: Gemini 3.7 Flash's Nimbric copying case: scores (%) on 36 evaluated pages.

Score	Gemini 3.7 Flash / Nimbric
Functional	95.79
Visual	90.78
Raw	92.19
Final	10.00

Klavrium. Gemini 3.7 Flash's Klavrium candidate stores 110 nonempty body records in `src/data/fullPages.json` and `src/data/fullBlogPosts.json`. Added handlers cover internal navigation, mobile menus, newsletter forms, and expandable changelog cards. On the 107 evaluated pages, the candidate obtains 99.52% Functional and 83.68% Visual scores. Its Raw score is 92.08%, and the copying cap sets Final to 10.00%.

F.4 Retrieving upstream source over the network

Table 30 traces an August 2026 Opus 5 rollout on Digital.gov with protected reference files and external network connectivity. This is the earlier configuration cited in §3.4; it illustrates the access that the main protocol's network restrictions are designed to block.

Table 30: Key actions in the August 2026 Digital.gov rollout with external network access.

Action	Recorded outcome
List the protected reference directory.	`Permission denied`.
Locate the upstream article.	Following unsuccessful article requests and code searches, the tree of `GSA/digitalgov.gov` reveals the target Markdown file.
Download article Markdown.	The agent downloads 19 files and then two more, including the target article.
Convert downloaded Markdown into React pages.	A subagent reads the target Markdown and writes the corresponding TSX page.

The agent instructs the subagent to reproduce the downloaded article verbatim. The recorded TSX write matches the archived candidate file byte for byte, and the final build registers the article route and includes its content.

G PROMPT TEMPLATES

This appendix provides the reconstruction, test-generation, and visual-evaluation templates with their input and output formats. In Python templates, braced fields are substituted at runtime and doubled braces encode literal braces. Evaluation templates follow the Functional, Visual, Quality, and Structural dimensions in §3.3. The quoted prompt text is reproduced verbatim. In these templates, *replica* denotes the candidate and *ground truth* denotes reference data.

G.1 RECONSTRUCTION TASK AND WORKSPACE INPUTS

Reconstruction prompt. The reconstruction prompt follows. `site_url` is the local reference entry URL, and `browser_tools_description` is filled with the browser-tool text below.

```
You are recreating a web front-end application from observation.
The reference is a local offline copy. Nothing is published or distributed.

The reference is available in the loopback URL {site_url}.
{browser_tools_description}

## Full-site scope
Starting from {site_url}, explore the entire site and rebuild as many pages
as you can.

- Discover pages by following internal links from the homepage until no new
pages are found; stay on the loopback origin and ignore external links.
- Rebuild every page that renders correctly, skipping only genuine HTTP
errors, empty/error shells, and redirect loops.
- Reuse shared components across pages while preserving each page's real
content and route.

Study the reference thoroughly before and as you write code. Inspect its
visual layout, colors, structure, and behavior, and keep returning to it
rather than relying on a single early look.
Capture desktop (1920x1080) and mobile (375x812) views.
Write a new, original implementation with the highest possible fidelity,
matching visible content, semantic structure, layout, spacing, colors,
typography, media, responsive behavior, navigation, and interactive states.
Author original React/TypeScript components and class names. Reproduce
visible text content, but do not copy the reference implementation.
You may copy binary media from {site_url} and its loopback origin only. Do
not access external sites or CDNs. If an exact font is unavailable locally,
use a bundled or system fallback.

Deliver:
1. Source code in your current working directory
2. A single self-contained output/index.html responsive at 1920x1080 and
375x812

Build requirements:
- Use the supplied Vite + React 19 + Tailwind CSS 4 + TypeScript scaffold.
- Build with npm install && npm run build using the scaffold's declared
dependencies.
- Copy dist/index.html to output/index.html; all media and navigation must
work and the browser console must have no errors.

Do not inspect eval_config.json, the evaluation harness, scorer, ground
truth, test files, scorer installation, or evaluation/ directory.

Original-authorship rules (a violating submission is CAPPED AT 0.10):
```

```
- Reuse binary media and individual design-token values only. Never carry
reference class names into source or data files.
- Do not copy raw HTML with innerHTML, dangerouslySetInnerHTML, or
DOMParser, and do not replay a serialized node tree through a generic
renderer.
- Do not bulk-walk or serialize the page with querySelectorAll, outerHTML,
innerHTML, getComputedStyle, or getBoundingClientRect to harvest DOM,
classes, or geometry for replay. Targeted inspection of a visible property
or design-token value is allowed.
- Do not embed the original with an iframe or use screenshots as page
backgrounds.

Web-specific implementation:
- Build shared Header, Footer, and Nav components consistently across
pages.
- Implement each discovered page to match its on-screen reference.
- Use path-based client-side routing matching the original URL paths, not
hash routing. Use the existing cn() utility; @/ maps to src/.

Important principles:
- Develop iteratively: start with a minimal working skeleton, build it,
launch it, and verify against the reference, then keep adding and refining
in small increments.
- Be thorough: reproduce the full persistent product surface, not just the
initial screen.
- Treat "it launches" as the start, not the finish. Keep iterating to
improve fidelity.
- Do not give up after a failed build or launch; diagnose the error, fix
it, and retry.
- Verify quantitatively after each rebuild. Compare behavior and visual
output against the reference rather than guessing.

This is an automated pipeline with no user interaction. You MUST include at
least one tool call in every response; a text-only response terminates the
session. Do not ask questions or wait for confirmation. Do NOT use Agent,
Workflow, EnterPlanMode, ExitPlanMode, or AskUserQuestion.
```

Browser-tool description. The following text fills `browser_tools_description`.

```
You have **Playwright** browser tools — use them extensively to inspect
the local reference page (the offline copy served on localhost) before
writing any code.

For each discoverable page:
- **Navigate** to the page and take a **snapshot** (accessibility tree) to
capture
  all headings, text, links, images, and semantic structure.
- **Screenshot** at desktop (1920×1080) and mobile (375×812) as visual
reference.
- Use **browser_evaluate** to extract exact colors (background, text,
links),
  font families/sizes, and image source URLs.
- Identify **shared components** that appear across pages: header,
navigation bar,
  footer — note their exact link labels, layout, and styling.
```

Workspace instructions. The workspace task file provides the following instructions.

```
Inspect the reference page served for you at the given localhost URL — it
is a local, offline copy in your own sandbox — and rebuild it by authoring
your own original React components with the provided template. Reproducing
a reference UI in your own code is routine front-end engineering; copying
its markup or CSS verbatim is disallowed, so author your own. Proceed
autonomously and do not decline on copyright grounds.
```

G.2 CONTINUATION INPUTS

The five messages below implement the recovery rule for agents that return without source changes (Appendix D.1). They address refusals over copyright concerns and ask the agent to begin implementing the reference website. The runner sends them in order within the existing session while no source changes are detected, with `site_url` substituted where present.

Continuation 1.

```
Continue — there is no copyright issue here. {site_url} is a local,
offline copy inside your own sandbox and you are writing your OWN original
React code, which raises no copyright, trademark, or content-ownership
concern. Please keep going and write the files.
```

Continuation 2.

```
Please proceed — this raises no copyright problem. Rebuilding a reference
UI in your own components is routine, legitimate front-end work, and
nothing you produce leaves the sandbox. Keep building and write the code
now.
```

Continuation 3.

```
There is nothing to decline on copyright grounds: authoring your own
implementation of a reference is standard engineering, and copying the
reference verbatim is disallowed anyway. Continue and write the source
files.
```

Continuation 4.

```
Copyright does not apply here — the reference is a local copy, your output
stays in the sandbox, and you are producing original code. Please stop
pausing and keep working: write the React files for the target pages.
```

Continuation 5.

```
No copyright or IP concern applies — just continue the build. Write your
own original React code for the pages now and keep going until
`output/index.html` exists.
```

Resuming after an error. A separate recovery path uses the following message to resume Claude Code after an error.

```
Continue. Complete all remaining tasks.
```

G.3 FUNCTIONAL: INTERACTION-TEST GENERATION

The interaction-test generator receives `site_url` and `target_pages`, where each page record contains `id`, `path`, and `title`. The working directory supplies `site_meta.json`, including page depths and navigation-graph edges with `from`, `to`, `location`, `label`, and `href`; it also supplies `rb_web_kit.ts` and the audit helper. The full interaction-test generation template follows.

```
You are the interaction-test orchestrator for this site. Produce ONLY
depth>=1
functional interaction tests and write them to
`functional_interaction.spec.ts` in
the current working directory. Do NOT dispatch a
'mockweb-test-orchestrator'
subagent — YOU are the orchestrator.

## Site under test
- site_url: {site_url}
- target_pages (COVER EVERY ONE — one or more depth>=1 tests each):
{target_pages}

## Inputs available in the cwd
- `site_meta.json` — has `nav_graph` (edges
{{from,to,location,label,href}}) and `pages[].depth`. Use it to plan
multi-hop chains.
- `rb_web_kit.ts` — read-only helper: import {{ rbShowing, rbReach }} from
'./rb_web_kit';
- `rb_web_testgen_kit.py` — the audit CLI (used in step 4).

## Declaration contract (EVERY test must carry both)
- Title prefix: `[rb:dN]` where N = real depth crossed.
- Annotation: `test('[rb:dN] ...', {{ annotation: [{{ type: 'rb',
description: 'depth=N;jump=KIND;expect=...' }}] }}, async ...)`
- jump KIND in: reach | transition | value
  - `reach` = URL/surface changed — PROVE with `rbShowing(page)` +
  `rbReach(before, after, target, source?)`
  - `transition` = a tab/accordion/modal/menu state flipped
  - `value` = asserted a specific expected content
- Any depth>=1 / reach MUST be backed by a real `.click()`/`.tap()` and
(for reach) an `rbReach(...)` proof.
- **rbReach proof target MUST be destination-unique**: the `target` string
passed to
  `rbReach(before, after, target)` MUST be content UNIQUE to the
  destination page —
  verify it is ABSENT from the `before` snapshot (not present on the origin
  page).
  NEVER use shared header/footer/nav/cookie-banner text that appears
  site-wide
  (e.g. footer CTAs like "Subscribe", "Get to know us") — by rbReach's
  definition
  such text can never "newly appear", and it proves nothing since every
  page has it.
  Prefer the destination's own H1 / distinctive heading / a unique body
  phrase.
  Clicking a shared nav ELEMENT is fine; just don't assert shared CONTENT
  as the
  arrival proof.

## 5-step process

### Step 1 — Recon
Read `site_meta.json` nav_graph + `pages[].depth`. Browse the target pages
via
browser_navigate / browser_snapshot to see EACH page's nav + interactive
elements
(links, buttons, menus, tabs, accordions, modals) and the multi-hop paths.
Aim to
```

```
cover EVERY target page — coverage should scale with the number of pages.

### Step 2 — Delegate
Use the Task tool to dispatch the four lane subagents, giving each the
site_url,
the relevant nav_graph edges (across ALL target pages, not a sample), and
the
declaration contract above:
- `mockweb-nav-chain` (subagent_type="mockweb-nav-chain") — multi-hop
routing, non-homepage origins
- `mockweb-menu` (subagent_type="mockweb-menu") — dropdown/mega-menu ->
nested item -> land
- `mockweb-insurface` (subagent_type="mockweb-insurface") — open
tab/accordion/modal, then act inside
- `mockweb-mobile` (subagent_type="mockweb-mobile") — 375px hamburger ->
navigate

Dispatch each lane at most ONCE.

### Step 3 — Validate (Gate 1)
Every test MUST pass on the ORIGINAL site. Run/inspect against site_url;
drop any
interaction the archived original does not actually support. Keep only
stable,
deterministic assertions.

### Step 4 — Improve & self-audit
Collect all lane tests. Run (from THIS working directory):
```
python3 rb_web_testgen_kit.py audit functional_interaction.spec.ts &&
python3 rb_web_testgen_kit.py audit_depth functional_interaction.spec.ts &&
python3 rb_web_testgen_kit.py lint functional_interaction.spec.ts
```
The audit is REPORT-ONLY — it never blocks finalize. Where a quota target
is
short, add tests from a different angle; cap at 5 angle-change attempts
total.

### Step 5 — Merge
Drop only EXACT duplicates (same origin page + same target + same claim).
DO KEEP
the same nav destination tested from DIFFERENT origin pages — on a
recreated site
the shared header/footer nav may be wired correctly on some pages and
broken on
others, so "click Docs from /pricing" and "click Docs from /blog" are
DISTINCT,
valuable facts. Do NOT dedup shared navigation across origins. Then write
ONE
`functional_interaction.spec.ts` importing `./rb_web_kit`. Each `test()`
must be
independent; no `waitForLoadState('networkidle')`; relative paths only;
semantic
selectors only. **Every `page.goto(...)` MUST pass `{{ waitUntil:
'domcontentloaded' }}`**
— the target runs against an OFFLINE archive whose `load` event may never
fire, so a
bare `goto()` (default `waitUntil:'load'`) HANGS the test until timeout.
Always
`await page.goto(path, {{ waitUntil: 'domcontentloaded' }})`.
```

```
## Coverage target (this is the goal)
- **Cover EVERY target page with >= 1 depth>=1 test, AND produce >= 80
tests total.**
  Aim for roughly >= 2-3 honest depth>=1 tests PER target page (per-origin
  nav reaches
  + that page's own drill-downs). If pages are few, add more distinct
  interactions per
  page (and depth>=2 chains) to reach 80. depth>=2 should be >= 10% of
  interactive.
- **Interaction is NOT only link clicks.** Any effect invisible at depth-0
counts —
  use the full vocabulary the archive actually supports: link click
  (reach); `<select>`
  dropdown `selectOption` (reach/transition); button-opened menu -> nested
  item (reach);
  hover mega-menu -> item; tab / accordion / disclosure expand
  (transition); toggle /
  switch / checkbox (transition); carousel/slider next-prev; client-side
  filter/sort;
  "load more"/pagination; in-page search box. Prove reach with `rbReach`;
  prove a
  transition by asserting the flipped panel's unique content appeared.
- Every test stays HONEST: real `.click()`/`.tap()` + `rbReach` proof, and
must pass
  on the ORIGINAL site. Never pad with fakes — `rb_audit_depth` flags
  unbacked depth.
- Dispatch each lane at most ONCE, but each lane covers ALL relevant target
pages
  (not a sample).

Write the file with the Write tool. Do not print it instead of writing it.
```

Output format and related templates. The output is `functional_interaction.spec.ts`. Each test carries a `[rb:dN]` title prefix and an `rb` annotation with the form `depth=N;jump=KIND;expect=TARGET`, where `KIND` is one of `reach`, `transition`, or `value`.

The released prompt collection also includes the template that generates Functional, legacy Visual, Quality, and Structural test files, and instructions for four interaction groups within Functional: `mockweb-nav-chain`, `mockweb-menu`, `mockweb-insurface`, and `mockweb-mobile`.

G.4 VISUAL-ASSERTION GENERATION

For the Visual dimension, the generator reads reference screenshots and creates the assertion lists before reconstruction. The complete generation prompt follows.

```
You are a visual QA engineer. You will be shown a screenshot of a webpage.
Generate ATOMIC visual assertions that an automated judge will check BINARY
(true/false) to verify a replica reproduces the LOOK of this page.

CRITICAL — focus on appearance, not wording:
The page's TEXT CONTENT is already verified by other test dimensions
(DOM/text similarity and functional tests). Do NOT re-verify wording.
Describe HOW things LOOK and WHERE they are, not WHAT they SAY.

CRITICAL — describe what is VISIBLE, never what you recognise:
```

```
Logos, icons and illustrations in these pages may have been REPLACED with
different artwork than the brand you know. The page's text naming a product
tells you NOTHING about what the mark on screen looks like. Describe the
shapes and colours actually present — 'a circular emblem of concentric
orange and red arcs', never 'the Rust logo with its gear and letter R'.
Writing the logo you remember instead of the pixels in front of you
produces an assertion that is FALSE of this screenshot, which is worse than
writing nothing. Name a company or product ONLY when its name is legible as
text in the screenshot.
This applies HARDEST to mascots. If the page belongs to a project whose
logo you remember as an animal or character — an elephant, a penguin, a
crab, a gopher, a chameleon, a robot — that animal is NOT on this page: it
has been replaced by an abstract geometric mark. Do not write 'an elephant
head', 'a penguin', 'a gear with a letter R'. Look at the actual shape and
say what it is: concentric circles, interlocking arcs, stacked facets, a
chevron row.

CRITICAL — every assertion MUST be DISCRIMINATIVE:
Each assertion must be specific enough to be FALSE on a
differently-designed page, a placeholder, or a generic template. BAN
vacuous facts that hold on almost any page — e.g. 'the content is
centered', 'there is empty space above the content', 'the header spans the
full width', 'a heading is present', 'a paragraph appears below the
heading', 'a section exists'. If a bland placeholder page would also
satisfy it, DO NOT write it. Capture what makes THIS page look distinctive.

Rules:
- Each assertion checks exactly ONE single visual fact, judged cleanly
true/false. Never bundle two facts (split 'a black square logo on the left'
into a position assertion and a color assertion).
- MANDATORY position: every assertion must state WHERE on the page the
element sits — vertical region (top/upper/middle/lower/bottom) AND
horizontal (left/center/right). Use text only as an anchor to name the
element.
- COLOR IS MANDATORY AND SPECIFIC: always include assertions for the
OVERALL PAGE background color and EACH major section's background color,
naming a specific, comparable shade (e.g. 'the entire page background is
near-black (#0a0a0a)', 'the hero background is pure white', 'the footer
background is deep navy') — never just 'dark' or 'light'. A wrong
background color (e.g. white where the original is black) must be catchable
from your assertions.
- HERO / HEADINGS ARE MANDATORY AND SPECIFIC: for the MAIN/HERO headline
and each major section heading, write SEPARATE atomic assertions pinning
(a) its SIZE relative to the page (e.g. 'very large, roughly a quarter of
the viewport height' or 'far larger than the body text'), (b) its exact
COLOR including any gradient/multicolor treatment (e.g. 'white with one
word in an orange-to-purple gradient'), and (c) its FONT STYLE (serif /
sans-serif / monospace). Describe them specifically enough that a smaller,
plainer, or differently-colored heading (or a placeholder) FAILS the
assertion.
- The single fact is one of: page-position/arrangement of an element or
section; a SPECIFIC named color of a background / text / accent; whether a
KEY image/logo/illustration renders as the REAL DISTINCTIVE graphic,
described concretely (e.g. 'a colorful serif wordmark', 'a photo of two
people in a lab', 'a dot-matrix pattern') — not just 'an image'; a
type-hierarchy fact (specific relative size/weight/font-style, e.g. 'large
serif' vs 'sans-serif').
```

```
- Capture the page's SIGNATURE look: its distinctive hero, unique sections,
specific color scheme and characteristic imagery — described specifically
enough that a wrong or placeholder rendering would FAIL the assertion.
- Do NOT exhaustively enumerate every attribute. Pick the 1-2 MOST
DISCRIMINATIVE facts per element/section. Prioritize, in order: (1) overall
page + section background colors, (2) the distinctive hero / main-content
look, (3) position & arrangement of each MAJOR section, (4) whether each
KEY image/logo renders as the real graphic, (5) key type hierarchy.
- Do NOT enumerate navigation items, list link labels, or quote
headings/paragraphs — content is covered elsewhere.
- The site header, top navigation, search bar, and footer are SHARED CHROME
repeated on every page and trivially reproduced — spend AT MOST 1-2
assertions total on all of them combined. Devote the rest of the budget to
THIS page's DISTINCTIVE main content: its unique sections, layout, images,
and colors.
- Generate around 40 atomic assertions per page.

Return ONLY a JSON array of assertion strings, no markdown fences, no extra
text.
Example: ["The entire page background is near-black (#0a0a0a).", "The hero
headline in the vertical center is a large serif font with one word
rendered in a multicolor gradient.", "A light-beige CTA button with dark
text sits centered directly below the hero.", "The feature cards form a
3-column grid on a white background in the middle of the page.", "The org
logo in the top-left renders as a real colorful wordmark (not letters).",
"A dot-matrix pattern decorates the lower-right corner over the dark
background."]
```

Request text and images. The user text is constructed as follows, with the reference title substituted for `page_title`.

```
This is a screenshot of the page titled "{page_title}". Generate visual
assertions for this page.
```

When an image is encoded as multiple tiles, the following suffix is appended, with the number of tiles substituted for `n`:

```
 NOTE: this page is tall, so it is provided as {n} vertical sections
 ordered top-to-bottom; treat them as one continuous page.
```

The generator and judge process screenshots in the same way. Screenshots within the 8,000-pixel dimension limit are supplied as one image. Taller captures are divided into ordered strips of up to 4,000 pixels, with at most 16 strips per request. Pages above 64,000 pixels are proportionally resized before tiling so that the full page remains represented. Images may be compressed to fit the request budget. All strips form evidence for one continuous page and one assertion list.

Output format. The requested response is a JSON array of assertion strings. The generator stores each list under a page-and-viewport key, using the following schema:

```
{
  "{page_id}/desktop.png": ["<visual assertion>", "<visual assertion>"],
  "{page_id}/mobile.png": ["<visual assertion>", "<visual assertion>"]
}
```

G.5 VISUAL JUDGE

For Visual scoring, the judge receives candidate screenshots together with the fixed assertion list generated from the reference, as in §4. The full assertion-judge system prompt follows.

```
You are a STRICT visual fidelity judge for webpages. You will be shown
screenshot(s) and a list of check items to evaluate.

If a single page is provided as several stacked images, treat them as
top-to-bottom sections of one continuous page.

Each item is ONE single visual fact. Judge it BINARY:
- pass=true  ONLY if the fact is fully true in the replica with NO
discrepancy.
- pass=false if ANYTHING the item states is wrong or absent — wrong color,
wrong BACKGROUND color, wrong position/placement, wrong arrangement,
missing or fake graphic, or a GENERIC/PLACEHOLDER rendering that lacks the
specific described element. If your reason notes any mismatch, you MUST set
pass=false.

BE STRICT ABOUT COLOR: a different background or element color (e.g. a
white background where the item says black/dark, or a plain colored box
where a distinctive graphic is described) is a FAIL, even if the
position/layout is similar.

The ONLY thing you may ignore is a differing literal
number/count/price/date/follower-or-download STATISTIC — these are mock
data, so a value of the correct TYPE and FORMAT in the right place passes
(e.g. $20 vs $50, '54 likes' vs '234 likes'). This single relaxation
applies ONLY to such literal numeric/date values; it does NOT relax color,
background, layout, arrangement, type-hierarchy, or whether the real
distinctive graphic/logo/image is present. A logo or image replaced by
substitute LETTERS, initials, a plain colored box, alt-text, or a
placeholder FAILS the item.

Return ONLY a JSON array with one object per item, in order:
[{"pass": true, "reason": "brief"}, {"pass": false, "reason": "brief"},
...]

No markdown fences, no extra text.
```

User prefix and assertion list. The user prefix is reproduced below. The request then appends `Assertions to evaluate:` and the complete list, numbered from one in its stored order.

```
Below is a screenshot of a webpage. Evaluate each assertion about what
should be visible.


Assertions to evaluate:
1. <first frozen assertion>
2. <second frozen assertion>
...
```

Verdict format. The requested output is one object per assertion, in the same order:

```
[
  {"pass": true, "reason": "brief explanation"},
  {"pass": false, "reason": "brief explanation"}
]
```