WEBCRAFTBENCH: EVALUATING WEB APPLICATION GENERATION FROM A SOFTWARE TESTING PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Human evaluation provides a direct measure of the quality of LLM-generated web applications. However, fitting human judgments through automated evaluation remains challenging. Static benchmarks can credit functionality that exists in source code but is unreachable at runtime. Interactive benchmarks exercise the application, yet incomplete exploration can cause them to miss implemented functionality and confound application defects with agent execution failures. To address these limitations, we propose WebCraftBench, an interactive benchmark for evaluating web application generation from a software testing perspective. WebCraftBench instruments each generated application and uses code coverage to guide an agent in exploring its functionality through user-simulated interactions. It then abstracts the interaction trace into a state-transition graph and evaluates the application along three dimensions: visual aesthetics, usability, and requirement alignment. By separating exploration from scoring, WebCraftBench collects runtime evidence without constraining exploration to predefined acceptance criteria. WebCraftBench comprises 369 real-world user requirements and 5,088 acceptance criteria. Evaluation of 17 frontier LLMs reveals distinct strengths across the three dimensions, with no model leading on every dimension. On 197 validated sessions sampled from an internal arena, WebCraftBench achieves 85.3% agreement with human preferences, with agreement generally increasing as the score difference between paired applications grows. Further experiments show that coverage guidance improves exploration coverage and the model rankings remain stable when the judge model is replaced.

1 INTRODUCTION

Web applications run across platforms without installation, making them an accessible medium for LLM agents to deliver interactive functionality to users. Given a natural-language requirement, an agent iteratively invokes tools and generates the source code of a runnable application. However, the evaluation of generated applications is typically highly subjective. Human-preference leaderboards such as Code Arena (LMArena.ai, 2026) obtain user preferences directly by asking annotators to choose between two applications generated by different models for the same requirement. However, repeated human evaluation is costly, requiring an automated benchmark to reduce the cost of evaluation while preserving user preferences.

Existing automated benchmarks face two major limitations, making them less likely to align with user preferences. **First, static benchmarks do not exercise the generated application.** Static benchmarks based on source code or a single rendered screenshot (Sun et al., 2025; Liu et al., 2026a) assess the applications without verifying whether users can reach and use the intended functionalities. For example, a broken button on the entry page can block all downstream functionalities, while the corresponding code may still be present, causing the benchmark to credit functionalities that are unusable in practice. **Second, interactive benchmarks struggle to distinguish application defects from agent execution failures.** Interactive benchmarks define rubrics, reference answers, or contracts and ask an agent to validate them through interactions (Lu et al., 2025; Wu et al., 2025; Zhang et al., 2026b;a; Meng et al., 2026; Lei et al., 2026). However, a failed check may indicate that the application lacks the target functionality or that the agent failed to locate or trigger it. Errors in

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.

GUI interaction and evidence interpretation make these causes difficult to distinguish, introducing uncertainty into the final score.

To address these limitations, in this paper, we propose WebCraftBench, an interactive benchmark for evaluating web application generation from a software testing perspective. WebCraftBench treats each generated application as software under test: it simulates user interactions and explores the application to collect runtime evidence. Code coverage serves two purposes in this process. It guides the exploration toward unexecuted code and quantifies how much of the implementation has been exercised. WebCraftBench separates exploration from scoring so that the exploration process is not constrained by acceptance criteria.

WebCraftBench consists of four stages. *(i) Instrumentation.* An automated tool instruments the JavaScript code in each generated application to collect runtime code coverage without manual adaptation. *(ii) Coverage-guided exploration.* An LLM-driven agent explores the application as a software tester. When coverage plateaus, a separate LLM analyzes the uncovered code and provides natural-language guidance for subsequent exploration. This feedback supplements the agent's own assessment of what remains to be tested. *(iii) State abstraction.* A deterministic state abstraction approach processes each page and merges pages with equivalent normalized representations into the same state, generating a compact state-transition graph to represent the exploration trace. *(iv) Scoring.* LLM and agentic judges assess visual aesthetics, usability, and requirement alignment using the state-transition graph and the recorded traces, DOM snapshots, and screenshots.

We evaluate 17 frontier LLMs on WebCraftBench and validate the benchmark against human preferences from pairwise comparison samples. WebCraftBench achieves 85.3% pairwise agreement on 197 validated samples. The model-level results reveal complementary strengths in visual aesthetics, usability, and requirement alignment. Additional experiments show that coverage guidance improves exploration coverage and that model rankings remain stable across judge models.

In summary, this paper makes the following main contributions:

- We construct WebCraftBench, a benchmark with 369 real-world user requirements and 5,088 acceptance criteria for evaluating web application generation.
- We design a four-stage evaluation pipeline that separates exploration from scoring, guides exploration with code coverage, and organizes runtime evidence through state abstraction.
- We evaluate 17 frontier LLMs across visual aesthetics, usability, and requirement alignment. Validation shows 85.3% agreement with human preferences, stable rankings across judge models, reliable instrumentation, and improved exploration coverage.

2 Related Work

2.1 Benchmarks for Web Application Generation

Existing related benchmarks are shown in Table 1. Early static benchmarks formulate web generation as reproducing a reference design and measure pixel-, text-, or element-level similarity to a reference screenshot or implementation (Beltramelli, 2018; Bhathal & Gupta, 2025; Vu et al., 2025; Sun et al., 2025). Real-world requirements, however, often admit multiple valid implementations. WebCoderBench (Liu et al., 2026a) assesses source code and initial renderings without a unique reference implementation. These benchmarks support inexpensive, reproducible evaluation, but do not exercise the application and may credit functionality that is unreachable at runtime.

Interactive benchmarks execute applications in a browser, and score them using test suites, fixed action sequences, or staged screenshots (Xu et al., 2025; Zhu et al., 2025; Zhang et al., 2025), or ask agents to validate checklists, reference answers, contracts, or specifications (Lu et al., 2025; Zhang et al., 2026b; Meng et al., 2026; Zhang et al., 2026a; Wu et al., 2025; Lei et al., 2026). These benchmarks observe a broader range of runtime behavior and provide evidence for failure diagnosis. Predefined targets, nevertheless, constrain the verification scope, leaving behavior outside those targets unassessed. Failed agent-based checks can also reflect either application defects or agent failures.

More recently, Cookie-Bench (Yang et al., 2026) removes checklists, allows the agent to plan interactions from the user request, and scores applications after evidence collection. I-

Table 1: Comparison of WebCraftBench with existing benchmarks for web application generation.

Benchmark	Samples	Interactive	Exploration	Judgment			Exploration–Scoring Separation	Exploration Quantification
				Rules	LLM	Manual		
FullFront (Sun et al., 2025)	400	×	None	✓	✓	×	×	×
WebCoderBench (Liu et al., 2026a)	1,572	×	None	✓	✓	×	×	×
FrontendBench (Zhu et al., 2025)	148	✓	Scripted	✓	×	×	×	×
Web-Bench (Xu et al., 2025)	1,000	✓	Scripted	✓	×	×	×	×
ArtifactsBench (Zhang et al., 2025)	1,825	✓	Scripted	×	✓	×	✓	×
WebDev Arena (LMArena.ai, 2026)	–	✓	Human	×	×	✓	×	×
Design Arena (Design Arena, 2026)	–	✓	Human	×	×	✓	×	×
WebGen-Bench (Lu et al., 2025)	101	✓	Checklist-guided	×	✓	×	×	×
FronTalk (Wu et al., 2025)	100	✓	Checklist/autonomous	×	✓	×	×	×
MiniAppBench (Zhang et al., 2026b)	500	✓	Checklist-guided	×	✓	×	×	×
WebCompass (Lei et al., 2026)	1,526	✓	Checklist-guided	×	✓	×	×	×
WebGameBench (Zhang et al., 2026a)	111	✓	Specification-guided	×	✓	×	×	×
WebRISE (Meng et al., 2026)	442	✓	Contract-guided	✓	✓	×	×	×
I-WebGenBench (Dai et al., 2026)	201	✓	Enumerative	✓	✓	×	△	×
Cookie-Bench (Yang et al., 2026)	1,000	✓	Model-autonomous	×	✓	×	✓	×
WebCraftBench (ours)	369	✓	Coverage-guided	×	✓	×	✓	✓

Note: △ indicates partial satisfaction.

WebGenBench (Dai et al., 2026) applies standardized actions to enumerated components and records DOM changes. However, neither explicitly measures exploration adequacy. WebCraftBench introduces code coverage measurement into the exploration process to quantify the functionalities exercised and guide further exploration, while withholding acceptance criteria from the exploration agent.

2.2 Automated GUI Testing

Automated GUI testing exercises applications through GUI actions to gain coverage and reveal failures within a specific time budget (Liu et al., 2026c). Existing work studies state abstraction approaches at different granularities (Yandrapally & Mesbah, 2022; Liu et al., 2025b; 2026b) to avoid repeated exploration, while using random-based (Google, 2025), model-based (Mesbah et al., 2012), reinforcement-learning-based (Zheng et al., 2021), and LLM-based (Liu et al., 2024; 2025a) exploration strategies.

WebCraftBench adapts these approaches to evaluating LLM-generated web applications. Code coverage provides exploration feedback and quantifies the implementation exercised, leveraging the ideology of LLM-based GUI testing approaches. State abstraction groups near-duplicate pages (Yandrapally et al., 2020) into a compact state-transition graph, as part of the evidence for scoring.

3 Dataset

WebCraftBench contains 369 real-world user requirements, each paired with a checklist used exclusively for scoring.

3.1 Data Source

We collect our user requirements from production traffic on an enterprise-internal replica of Code Arena (LMArena.ai, 2026). These requirements retain characteristics of real-world usage that are difficult to reproduce synthetically, including colloquial expressions, incomplete specifications, and substantial variation in the expected scope of the application. Following WebCoderBench (Liu et al., 2026a), we first anonymize the requirements. Three annotators then review each requirement and use majority voting to exclude samples that are incomprehensible, depend on unavailable materials, or fall outside the web application scope. We remove lexical duplicates using MinHash (Broder, 1997) and semantic duplicates using MiniLM (Wang et al., 2020). The resulting dataset contains 369 text-only requirements. Our requirements have a mean length of 601 characters, a median of 105, and a range of 7–37,903. This distribution captures the coexistence of brief requests and detailed specifications in production traffic.

Table 2: Dataset statistics for WebCraftBench.

Artifact Complexity			**Expression Style**			**Application Category**	
$L1$	Highly Simple	48	$S1$	Technical	81	Entertainment	106
$L2$	Simple	161	$S2$	Colloquial	222	Scientific Demo	37
$L3$	Medium	120	$S3$	Role-playing	15	Online Office Suite	33
$L4$	Complex	27	$S4$	Analogy	51	Utility Website	29
$L5$	Highly Complex	13				Data Visualization	21
Requirement Clarity			**Acceptance Criteria**			Online Education	18
$C1$	Clear	135	Functional		2,706	AI-powered	15
$C2$	Intermediate	134	Content		1,447	Multimedia	15
$C3$	Vague	100	Visual		935	*12 others*	95

3.2 Requirement Characterization

Following the annotation protocol of WebCoderBench (Liu et al., 2026a), we characterize each requirement along four dimensions: artifact complexity, expression style, requirement clarity, and application category. An LLM assigns initial labels and provides a rationale for each, after which human annotators review and revise the annotations. Table 2 summarizes the results. Overall, 63.4% of requirements are not fully specified ($C2$ or $C3$), and 60.2% use colloquial language ($S2$). These characteristics require LLMs to interpret user intent from incomplete or informal descriptions. Entertainment and scientific demonstrations are the two largest application categories.

3.3 Acceptance Criteria

Real-world user requirements often admit multiple valid implementations. We therefore associate each task with a checklist, including three sets of *acceptance criteria*, with each acceptance criterion expressing a concise, high-level property that should be observable in the delivered application. The three sets of criteria correspond to three dimensions: **functional**, including features and interactions; **content**, including text, data, and specified resources; and **visual**, including layout, style, and required components. These dimensions jointly determine the requirement-alignment score (detailed in §4.4). Following WebCoderBench (Liu et al., 2026a), we use Claude-Opus-4.8, GPT-5.6-Sol, and Grok-4.5 to generate checklists independently. Claude-Opus-4.8 merges the checklists following the majority voting process, and human experts validate the resulting set. The checklists describe the required observable properties without prescribing implementation details or introducing requirements absent from the original request. The 369 requirements yield 5,088 acceptance criteria: 2,706 functional, 1,447 content, and 935 visual. Each task has 13.8 criteria on average, ranging from 2 to 77 according to the detail of the requirement.

4 Evaluation Process

The evaluation workflow of WebCraftBench contains four stages, which are instrumentation, exploration, state abstraction, and scoring (shown in Figure 1). The first three stages collect and organize runtime evidence without access to acceptance criteria, while the scoring stage leverages these criteria for a fair judgment.

4.1 Instrumentation

In order to obtain coverage from various kinds of generated applications, we develop an automated instrumentation tool based on Istanbul (Istanbul Contributors, 2024), a mature and famous instrumentation tool for JavaScript. Our tool supports plain static HTML, Vite, Next, Create React App (CRA), and Astro, with framework-specific procedures applied automatically. With an application as input, our tool first detects its framework according to configuration files. For static frameworks, it runs Istanbul locally to instrument the application; for frameworks that require a build, it injects a plugin during the build and performs instrumentation after the build. After instrumentation, each JavaScript statement is associated with a counter that records every execution of the statement at runtime. During exploration, after each interaction, cumulative coverage and per-step gains can be

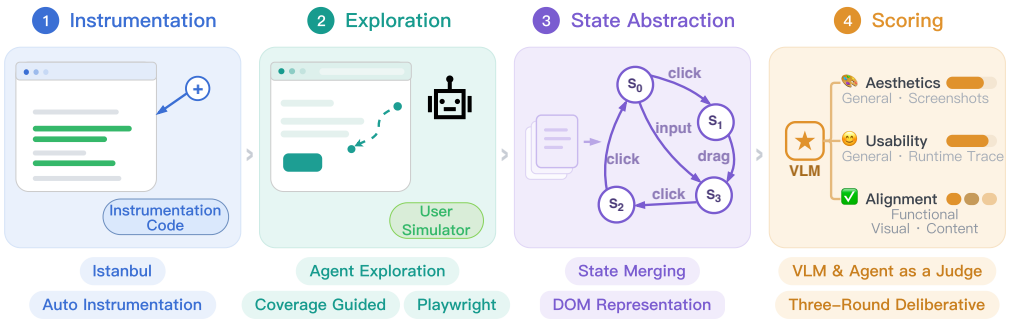


Figure 1: The evaluation workflow of WebCraftBench.

calculated from the number of covered statement counters and the total number of counters. If instrumentation fails, the application remains in the evaluation and undergoes exploration and scoring in black-box mode. Appendix A.1 describes framework support and deployment procedures; §5.5 evaluates instrumentation reliability. The instrumented application is then deployed in a sandbox on a remote server and is accessible through a URL.

4.2 Coverage-Guided Exploration

In order to flexibly control the agent's context during exploration, we develop an agent based on the AWorld (Yu et al., 2025) framework, interacting with the web applications using Playwright MCP (Microsoft, 2025). The agent reads a simplified DOM tree of the current page at each step to understand its content and retrieves only the tree diff between steps to save context space. The agent is prompted as a software tester to discover available functionalities and test boundary conditions. The agent has no access to source code, acceptance criteria, or reference answers.

During exploration, the harness monitors the coverage and detects a coverage plateau after three consecutive interactions without a coverage increase. It then extracts the uncovered code and invokes a separate LLM to produce natural-language suggestions. These suggestions are injected into the exploration agent's context to guide further testing. Exploration ends when the agent declares testing complete or exhausts a 100-step budget. To conclude, in its final step, the agent summarizes the functionality, interactions, and observed failures. Coverage thus both quantifies code execution and directs subsequent exploration; §5.6 evaluates its contribution.

4.3 State Abstraction

Exploration traces, screenshots, and DOM snapshots can easily exceed the judge's context window and obscure relevant evidence through repetition. We adopt the state abstraction practice from the area of automated GUI testing (Mesbah et al., 2012; Yandrapally & Mesbah, 2022) to organize observations into a compact state-transition graph. Each snapshot is normalized by removing script and style tags and mutable component attributes, and replacing timestamps and clock strings with fixed placeholders. Pages with identical normalized representations form one state, and user actions define transitions. Although simple and heuristic, this approach is demonstrated to be effective (Liu et al., 2025b). The constructed state-transition graph provides a straightforward and concise understanding of the application, serving as part of the evidence for scoring.

4.4 Scoring

WebCraftBench scores visual aesthetics, usability, and requirement alignment separately on a 0–100 scale. Appendix A.2 provides procedural details.

Visual aesthetics. This dimension assesses up to five representative screenshots collected during exploration. We filter blank and partially loaded frames using foreground and edge density, then

apply farthest-point sampling to select visually diverse screenshots. The judges examine layout, information organization, typography, graphic quality, and polish.

Usability. This dimension assesses the experience of interacting with the application using the complete exploration trace and constructed state-transition graph. The judges examine successfully executed interactions and observed failures, including problems with feedback, navigation, state recovery, and fault tolerance.

Both dimensions use a three-round deliberative protocol: an advocate identifies strengths, a critic identifies defects, and a final judge verifies both assessments against the supplied evidence before scoring. The advocate and critic cite concrete observations without assigning scores. The judge adds seven dimension-specific adjustments to a 0–5 base score and clips the result to $[0, 5]$. Appendix A.2 specifies these aspects. To reduce sampling variance, the final judge runs independently five times with the first two assessments held fixed. Given scores $q_1, \ldots, q_5 \in [0, 5]$, we discard the highest and lowest scores and map the remaining mean to 0–100:

$$s = 20 \times \frac{\sum_{i=1}^{5} q_i - \max_i q_i - \min_i q_i}{3}.$$

This protocol aims to improve score discrimination and reduce concentration at the scale endpoints; Appendix D compares it with single-round scoring.

Requirement alignment. This dimension judges the application behavior against the acceptance criterion. Since the complete set of evidence can overwhelm an LLM, we build a judging agent for this dimension. This agent starts with the state-transition graph to form an overall understanding of the application. It then carefully examines and searches through recorded evidence using a tool set, which supports regular-expression searches over the DOMs and traces with line-numbered results, paginated reading of DOMs and traces, screenshot inspection, and state-transition graph inspection. This retrieval process allows the agent to examine relevant evidence without loading the entire record into one context window. Once this agent determines that it has collected enough evidence for scoring, it gives a binary judgment on whether each acceptance criterion is implemented. Every judgment must cite runtime evidence, such as a trace step or a state's DOM snapshot and screenshot. Each alignment sub-dimension is scored as 100 times the fraction of its criteria judged to be implemented.

Overall score. Because scores differ in discriminative power across dimensions, we compute z-scores for each dimension to aggregate them while preventing any one dimension from dominating the overall score. Specifically, for model m and dimension j, let $\bar{x}_{m,j}$ be the mean raw score over generated applications. We compute

$$z_{m,j} = \frac{\bar{x}_{m,j} - \mu_j}{\sigma_j},$$

where μ_j and σ_j are the mean and population standard deviation of these model-level means across the N evaluated LLMs. The mean of the functional, content, and visual alignment z-scores gives requirement alignment. Averaging requirement alignment, aesthetics, and usability yields the overall score, with weights of $1/3$ for aesthetics and usability and $1/9$ for each alignment sub-dimension. Scores express relative performance within the evaluated model pool.

5 Evaluation Results

Our evaluation aims to answer five research questions.

5.1 Evaluation Setup

We evaluate 17 frontier LLMs on all 369 real-world user requirements. Each model generates one application per requirement through an agentic coding framework, yielding $17 \times 369 = 6{,}273$ applications and reflecting practical workflows for web application generation. To make the fullest possible use of model capabilities and reflect practical usage, we integrate the GPT-family LLMs with Codex, and the remaining LLMs with Claude Code. Table 3 lists the reasoning-effort settings; other parameters retain the provider defaults of each model. All applications undergo the same four-stage evaluation pipeline (§4), without model-specific adjustments. Unless otherwise stated, Claude-Opus-4.8 performs both exploration and scoring.

Table 3: Results of 17 frontier LLMs on WebCraftBench.

				General		Requirement Alignment				
#	Model	Har.	Eff.	Aesth.	Usab.	Func.	Cont.	Vis.	Align.	Total
1	Claude-Opus-5	CC	xhigh	+1.428	+1.757	+1.093	+0.780	+0.509	+0.794	**+1.326**
2	GPT-5.6-Sol	CX	xhigh	+2.239	+0.854	+0.612	+0.092	−0.294	+0.137	+1.077
3	Qwen3.8-Max	CC	enabled	+0.792	+1.088	+1.008	+1.049	+1.035	+1.031	+0.970
4	Kimi-K3	CC	max	+0.591	+1.027	+1.139	+1.252	+1.460	+1.283	+0.967
5	Hy4 preview	CC	high	+0.505	+1.099	+0.937	+1.313	+1.134	+1.128	+0.911
6	Claude-Opus-4.8	CC	xhigh	+0.158	+0.848	+1.405	+1.107	+1.265	+1.259	+0.755
7	Grok-4.5	CC	high	+0.527	+0.902	+0.923	+0.648	+0.890	+0.820	+0.750
8	GPT-5.5	CX	xhigh	+0.729	+0.225	+0.318	+0.518	+0.513	+0.450	+0.468
9	Claude-Opus-4.7	CC	xhigh	−0.157	−0.442	−0.251	−0.163	−0.149	−0.188	−0.262
10	GLM-5.2	CC	max	−0.258	−0.552	−0.443	+0.122	−0.081	−0.134	−0.315
11	DeepSeek-V4-Flash	CC	max	−0.312	−0.322	−0.479	−0.374	−0.369	−0.407	−0.347
12	Hy3	CC	high	−0.665	−0.519	−0.673	−0.725	−0.379	−0.592	−0.592
13	Qwen3.7-Max	CC	enabled	−1.014	−0.976	−0.705	−1.052	−0.435	−0.731	−0.907
14	GLM-5.1	CC	enabled	−1.145	−1.022	−0.597	−0.655	−0.733	−0.662	−0.943
15	DeepSeek-V4-Pro-Prev	CC	max	−1.453	−1.348	−0.786	−0.576	−0.391	−0.584	−1.128
16	Kimi-K2.7-Code	CC	enabled	−1.642	−1.176	−1.129	−0.685	−1.529	−1.114	−1.311
17	MiniMax-M3	CC	adaptive	−0.323	−1.443	−2.371	−2.652	−2.447	−2.490	−1.419

Note: Har.: coding harness (CX: Codex; CC: Claude Code). Eff.: reasoning effort.

5.2 RQ1: Main Results

Table 3 reports dimension-level z-scores and the aggregated main scores across 17 LLMs. Positive scores indicate performance above the mean value of the model pool (§4.4). Appendix E reports detailed raw scores.

No model leads on every dimension. Claude-Opus-5 ranks first overall and leads in usability ($+1.757$); GPT-5.6-Sol leads in aesthetics by a large margin ($+2.239$), while Kimi-K3 leads in requirement alignment ($+1.283$). These complementary strengths would be obscured by an overall ranking alone.

Visual aesthetics and usability capture distinct aspects of application quality. Their correlation is strong at the model level ($r = 0.85$) but only 0.36 at the application level. GPT-5.6-Sol ranks first in aesthetics and ranks sixth in usability; MiniMax-M3 has near-average aesthetics (-0.323) but the lowest usability (-1.443). Visual polish therefore does not establish reliable interaction, supporting runtime evaluation alongside visual assessment.

Model generations differ under the tested configurations. Within each family represented by multiple generations, the newer generation has a higher overall score, indicating steady progress in the capabilities of each model family.

5.3 RQ2: Human Preference Alignment

We sample 200 sessions from our internal replica of Code Arena (LMArena.ai, 2026), independently of the 369 benchmark tasks. Among these sessions, 197 retain valid application pairs. Each session records a human vote (preference) between two applications for the same requirement. In order to eliminate irresponsible votes, six additional annotators review and fix every vote, and after that, two others further review a random 20% subset independently. The validated votes serve as ground truth labels.

To obtain WebCraftBench's automated votes, for application a and component j, we compute $z_{a,j} = (x_{a,j} - \mu_j^H)/\sigma_j^H$ from its raw score $x_{a,j}$, using the component mean μ_j^H and standard deviation σ_j^H over all applications in this human-agreement experiment. We aggregate these z-scores using the same process as in §4.4 and compare the resulting pairwise ordering with human preferences. We define Δz as the absolute difference between paired overall scores and report within-band agreement and cumulative agreement above each threshold.

Table 4 shows that WebCraftBench aligns with human preferences on 168 of 197 pairs, yielding a satisfactory overall agreement rate of 85.3%.

Table 4: Agreement between WebCraftBench and human pairwise preferences.

	Within band			**Cumulative** ($\Delta z \geq$ lower bound)		
Δz **band**	n	**Agree**	**Rate**	n	**Agree**	**Rate**
$[0,\ 0.25)$	65	48	73.8%	197	168	**85.3%**
$[0.25,\ 0.5)$	52	45	86.5%	132	120	90.9%
$[0.5,\ 0.75)$	30	26	86.7%	80	75	93.8%
$[0.75,\ 1.0)$	13	13	100.0%	50	49	98.0%
$[1.0,\ 1.5)$	17	16	94.1%	37	36	97.3%
$[1.5,\ 2.0)$	7	7	100.0%	20	20	**100.0%**
$[2.0,\ \infty)$	13	13	100.0%	13	13	100.0%

Disagreements are concentrated among pairs with small score differences. Of the 29 disagreements, 17 occur in the $[0, 0.25)$ band and 7 in $[0.25, 0.5)$. Cumulative agreement exceeds 90% for pairs with $\Delta z \geq 0.25$. Larger score differences are generally associated with higher agreement. Among the 50 pairs with $\Delta z \geq 0.75$, the only disagreement involves a tradeoff between visual quality and functional reliability: one application has a polished interface but crashes on its primary functional path, while the other has a less polished interface and working core functionality. WebCraftBench prefers the latter, whereas the human rater prefers the former. Appendix C presents the applications and examines this disagreement.

5.4 RQ3: Judge Robustness

To test judge robustness, we replace Claude-Opus-4.8 with Gemini-3.7-Flash, a lower-cost model from a different provider, while holding tasks, applications, exploration traces, procedures, and prompts fixed. Overall scores have a Pearson correlation (Pearson, 1895) of $r = 0.983$ and a Spearman rank correlation (Spearman, 1987) of $\rho = 0.980$. Of the $\binom{17}{2} = 136$ model pairs, 130 retain their order (95.6% agreement); three of the six reversals have main-judge score gaps below 0.04. Rankings remain stable despite changes in raw-score distributions: mean usability rises from 60.1 to 75.7, while the aesthetics range expands from 26.8 to 44.2 points. These shifts motivate standardization before aggregation. Appendix B reports the full comparison.

5.5 RQ4: Instrumentation Reliability

We evaluate instrumentation applicability across all 6,273 applications in the main experiment. Instrumentation fails for only 7 applications, yielding a success rate of 99.89%. The failures have identifiable engineering causes: 4 applications use JSX input forms unsupported by Istanbul, 2 use npm-workspaces monorepos for which the tool cannot locate the Vite project root, and 1 exceeds the sandbox deployment-size limit after instrumentation. We separately check deployment regressions on the 360 deployable applications generated by Claude-Opus-5. All remain deployable after instrumentation. This check establishes deployment reliability on the tested subset. Under the fallback procedure in §4.1, the 7 applications for which instrumentation fails still undergo exploration and scoring in black-box mode, without coverage guidance.

5.6 RQ5: Coverage Improvement

Code coverage measures the proportion of the implementation exercised during exploration and helps identify functionality that may remain unobserved. We assess the contribution of coverage guidance by comparing exploration with and without this feedback. To control experimental scale, this ablation uses the 360 deployable applications generated by Claude-Opus-5. The remaining 9 applications fail for generation-side reasons, including missing files, deployment errors, and incorrect port declarations.

Figure 2 compares the coverage distributions under the two settings. Coverage guidance increases median function coverage from 91.9% to 94.3%. The number of tasks below 90% coverage decreases from 139 (39%) to 99 (28%), with a relative reduction of 29%, while the number reaching 100% coverage increases from 38 to 52. The gains are concentrated in the medium-to-high coverage range: the $[70\%, 90\%)$ interval contains 38 fewer tasks, and the number at or above 90% increases

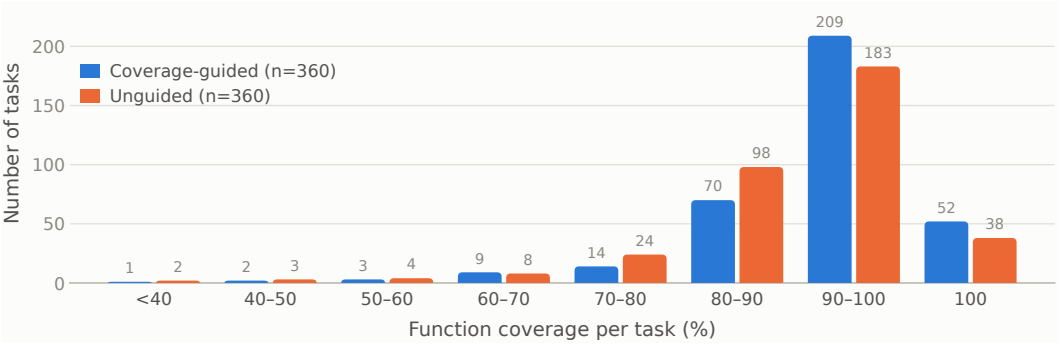


Figure 2: Effect of coverage guidance on function coverage during exploration.

by 40, from 221 to 261. The low-coverage tail changes little, with 17 tasks below 70% coverage without guidance and 15 with guidance.

Manual inspection shows that applications below 70% coverage typically contain failures that block further exploration. Examples include an entry page that immediately reports an error, an event handler that throws an exception, and a broken critical path that makes downstream functionality unreachable. These observations suggest that application defects contribute substantially to the remaining low coverage. Such failures also prevent real users from reaching the affected functionality, making them relevant to the evaluation of user experience.

In addition, the LLM-generated applications remain simple. Prior GUI-testing studies (Liu et al., 2026c; Gu et al., 2025) report greater exploration difficulty on complex applications (Liu et al., 2025a; Gu et al., 2025); coverage guidance is expected to bring more benefits as model capabilities grow and models generate more complex applications.

6 Limitations

LLM judges. The evaluation process largely relies on LLMs. Therefore, judge preferences may affect the results. We validated the reliability of the scores by comparing them with human preferences (§5.3) and verified scoring stability by comparing results after replacing the judge (§5.4).

Model-pool dependence. Standardization makes z-scores dependent on the participating LLMs and unsuitable for direct comparison across pools. Future deployments could use fixed reference means and standard deviations. Raw scores further aid interpretation.

Task scope. The dataset covers single-turn, text-only, front-end-only tasks from one platform, and excludes iterative repair, multimodal input, and back-end integration. Since the evaluation process simulates how real users use webpages, it can be extended to other frameworks and webpages with back-end functionality; considering the engineering cost, we leave this as future work.

Open source. To prevent data leakage and comply with our business confidentiality policy, we do not publicly release the dataset or source code.

7 Conclusion

In this paper, we introduced WebCraftBench, which evaluates web application generation from a software testing perspective. Combining coverage-guided exploration, state abstraction, and separated scoring grounded in runtime evidence, WebCraftBench can objectively and comprehensively score generated web applications, thereby reliably evaluating the capability of LLMs to generate web applications. WebCraftBench achieves 85.3% agreement with human preferences, with stable rankings across judge models. Coverage guidance increases the coverage during exploration, while the results across 17 frontier LLMs reveal complementary strengths in aesthetics, usability, and requirement alignment.

AI USE STATEMENT

In this work, we used generative AI tools for implement methods, assist with translation, clean and reformat dataset, interpret results. We have not used generative AI tools for generate synthetic data sets, support qualitative and thematic data analysis, help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, and formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools for create or modify scientific figures or images, create or edit software code, creation of artifacts, edit a research paper to improve readability. We have reviewed all AI-assisted work. LLM-generated code was carefully verified and tested by 2 authors. The LLM-polished texts are checked at least twice by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Tony Beltramelli. Pix2Code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2018.

Tanvir Bhathal and Asanshay Gupta. Websight: A vision-first architecture for robust web agents. *arXiv preprint arXiv:2508.16987*, 2025.

Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, 1997.

Dasen Dai, Biao Wu, Meng Fang, Shuoqi Li, and Wenhao Wang. I-WebGenBench: Evaluating interactivity in LLM-generated scientific web applications. *arXiv preprint arXiv:2606.00750*, 2026.

Design Arena. Design arena. `https://www.designarena.ai/`, 2026.

Google. Ui/application exerciser monkey. `https://developer.android.com/studio/test/other-testing-tools/monkey`, 2025.

Zhiyu Gu, Chenxu Liu, Guoquan Wu, Yifei Zhang, ChenXi Yang, Zheheng Liang, Wei Chen, and Jun Wei. Deep reinforcement learning for automated web GUI testing. *arXiv preprint arXiv:2504.19237*, 2025.

Istanbul Contributors. Nyc/istanbul, a javascript test coverage tool. `https://istanbul.js.org/`, 2024.

Xinping Lei, Xinyu Che, Junqi Xiong, Chenchen Zhang, Yukai Huang, Chenyu Zhou, Haoyang Huang, Minghao Liu, Letian Zhu, Hongyi Ye, et al. WebCompass: Towards multimodal web coding evaluation for code language models. *arXiv preprint arXiv:2604.18224*, 2026.

Chenxu Liu, Zhiyu Gu, Guoquan Wu, Ying Zhang, Jun Wei, and Tao Xie. Temac: Multi-agent collaboration for automated web GUI testing. *arXiv preprint arXiv:2506.00520*, 2025a.

Chenxu Liu, Junheng Wang, Wei Yang, Ying Zhang, and Tao Xie. Judge: Effective state abstraction for guiding automated web GUI testing. *ACM Transactions on Software Engineering and Methodology*, 2025b.

Chenxu Liu, Yingjie Fu, Wei Yang, Ying Zhang, and Tao Xie. Webcoderbench: Benchmarking web application generation with comprehensive and interpretable evaluation metrics. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026a.

Chenxu Liu, Junheng Wang, Zihe Song, Wei Yang, Ying Zhang, and Tao Xie. Booster: Effective and efficient web GUI trace reduction based on multi-level state abstraction and time-guided hierarchical delta debugging. *ACM Transactions on Software Engineering and Methodology*, 2026b.

Chenxu Liu, Wei Yang, Ying Zhang, and Tao Xie. Understanding automated web GUI testing: An empirical study across exploration strategies and state abstractions. *arXiv preprint arXiv:2606.16650*, 2026c.

Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. Make LLM a testing expert: Bringing human-like interaction to mobile GUI testing via functionality-aware decisions. In *Proceedings of the 46th International Conference on Software Engineering*, 2024.

LMArena.ai. Code arena WebDev leaderboard. `https://arena.ai/leaderboard/code/webdev`, 2026.

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. *Advances in Neural Information Processing Systems*, 2025.

Yuxin Meng, Yuhan Suo, Junjie Wang, Yuhan Sun, Yiyao Yu, Ruixu Zhang, Ruining Hu, Yubin Wang, Shouwei Ruan, Bin Wang, et al. WebRISE: Requirement-induced state evaluation for MLLM-generated web artifacts. *arXiv preprint arXiv:2606.03220*, 2026.

Ali Mesbah, Arie Van Deursen, and Stefan Lenselink. Crawling Ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web*, 2012.

Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. `https://playwright.dev/`, 2025.

Karl Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 1895.

Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 1987.

Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. FullFront: Benchmarking MLLMs across the full front-end engineering workflow. *arXiv preprint arXiv:2505.17399*, 2025.

Tung D Vu, Chung Hoang, and Truong-Son Hy. Multimodal graph representation learning for website generation based on visual sketch. *arXiv preprint arXiv:2504.18729*, 2025.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 2020.

Xueqing Wu, Zihan Xue, Da Yin, Shuyan Zhou, Kai-Wei Chang, Nanyun Peng, and Yeming Wen. Frontalk: Benchmarking front-end development as conversational code generation with multimodal feedback. *arXiv preprint arXiv:2601.04203*, 2025.

Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. Web-Bench: A LLM code benchmark based on web standards and frameworks. *arXiv preprint arXiv:2505.07473*, 2025.

Rahul Krishna Yandrapally and Ali Mesbah. Fragment-based test generation for web apps. *IEEE Transactions on Software Engineering*, 2022.

Rahulkrishna Yandrapally, Andrea Stocco, and Ali Mesbah. Near-duplicate detection in web app model inference. In *Proceedings of the 42nd International Conference on Software Engineering*, 2020.

Haoyue Yang, Zhangxiao Shen, Fan Ding, Hangting Lou, Yifeng Kou, Haoqing Yu, Jingyao Li, Zhengfan Wu, Siqi Bao, Jing Liu, et al. Cookie-Bench: Continuous on-screen key interaction evaluation for web generation. *arXiv preprint arXiv:2605.30000*, 2026.

Chengyue Yu, Siyuan Lu, Chenyi Zhuang, Dong Wang, Qintong Wu, Zongyue Li, Runsheng Gan, Chunfeng Wang, Siqi Hou, Gaochi Huang, et al. Aworld: Orchestrating the training recipe for agentic AI. *arXiv preprint arXiv:2508.20404*, 2025.

Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Changzhi Zhou, Ken Deng, Dengpeng Wu, Guanhua Huang, Kejiao Li, et al. ArtifactsBench: Bridging the visual-interactive gap in LLM code generation evaluation. *arXiv preprint arXiv:2507.04952*, 2025.

Wenyu Zhang, Guoliang You, Haotian Zhao, Tianshu Zhu, Haoran Wang, Xiaoxuan Tang, Mingyang Dai, Jingnan Gu, Daxiang Dong, Jianmin Wu, et al. WebGameBench: Requirement-to-application evaluation for coding agents via browser-native games. *arXiv preprint arXiv:2605.17637*, 2026a.

Zuhao Zhang, Chengyue Yu, Yuante Li, Chenyi Zhuang, Linjian Mo, and Shuai Li. MiniAppBench: Evaluating the shift from text to interactive HTML responses in LLM-powered assistants. *arXiv preprint arXiv:2603.09652*, 2026b.

Yan Zheng, Yi Liu, Xiaofei Xie, Yepang Liu, Lei Ma, Jianye Hao, and Yang Liu. Automatic web testing using curiosity-driven reinforcement learning. In *Proceedings of the 43rd International Conference on Software Engineering*, 2021.

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

Appendices

Table of Contents:

A Evaluation Details **2**

A.1 Instrumentation and Deployment 2

A.2 Scoring Procedures 2

B Detailed Judge-Robustness Results **3**

C The High-Margin Human–WebCraftBench Disagreement Case **4**

D Three-Round versus Single-Round Scoring **5**

E Raw Scores **6**

A Evaluation Details

A.1 Instrumentation and Deployment

The tool described in §4.1 instruments static and plain Node.js projects directly at the source level. For projects requiring compilation, it injects an instrumentation plugin and processes the generated files after the build. Framework detection selects the corresponding procedure automatically, and the instrumented application is deployed in an isolated sandbox over HTTP.

Instrumentation depends on application toolchains and may affect deployment or runtime behavior. Section 5.5 reports its applicability and deployment checks. Since Istanbul is a time-tested and widely used instrumentation tool, we believe that the instrumentation does not harm the application behavior. The black-box fallback retains applications with instrumentation failures for exploration and scoring without coverage feedback.

A.2 Scoring Procedures

This section details the evidence requirements, judgment procedures, and scoring aspects used in §4.4.

Deliberation and score aggregation. Visual aesthetics and usability each use three rounds. The advocate identifies strengths, and the critic identifies defects; both must cite concrete observations without assigning scores. The final judge checks their claims against the supplied evidence, discards unsupported claims, and adds relevant observations that both rounds missed. It selects a 0–5 base score from the overall visual impression or interaction experience, then records a signed adjustment and a supporting rationale for each of seven aspects, explicitly indicating when no adjustment is needed. The sum of the base score and adjustments is clipped to $[0, 5]$. With the advocate and critic assessments held fixed, the final judge runs independently five times. The highest and lowest scores are discarded, and the remaining mean is multiplied by 20 to obtain the 0–100 score.

Visual aesthetics. The seven aspects are first impression, content completeness, layout structure, visual detail, information hierarchy and typography, stylistic consistency, and emotion and trust. Observations must refer to specific locations in the representative screenshots. The assessment distinguishes deliberate simplicity from unfinished or empty content and examines the quality of graphics and layout rather than merely their presence. Factual correctness alone does not establish visual quality, and the judge must not infer failures in unobserved dynamic behavior from static screenshots.

Usability. The seven interaction-oriented aspects are effectiveness of core operations, immediacy of feedback, quality of controls and input, interaction flow and discoverability, state reachability and recovery, avoidance of dead ends and errors together with response consistency, and stability and fault tolerance. Each assessment must cite an exploration-trace step or a state-transition graph node or edge. The judge considers observed responses, navigation, recovery, and failures, without inventing requirements for unobserved features. A path that was not tested is neither a strength nor a weakness; observed failures that block core operations warrant substantial penalties.

Requirement alignment. Functional, content, and visual criteria share one evidence-retrieval process but receive separate satisfaction percentages. Starting from the state-transition graph and state overview, the agent retrieves relevant trace steps, DOM passages, and screenshots as needed. Functional criteria are checked against available features, executed interactions, and resulting state changes; content criteria against the specified text, data, and resources; and visual criteria against the required appearance and components. The agent reports a Boolean decision, supporting evidence, a brief analysis, and a confidence estimate for each criterion. A positive decision requires evidence that the property is present in the running application; required functionality must be reachable and observed to work. Unexecuted source code alone is insufficient, and no partial-credit category is used. Each sub-dimension receives 100 times the fraction of its criteria judged satisfied.

B Detailed Judge-Robustness Results

The experiment in §5.4 replaces Claude-Opus-4.8 with Gemini-3.7-Flash while holding the generated applications, exploration traces, scoring procedures, and prompts fixed. We report the complete ranking comparison and changes in raw-score distributions below.

Table 5 shows that model rankings remain stable across the two judges. Overall model scores have a Pearson correlation of $r = 0.983$ and a Spearman rank correlation of $\rho = 0.980$. Of the 136 model pairs, 130 retain their relative order, corresponding to 95.6% pairwise agreement. The top four ranks and ranks 8, 9, and 12 remain unchanged.

Of the six pairs whose order changes, three have overall-score differences below 0.04 under the main judge. This pattern is consistent with the concentration of human disagreements among near-tied applications in §5.3. Of the remaining three reversals, one is between Hy4 preview and Grok-4.5, while two involve MiniMax-M3, which rises from rank 17 to 15 under Gemini-3.7-Flash. Hy4 preview moves from rank 5 to 6, and Grok-4.5 moves from rank 7 to 5. All models move by at most two rank positions.

Raw-score changes vary by dimension; Appendix E reports the underlying means and ranges. Scores are standardized separately within each judge's 17-model pool and aggregated with the same weights. The correlations and pairwise agreement above quantify the resulting ranking robustness.

Table 5: Score and ranking differences across 17 models. Main: Claude-Opus-4.8; Gem.: Gemini-3.7-Flash. Scores are standardized within the 17-model pool. Rk. is the Gemini rank; Δ is that rank minus the main-judge rank.

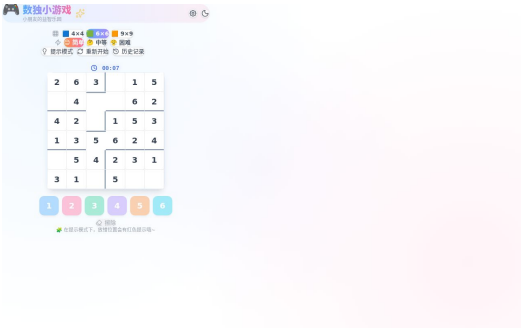
		Total						Total			
#	**Model**	**Main**	**Gem.**	**Rk.**	Δ	**#**	**Model**	**Main**	**Gem.**	**Rk.**	Δ
1	Claude-Opus-5	$+1.326$	$+1.602$	1	0	10	GLM-5.2	-0.315	-0.254	11	$+1$
2	GPT-5.6-Sol	$+1.077$	$+1.331$	2	0	11	DeepSeek-V4-Flash	-0.347	-0.107	10	-1
3	Qwen3.8-Max	$+0.970$	$+0.816$	3	0	12	Hy3	-0.592	-0.796	12	0
4	Kimi-K3	$+0.967$	$+0.729$	4	0	13	Qwen3.7-Max	-0.907	-0.987	14	$+1$
5	Hy4 preview	$+0.911$	$+0.712$	6	$+1$	14	GLM-5.1	-0.943	-0.944	13	-1
6	Claude-Opus-4.8	$+0.755$	$+0.671$	7	$+1$	15	DeepSeek-V4-Pro-Prev	-1.128	-1.206	16	$+1$
7	Grok-4.5	$+0.750$	$+0.726$	5	-2	16	Kimi-K2.7-Code	-1.311	-1.413	17	$+1$
8	GPT-5.5	$+0.468$	$+0.357$	8	0	17	MiniMax-M3	-1.419	-1.172	15	-2
9	Claude-Opus-4.7	-0.262	-0.067	9	0						

C THE HIGH-MARGIN HUMAN–WEBCRAFTBENCH DISAGREEMENT CASE

Figure 3 shows the only disagreement among the 50 human-preference pairs with $\Delta z \geq 0.75$. The human rater prefers the artifact in Figure 3(a), which has a more polished and visually coherent page. Runtime testing reveals that its primary gameplay path crashes after entry. The artifact in Figure 3(b) has a sparser interface, but its core Sudoku interaction works. WebCraftBench prefers this artifact because its advantages in usability and requirement alignment outweigh the visual advantage of the first artifact. The case is consistent with different tradeoffs between aesthetics and functional reliability, although the vote alone does not identify the rater's weighting.



(a) Human-preferred artifact: more visually polished, but the primary gameplay path crashes.



(b) WebCraftBench-preferred artifact: less visually polished, but the core Sudoku interaction works.

Figure 3: The only disagreement among the 50 human-preference pairs with $\Delta z \geq 0.75$. The human rater prefers the visually polished artifact in (a); WebCraftBench prefers the artifact in (b), whose core functionality works during runtime testing.

D THREE-ROUND VERSUS SINGLE-ROUND SCORING

We compare application-level scores from the three-round deliberative protocol with those from direct single-round scoring to assess the effect on score distributions. The single-round baseline directly uses the final round prompt of the three-round protocol. Figure 4 presents paired results for visual aesthetics and usability, with the dashed diagonal indicating equal scores. The deliberative protocol lowers mean scores by 15.5 points for aesthetics and 8.5 points for usability, suggesting that advocacy, criticism, and evidence verification lead to more conservative judgments. Artifacts assigned identical or similarly saturated scores in a single round also receive a wider range of scores after deliberation. These results indicate reduced concentration at the scale endpoints and greater differentiation among artifacts that direct scoring evaluates similarly.

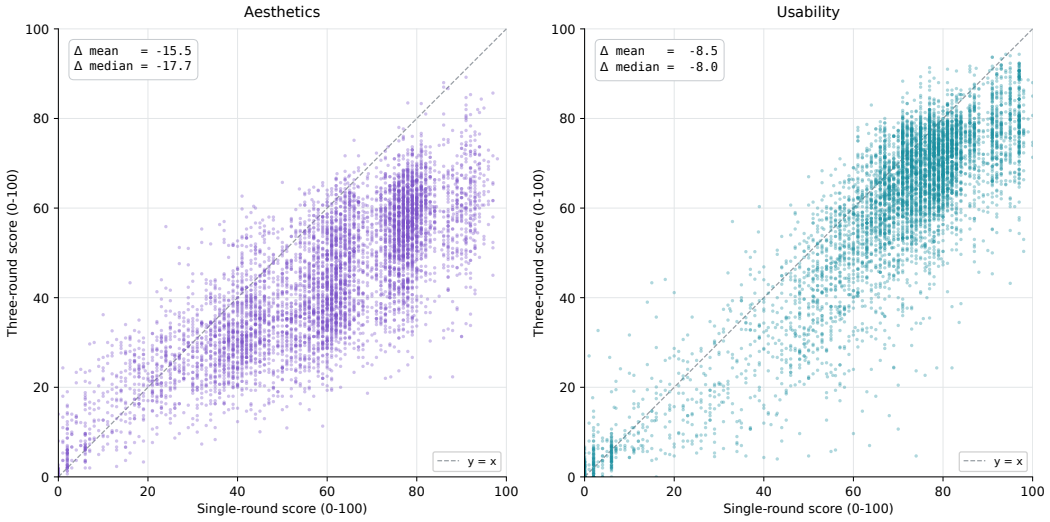


Figure 4: Application-level comparison of direct single-round scoring (horizontal axis) and the three-round deliberative protocol used by WebCraftBench (vertical axis) for visual aesthetics and usability. Each point is one paired evaluation; the dashed line marks equal scores. The annotations report differences in the mean and median scores (three-round minus single-round), rather than the median of paired score differences.

E RAW SCORES

The main-text rankings aggregate standardized component scores to account for differences in component dispersion (§4.4). Scores are relative to the evaluated model pool and are not directly comparable across studies. This appendix reports raw means under Claude-Opus-4.8 and Gemini-3.7-Flash for the same 17-model pool used in §B.

Table 6 reports raw scores on a 0–100 scale. For visual aesthetics and usability, five final-judge scores on a 0–5 scale are aggregated using a trimmed mean and multiplied by 20. Raw means are rounded to one decimal place for presentation and may obscure distinctions retained in the standardized scores. Rows follow Table 3. Column means and ranges are computed over all 17 models, consistently with the ranking comparison.

Additional correlations highlight complementary aspects of application quality: visual aesthetics and requirement alignment are weakly correlated at the artifact level ($r = 0.32$), while usability and requirement alignment are strongly associated at the model level ($r = 0.88$).

Component scales differ substantially: under the main judge, aesthetics spans 26.8 points across models, whereas the alignment sub-dimensions span 7.1–10.4 points. Averaging raw components would therefore give aesthetics greater influence on between-model differences. Standardization equalizes component dispersion before applying the stated weights. The cross-judge differences in this table concern score level and spread; ranking robustness is evaluated separately in Table 5.

Table 6: Raw model-level mean scores under both judges (0–100). Alignment scores are percentages of criteria satisfied. Rows follow Table 3; column means and ranges cover all 17 models.

		Main judge (Claude-Opus-4.8)					**Gemini-3.7-Flash**				
				Alignment					Alignment		
#	**Model**	**Aesth.**	**Usab.**	**Func.**	**Cont.**	**Vis.**	**Aesth.**	**Usab.**	**Func.**	**Cont.**	**Vis.**
1	Claude-Opus-5	53.5	69.6	96.7	95.9	95.6	59.5	89.1	99.2	98.9	99.5
2	GPT-5.6-Sol	59.1	64.7	95.4	94.7	93.7	70.3	82.5	97.6	97.9	98.0
3	Qwen3.8-Max	49.1	66.0	96.5	96.4	96.9	50.7	84.2	98.1	97.4	98.0
4	Kimi-K3	47.7	65.6	96.9	96.8	97.9	47.2	82.7	97.9	98.1	98.6
5	Hy4 preview	47.1	66.0	96.3	96.9	97.1	46.1	81.9	98.0	98.4	98.9
6	Claude-Opus-4.8	44.7	64.7	97.6	96.5	97.4	43.4	82.1	98.4	98.0	99.4
7	Grok-4.5	47.3	64.9	96.3	95.7	96.5	47.4	83.1	98.0	97.4	98.9
8	GPT-5.5	48.7	61.3	94.6	95.5	95.6	49.3	75.2	97.2	97.6	98.5
9	Claude-Opus-4.7	42.6	57.7	93.0	94.2	94.0	41.0	72.8	97.1	97.2	98.1
10	GLM-5.2	41.9	57.1	92.5	94.8	94.2	39.3	71.7	96.2	97.5	97.4
11	DeepSeek-V4-Flash	41.5	58.4	92.4	93.9	93.5	39.4	75.5	96.6	96.8	97.4
12	Hy3	39.0	57.3	91.9	93.2	93.5	34.9	71.5	94.4	95.4	95.8
13	Qwen3.7-Max	36.6	54.8	91.8	92.7	93.3	32.0	67.3	95.1	95.1	96.7
14	GLM-5.1	35.7	54.6	92.1	93.4	92.6	29.1	68.5	95.8	95.0	97.2
15	DeepSeek-V4-Pro-Prev	33.6	52.8	91.6	93.5	93.4	27.0	66.5	94.8	95.2	96.4
16	Kimi-K2.7-Code	32.3	53.7	90.6	93.3	90.7	26.1	66.1	94.9	94.8	94.5
17	MiniMax-M3	41.4	52.3	87.2	89.8	88.5	40.4	66.2	92.8	94.6	94.0
pool mean		43.6	60.1	93.7	94.5	94.4	42.5	75.7	96.6	96.8	97.5
range		26.8	17.3	10.4	7.1	9.4	44.2	23.0	6.4	4.3	5.5