WEBMETABENCH: TOWARDS INTERACTIVE WEB TESTING AS MULTIMODAL META-EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Contemporary web development agents can build sophisticated websites with rich interactions, compelling visual effects, and polished aesthetics. However, these advances also make reliable evaluation of agent-generated websites increasingly challenging. Existing evaluation methods largely rely on predefined programmatic checks or static screenshots, limiting their ability to assess diverse, open-ended web development requirements, such as motion effects, animations, and interactive responses. Addressing these limitations calls for agentic evaluators that can adapt their testing to each requirement, acquire diagnostic video evidence, and reason about the observed dynamics. To assess the reliability of such evaluators, we introduce **WebMetaBench**, an interactive black-box benchmark for multimodal meta-evaluation of web development agents. It contains 120 single-defect and 19 multi-defect instances, with a gold-video setting that decouples active testing from video understanding and enables fine-grained evaluation of judge reliability. We further propose a three-stage failure paradigm consisting of requirement coverage, evidence acquisition, and video reasoning for fine-grained diagnosis of agent capabilities. Our experiments reveal substantial limitations in current models, with the best model achieving only 25.8% accuracy on single-defect instances and 26.9% defect recall in the multi-defect setting. WebMetaBench exposes critical limitations in both active testing and video understanding, providing a foundation for developing reliable judges for complex web development.

1 INTRODUCTION

Web development (WebDev) agents are increasingly capable of end-to-end application development beyond isolated code generation (Anthropic, 2026b; DeepSeek-AI, 2026; OpenAI, 2026b). Their expanding scope makes reliable evaluation critical and challenging, motivating systematic assessment of agentic judges. For complex frontends, many requirements concern dynamic behaviors rather than static page states (Chen et al., 2025; Dai et al., 2026). Particles and animations evolve continuously, while object motion and interactive responses unfold over time. Yet existing evaluation relies mainly on structured page information, static screenshots, or predefined checks, leaving such behaviors insufficiently assessed. Video preserves their temporal evolution and provides behavioral evidence that static observations cannot adequately capture.

In realistic WebDev evaluation, however, relevant video evidence is not given in advance. A judge must navigate and interact with the application to capture diagnostic evidence, then reason over the resulting dynamics against the acceptance requirements. This process constitutes requirement-driven interactive black-box testing. From a WebDev perspective, the judgments also serve as supervision and reward signals, making judge reliability itself a problem of multimodal meta-evaluation (Lù et al., 2025; Zhuge et al., 2024). Recent web judge benchmarks have begun to study Agent-as-a-Judge reliability, but systematic evaluation of the capabilities required for active testing and video-based judgment remains largely unexplored (Li et al., 2026; Lei et al., 2026).

To address this gap, we introduce **WebMetaBench**, an interactive black-box benchmark for multimodal meta-evaluation of WebDev agents. Given a deployed application and an acceptance checklist, an agent autonomously plans its inspection, interacts with the application, collects video evidence, and identifies requirement-level defects, capturing the full agentic process of WebDev evaluation. WebMetaBench contains 120 single-defect and 19 multi-defect instances from 42 complex

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

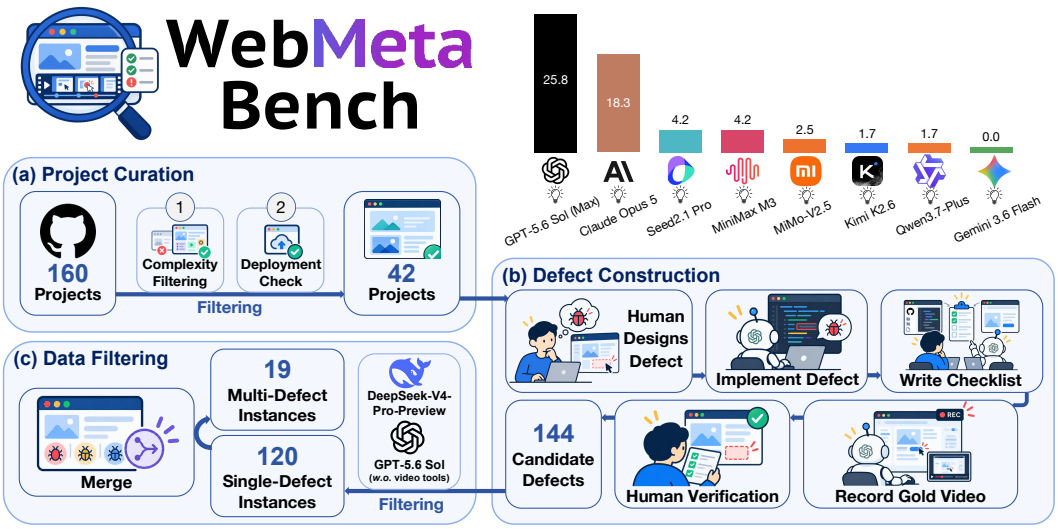


Figure 1: Overview of WebMetaBench. Through project curation, defect construction, and data filtering, WebMetaBench builds 120 single-defect and 19 multi-defect instances from 42 complex frontend projects to evaluate agentic judges through active testing and video understanding. Frontier agents remain substantially challenged, with the best model achieving only 25.8% accuracy.

frontend projects, with controlled requirement-level defects enabling direct evaluation of judge reliability. We further introduce a gold-video ablation that provides diagnostic videos containing the target defects, enabling independent evaluation of video understanding.

Our experiments reveal substantial limitations in current models. The best agent achieves only 25.8% accuracy on single-defect instances and 26.9% recall in the multi-defect setting. Through systematic analysis of agent behaviors, we further propose a three-stage failure paradigm consisting of requirement coverage, evidence acquisition, and video reasoning, enabling fine-grained diagnosis beyond end-to-end performance. These results expose critical limitations in both active testing and video understanding, establishing WebMetaBench as a testbed for reliable Agent-as-a-Judge and automated reward in complex web development. Our main contributions are as follows:

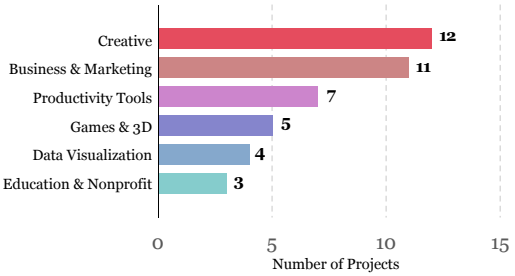
- We formulate complex WebDev evaluation as an interactive black-box agent task that combines active testing with video reasoning, and further position it as multimodal meta-evaluation of WebDev agents by assessing the reliability of their judges and supervision signals.
- We introduce **WebMetaBench**, a challenging benchmark with single- and multi-defect settings and a gold-video setup for independent evaluation of video understanding.
- Fine-grained analysis establishes a three-stage failure paradigm of requirement coverage, evidence acquisition, and video reasoning, revealing systematic bottlenecks of current agents.

2 Related Work

2.1 Web Development Benchmarks

WebDev benchmarks have evolved alongside increasingly complex development tasks. Early work focused on static visual-to-code generation and visual fidelity (Si et al., 2025; Yun et al., 2024; Zhang et al., 2026), followed by iterative editing, interactive web pages, and end-to-end development (Xu et al., 2026; He et al., 2026). Recent benchmarks further use video to capture dynamic requirements beyond static screenshots (Dai et al., 2026; Chen et al., 2025). Despite this progression, these benchmarks primarily evaluate whether agents can build or reproduce target applications, while their evaluators remain largely based on static observations or predefined checks, leaving dynamic WebDev outcomes that require active testing and video reasoning insufficiently assessed. WebMetaBench advances evaluation to match this growing development complexity by requiring judges to actively test applications and reason over their temporal visual behaviors.

Figure 2: Detailed statistics and distributions of WebMetaBench. **Left**: Distribution across application domains. **Right**: Statistics of project scale and codebase complexity. **PLOC**: Physical Lines of Code. **Cyclomatic Comp.**: Cyclomatic Complexity.



Metric	Mean	Median	Max
PLOC (K)	58.5	4.4	864.1
Dependencies	28.3	9	393
Code (MiB)	2.2	0.2	23.9
Assets (MiB)	30.7	10.2	379.9
Cyclomatic Comp.	8,435	470	83,740

2.2 Meta-Evaluation Benchmarks

Meta-evaluation measures evaluator reliability against human judgments or verifiable ground truth. Early benchmarks focused on text, preference, and reasoning evaluation (Bai et al., 2024; Tan et al., 2025; Zeng et al., 2024), while recent work extends to interactive agent tasks such as search and GUI operation (Lù et al., 2025; Shi et al., 2026; Wang et al., 2026). WebDev meta-evaluation further targets executable applications, but remains largely based on static visual evidence or limited interaction (Li et al., 2026; Lei et al., 2026; Zhang et al., 2025). Multimodal meta-evaluation has also emerged for tasks such as video generation, yet typically assumes given outputs rather than agent-acquired evidence (Liu et al., 2026; 2025). WebMetaBench brings these dimensions together, evaluating judges that must conduct long-horizon testing, acquire diagnostic video evidence, and reason over complex software behaviors within a unified WebDev task.

3 WebMetaBench

3.1 Overview of WebMetaBench

WebMetaBench is an interactive black-box benchmark for multimodal meta-evaluation of complex frontends, comprising 139 instances from 42 real-world projects. As illustrated in Figure 2, these projects span six common frontend domains and exhibit substantial application complexity. The benchmark includes 120 single-defect and 19 multi-defect instances across six categories that jointly challenge agentic interaction and video understanding, with representative cases shown in Figure 3.

Formally, each instance is represented as $\mathcal{I} = (W, C, G, V)$, where W denotes the deployed application, $C = \{c_1, \ldots, c_n\}$ the acceptance checklist, $G = \{g_1, \ldots, g_m\}$ the ground-truth defect set, and V the gold video evidence exposing the target defects. Given only W and C, an agent must autonomously inspect and interact with the application to acquire video evidence $\hat{V}$ and subsequently predict a defect set $\hat{G}$. This setting targets frontier agents with increasingly strong end-to-end project development capabilities, for which proactively identifying implementation issues has become particularly important.

3.2 Data Curation and Generation

WebMetaBench follows the three-stage data construction pipeline illustrated in Figure 1. For projects with multiple single-defect instances, we further merge independent and non-interfering defects to construct multi-defect instances.

Project Curation. We collect projects from open-source platforms such as GitHub. The selected projects span diverse frontend domains and feature rich dynamic visual effects and interaction logic, providing a foundation for constructing challenging test instances.

Defect Construction. Annotators design candidate defects around the original functionality and behavior of each project, targeting implementation errors that could plausibly arise in real-world development. With the assistance of GPT-5.6 Sol (OpenAI, 2026a), annotators implement these

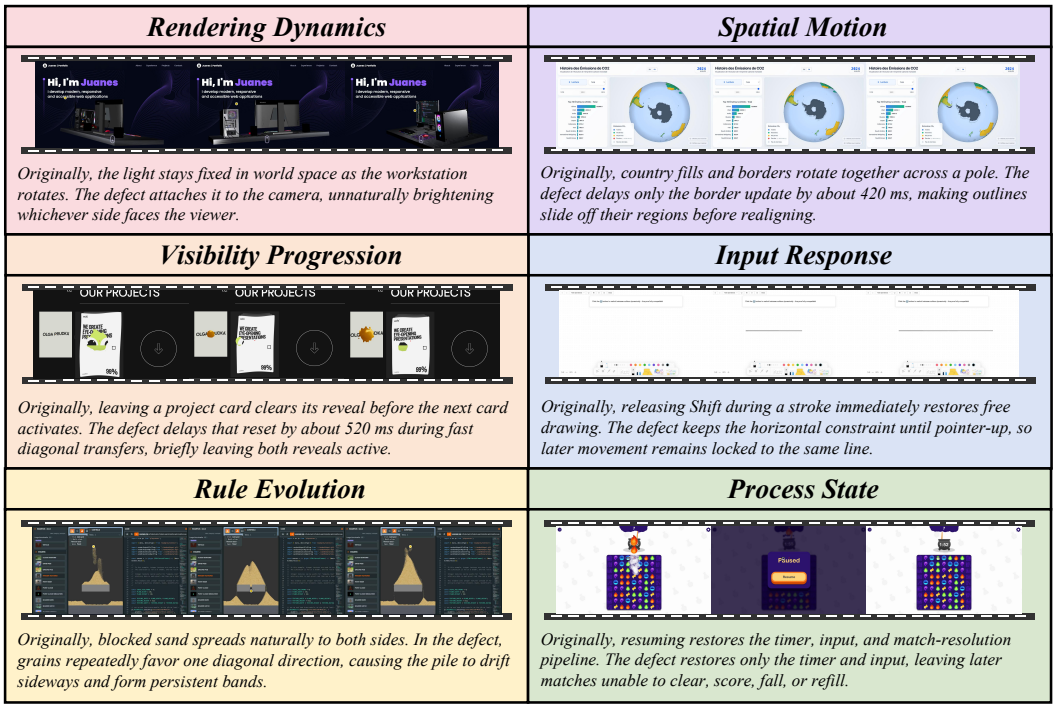


Figure 3: Representative cases of the six defect categories in WebMetaBench.

defects, construct comprehensive acceptance checklists, and record video evidence V that exposes the target defects. All instances are manually verified to ensure that the defects are reproducible, consistent with the specified requirements, and sufficiently challenging to distinguish.

Data Filtering. We evaluate all candidates using DeepSeek-V4-Pro-Preview (DeepSeek-AI et al., 2026) and GPT-5.6 Sol, with video tools disabled for the latter. Any instance whose target defect can be correctly identified without video input is removed, excluding cases that can be solved from textual or structured page information alone. Further construction details are provided in Appendix B.

4 Experiments

4.1 Experimental Setup

Models. We evaluate 10 frontier models, most with native video support (OpenAI, 2026a; Anthropic, 2026a; ByteDance Seed Team, 2026; Team et al., 2026; Alibaba Cloud, 2026; Google DeepMind, 2026; Moonshot AI, 2026; Lai et al., 2026). For image-only models, we provide sampled video frames, with the sampling rate controlled by the agent to accommodate different dynamic behaviors.

Harness. Claude Code serves as the default agent harness, while GPT-5.6 models are paired with Codex, the provider-developed harness. All models use the same browser tools and runtime environment, with webpage interaction supported by our Playwright-based MCP server and headless Chromium. Full configurations are provided in Appendix C.

Evaluation Protocol. We use Qwen3.7-Flash to evaluate whether the final response correctly identifies the defects. The evaluator has no access to the trajectory or collected evidence, reducing scoring to a direct comparison with human-annotated ground truth and keeping it independent of the active testing and video understanding capabilities under evaluation. Robustness checks and scoring details are provided in Appendix D.

Table 1: Main results on WebMetaBench. **RD**: *Rendering Dynamics*; **SM**: *Spatial Motion*; **IR**: *Input Response*; **VP**: *Visibility Progression*; **PS**: *Process State*; **RE**: *Rule Evolution*.

Model	Harness	Avg.	RD	SM	IR	VP	PS	RE
GPT-5.6 Sol	Codex	**25.8**	12.0	**28.0**	**40.0**	15.0	**26.7**	**40.0**
GPT-5.6 Terra	Codex	20.0	4.0	24.0	35.0	**20.0**	13.3	26.7
Claude Opus 5	Claude Code	18.3	**20.0**	4.0	**40.0**	10.0	20.0	20.0
GPT-5.6 Luna	Codex	17.5	12.0	12.0	**40.0**	10.0	20.0	13.3
Seed2.1 Pro	Claude Code	4.2	0.0	8.0	5.0	10.0	0.0	0.0
MiniMax M3	Claude Code	4.2	0.0	0.0	10.0	10.0	0.0	6.7
MiMo-V2.5	Claude Code	2.5	0.0	4.0	5.0	5.0	0.0	0.0
Kimi K2.6	Claude Code	1.7	0.0	0.0	5.0	5.0	0.0	0.0
Qwen3.7-Plus	Claude Code	1.7	0.0	0.0	0.0	10.0	0.0	0.0
Gemini 3.6 Flash	Claude Code	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4.2 Main Results

4.2.1 Single-Defect Evaluation

Table 1 presents our evaluation results, revealing a clear performance landscape and several key observations. More detailed results are provided in Appendix D.

Reliable WebDev judging remains highly challenging. GPT-5.6 Sol achieves the highest overall accuracy at only 25.8%, with the GPT-5.6 series and Claude Opus 5 forming a clear top tier. All remaining models score below 5%. The unsatisfactory overall performance and pronounced tier gap highlight reliable end-to-end WebDev judging as a major challenge for current agents.

Similar overall accuracy can mask distinct category-level capabilities. GPT-5.6 Sol and Claude Opus 5 achieve comparable overall performance, yet exhibit contrasting results on spatial motion and dynamic visual effects. This divergence demonstrates the discriminative value of our defect taxonomy, enabling a more informative characterization of agent capabilities and offering guidance for targeted improvements.

Defect categories exhibit distinct difficulty patterns across model tiers. Among top-tier models, *Input Response* is relatively tractable, whereas *Rendering Dynamics* and *Visibility Progression* remain consistently harder. The former has explicit interaction targets and direct feedback, while the latter involves subtle or transient visual changes that are harder to capture reliably. *Spatial Motion* lies between them, with more salient displacement cues. Interestingly, *Visibility Progression* is less degraded among second-tier models and becomes their most frequently solved category. These patterns motivate the following fine-grained analyses.

4.2.2 Multi-Defect Evaluation

To better reflect practical WebDev evaluation, where multiple requirement violations may coexist, we evaluate agents on 19 multi-defect instances (Table 2).

Table 2: Results on the multi-defect setting. **Ma.**: Macro Dice; **Mi.**: Micro Dice.

Model	Recall	Ma.	Mi.
GPT-5.6 Sol	26.9	23.0	21.4
GPT-5.6 Luna	14.9	15.1	14.0
GPT-5.6 Terra	13.4	15.0	14.5
Claude Opus 5	11.9	12.1	12.5
MiniMax M3	1.5	3.5	2.3
Gemini 3.6 Flash	1.5	2.6	2.2
Qwen3.7-Plus	1.5	2.1	2.1
Kimi K2.6	1.5	1.3	2.6
MiMo-V2.5	0.0	0.0	0.0

Performance remains consistently low in realistic multi-defect settings. Despite having more targets per instance, models do not become more likely to trigger and identify defects, showing performance comparable to the single-defect setting. Most models still report only a few defects, demonstrating that WebMetaBench remains highly challenging in the more realistic setting.

Finding more defects comes at the cost of false alarms. Stronger models tend to achieve broader defect coverage through more predictions, whereas weaker models produce fewer false positives but identify almost no defects. These results reveal system-

Table 3: Results of the video reasoning analysis. Δ: the absolute improvement over the main results.

Model	Avg.	Δ	RD	SM	IR	VP	PS	RE
GPT-5.6 Sol	**60.8**	+35.0	**60.0**	**72.0**	60.0	**65.0**	**60.0**	**40.0**
GPT-5.6 Terra	46.7	+26.7	48.0	52.0	50.0	45.0	46.7	33.3
Seed2.1 Pro	45.0	+40.8	36.0	40.0	45.0	60.0	53.3	**40.0**
GPT-5.6 Luna	43.3	+25.8	48.0	40.0	45.0	55.0	33.3	33.3
Gemini 3.6 Flash	37.5	+37.5	32.0	32.0	40.0	50.0	40.0	33.3
Claude Opus 5	34.2	+15.9	24.0	20.0	**65.0**	45.0	33.3	20.0
Kimi K2.6	21.7	+20.0	12.0	12.0	25.0	35.0	33.3	20.0
Qwen3.7-Plus	18.3	+16.6	4.0	20.0	20.0	35.0	26.7	6.7
MiMo-V2.5	15.0	+12.5	12.0	4.0	25.0	20.0	13.3	20.0
MiniMax M3	10.8	+6.6	12.0	0.0	20.0	15.0	6.7	13.3

atic limitations in current agents, with substantial room for improvement in both completeness and precision.

4.3 Fine-grained Analysis

4.3.1 Video Reasoning Analysis

To isolate video reasoning from upstream testing, we provide each single-defect instance with a gold video and recording trajectory, bypassing evidence acquisition. Videos are provided at 720p and 16FPS to preserve sufficient details of the target behaviors. Results are shown in Table 3.

Gold evidence reveals substantial difficulty in evidence acquisition. All models improve markedly, with Seed2.1 Pro gaining 40.8 points and Gemini 3.6 Flash rising from 0% to 37.5%. These gains show that acquiring evidence remains challenging in end-to-end WebDev judging.

Video reasoning remains challenging with gold evidence. With evidence acquisition completed, the strongest model, GPT-5.6 Sol, still reaches only 60.8%, while most models remain below 50%. Reliable WebDev judging thus requires both evidence acquisition and video reasoning.

End-to-end and video reasoning difficulty differ across categories. *Spatial Motion*, *Visibility Progression*, and *Rendering Dynamics* improve substantially with gold evidence. Notably, visibility progression shifts from a difficult end-to-end category to a relatively easier one for several models, indicating substantial impact from upstream evidence acquisition.

4.3.2 Agent Behavior Analysis

Performance tiers correspond to distinct levels of testing activity. Top-tier systems (the GPT-5.6 series and Claude Opus 5) average over 100 tool calls and approximately 10M cumulative input tokens per task, indicating more sustained interaction and extensive context processing. These results highlight the long-horizon interaction demands of complex frontend acceptance testing.

Within the top tier, stronger models achieve more accurate verification with more efficient execution. GPT-5.6 Sol outperforms GPT-5.6 Luna by 8.3 points in accuracy while using 23.4% fewer tool calls and approximately 33.9% fewer input and output tokens, achieving more reliable verification with lower interaction and generation overhead. Long-horizon tasks may further amplify the impact of inefficient exploration. Redundant actions repeatedly introduce irrelevant records into subsequent requests, increasing context overhead and competing with requirement-relevant evidence. This may hinder the model's ability to maintain associations among requirements, actions, and temporal observations, ultimately affecting judgment reliability.

Video evidence collection further differentiates performance. In Figure 5, we report the average videos and screenshots collected per task across models. GPT-5.6 Sol and GPT-5.6 Terra have comparable screenshots, yet the former records 78.0% more videos and achieves 5.8 points higher accuracy. Across models, video counts generally align with accuracy, whereas tool call volume alone does not consistently indicate better performance. Stronger agents are distinguished not only

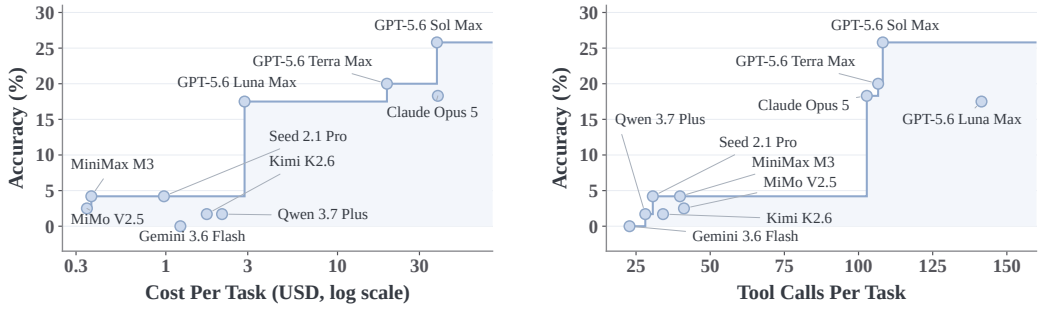


Figure 4: **Left**: Pareto frontier between cost per task and accuracy; **Right**: Pareto frontier between tool calls per task and accuracy.

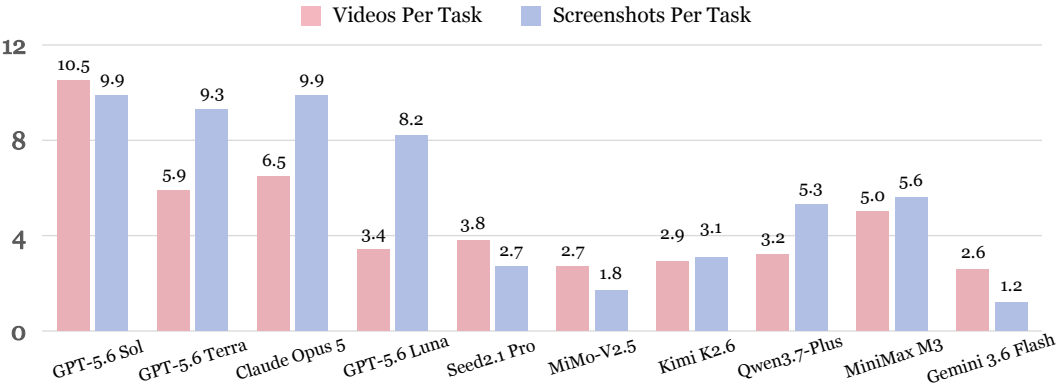


Figure 5: Statistics of video and screenshot collection behaviors during task execution.

by sustained interaction, but also by their ability to acquire more effective video evidence for reliable WebDev judging.

Cost and execution efficiency provide distinct advantages. Figure 4 presents the Pareto frontiers relating accuracy to cost and tool calls. GPT-5.6 Luna costs only 7.6% as much as GPT-5.6 Sol, at the expense of 30.6% more tool calls. Its lower inference cost offsets its larger input and output volumes, whereas GPT-5.6 Sol achieves more accurate verification with fewer environment interactions. These results reveal two complementary efficiency advantages, with lower inference costs enabling more scalable testing while efficient execution reducing interaction overhead.

4.4 Error Analysis

To comprehensively characterize agent performance and its limitations, we first use GPT-5.6 Sol for initial attribution and then manually review each case, reducing the cost of analyzing long-horizon trajectories while limiting systematic errors from meta-evaluation.

Error analysis reveals three sequential failure patterns, *Requirement Coverage* (RC), *Evidence Acquisition* (EA), and *Video Reasoning* (VR), which respectively capture missing the target requirement, failing to acquire valid defect evidence, and making an incorrect judgment despite valid videos. Since later failures can only occur after preceding stages succeed, each instance is assigned to its earliest failure stage (Figure 6). To account for this sequential structure, we additionally report conditional failure rates in Figure 7. Let N_{RC}, N_{EA}, N_{VR}, and N_{C} denote the numbers of the three failure types and correct instances, respectively:

$$r_{\mathrm{RC}} = \frac{N_{\mathrm{RC}}}{N}, \qquad r_{\mathrm{EA}} = \frac{N_{\mathrm{EA}}}{N - N_{\mathrm{RC}}}, \qquad r_{\mathrm{VR}} = \frac{N_{\mathrm{VR}}}{N_{\mathrm{VR}} + N_{\mathrm{C}}}.$$

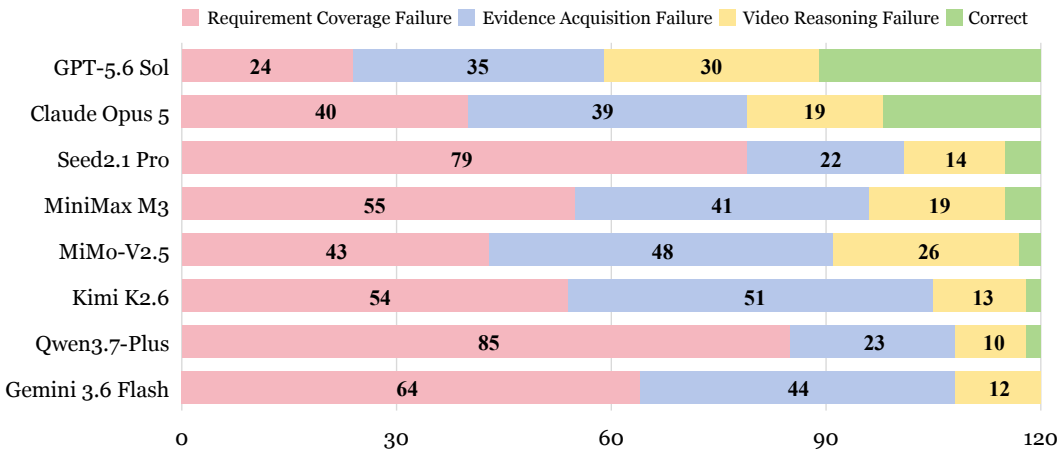


Figure 6: Distribution of three-stage failure patterns.

Failures remain prevalent across all three stages but exhibit distinct patterns across models. All three failure types occur frequently, revealing limitations throughout the WebDev judging pipeline. EA remains prevalent even among the strongest agents, with GPT-5.6 Sol and Claude Opus 5 failing to acquire valid evidence in 35 and 39 instances, respectively. This aligns with the improvements from gold videos in Section 4.3, indicating that evidence acquisition remains a fundamental challenge. Other models also exhibit distinct failure profiles. Qwen3.7-Plus and Seed2.1 Pro are primarily limited by requirement coverage, whereas Kimi K2.6, MiMo-V2.5, and MiniMax M3 more frequently fail in EA and VR. However, due to the sequential nature of the pipeline, these patterns should be considered together with preceding stages.

Stronger agents shift failures toward later stages. Top-tier agents exhibit substantially fewer RC failures and more actively inspect and verify implemented functionality, whereas other models often fail to cover all requirements. Improved agent capability is therefore reflected not only in higher final accuracy, but also in a systematic shift of failures toward later stages of the judging process.

Conditional analysis reveals cascading failures across stages. For the six other models, conditional failure rates exceed 50% at EA and 70% at VR, indicating substantial difficulties even after successful requirement coverage. Their low end-to-end accuracy thus reflects limitations across multiple stages, where upstream improvements alone are insufficient for final gains without corresponding downstream capability. Even among top-tier models, conditional failure rates at both stages remain above 30%, suggesting that these bottlenecks persist across capability levels.

End-to-end rankings do not always reflect stage-level advantages. GPT-5.6 Sol outperforms Claude Opus 5 end-to-end (25.8% vs. 18.3%), yet has a slightly higher conditional VR failure rate (49.2% vs. 46.3%). Its advantage arises primarily from upstream progression. GPT-5.6 Sol advances 61 instances to VR, compared with only 41 for Claude Opus 5, resulting in more successful final judgments overall. Thus, evidence acquisition and video reasoning ability do not necessarily improve in tandem, and end-to-end performance depends on both. Conditional analysis further distinguishes whether improvements arise from stronger upstream progression or better judgment after evidence acquisition.

Agentic capability shapes multimodal capability in autonomous evaluation. Across agents, conditional VR performance is generally consistent with performance using gold video. MiMo V2.5, for example, achieves 10.3% accuracy on instances reaching VR, close to its 15.0% accuracy with gold video, indicating that video judgment remains a major weakness even after removing the burden of active evidence acquisition. Gemini 3.6 Flash is a notable exception, as its video reasoning capability does not translate effectively to the end-to-end setting. This gap may be partly explained by the autonomous evidence acquisition process. Agents select video sampling rates during interaction, and suboptimal evidence selection may prevent Gemini from fully leveraging its video reasoning ability.

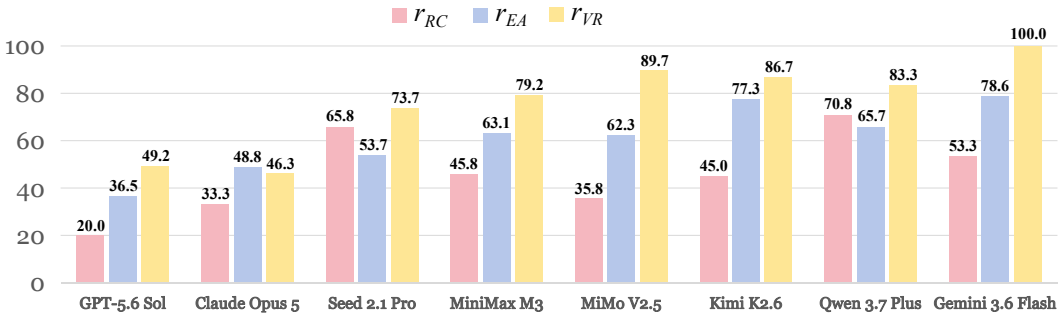


Figure 7: Conditional failure rates across three stages.

Table 4: Comparison of different harnesses. **CC**: Claude Code. **Codex**$^+$: Codex with conditional interaction recovery. Δ: the change from Codex to Codex$^+$.

Model	Accuracy (%)				Tool calls Per Task			Videos Per Task		
	CC	Codex	Codex$^+$	Δ	Codex	Codex$^+$	Δ	Codex	Codex$^+$	Δ
Qwen3.7-Plus	1.7	3.3	**10.0**	+6.7	25.9	36.9	+11.0	1.4	2.0	+0.6
Kimi K2.6	1.7	6.7	**12.5**	+5.8	28.7	35.9	+7.2	3.5	3.3	-0.2
MiMo-V2.5	2.5	1.7	**3.3**	+1.6	19.4	33.4	+14.0	1.5	2.5	+1.0

4.5 Harness Analysis

To investigate the impact of harness design, we additionally evaluate three models using Codex. We select Qwen3.7-Plus, Kimi K2.6, and MiMo-V2.5 as they have substantial improvement room and are not natively optimized for existing harnesses. As shown in Table 4, the two general-purpose harnesses achieve comparable overall performance across the evaluated models.

During harness analysis, we observe differences in tool invocation specifications between Codex and Claude Code. Inspecting agent trajectories reveals that some agents skip necessary tool interactions and directly output planned actions as final responses, providing an execution-level explanation for some observed *Requirement Coverage* failures. We introduce conditional interaction recovery in Codex, where the same model resumes interaction from the current context when abnormal termination is detected, such as empty responses or intermediate replies without tool calls.

After this enhancement, all three models improve, demonstrating the value of sustained execution in long-horizon evaluation. Qwen3.7-Plus and Kimi K2.6 benefit more substantially, as interaction recovery enables them to complete previously unfinished inspections, consistent with the previous error analysis. In contrast, MiMo-V2.5 still faces significant bottlenecks in subsequent evidence acquisition and video reasoning. These findings highlight the importance of jointly considering model capability and harness design in autonomous evaluation.

5 Conclusion and Future Work

We introduce WebMetaBench, an interactive black-box benchmark that addresses a critical gap in multimodal meta-evaluation by assessing the reliability of agentic judges in WebDev tasks. Our results reveal that current agents remain far from reliable evaluators, with limitations spanning both active testing and video understanding. We hope WebMetaBench provides a foundation for future research on reliable WebDev agents, including Agent-as-a-Judge, automated supervision and reward generation, and self-verifying software development. As agents continue to advance toward autonomous software development, the ability to reliably verify and improve their own outputs will become an essential capability for agentic software engineering.

AI USE STATEMENT

Beyond the uses in dataset construction and experiments disclosed in this paper, we used generative AI only for code assistance and language polishing. The literature review and initial manuscript were prepared by the authors. All AI-assisted outputs were reviewed and verified by the authors, who take responsibility for the final content.

REPRODUCIBILITY STATEMENT

To facilitate verification and future research, we have released the repository on https://anonymous.4open.science/r/webmetabench-0325/. The dataset repository will include WebMetaBench instances and associated evaluation resources, while the code repository will provide the evaluation pipeline, experimental configurations, and instructions for reproducing our results.

REFERENCES

Alibaba Cloud. Qwen3.7-plus. https://www.alibabacloud.com/help/en/model-studio/qwen3-7-plus, 2026. Alibaba Cloud Model Studio model documentation.

Anthropic. System card: Claude opus 5. https://www.anthropic.com/system-cards, July 2026a. Model system card.

Anthropic. Claude code. https://github.com/anthropics/claude-code, 2026b. GitHub repository.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7421–7454. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.401. URL http://dx.doi.org/10.18653/v1/2024.acl-long.401.

ByteDance Seed Team. Seed2.1: A next-generation agent for real-world productivity. https://seed.bytedance.com/en/seed2_1, 2026. Model card.

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.

Yuhong Dai, Yanlin Lai, Mitt Huang, Hangyu Guo, Dingming Li, Hongbo Peng, Haodong Li, Yingxiu Zhao, Haoran Lyu, Zheng Ge, Xiangyu Zhang, and Daxin Jiang. Webvr: Benchmarking multimodal llms for webpage recreation from videos via human-aligned visual rubrics, 2026. URL https://arxiv.org/abs/2603.13391.

DeepSeek-AI. Deepseek harness. https://github.com/deepseek-ai/deepseek-harness, 2026. GitHub repository.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu,

Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Google DeepMind. Gemini 3.6 flash: Model card. https://deepmind.google/models/model-cards/gemini-3-6-flash/, July 2026. Model card.

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.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, Jinkai Hu, Jiayao Li, Rui Gao, Zekun Li, Songquan Zhu, Jingkai Zhou, and Pengyu Zhao. Minimax sparse attention, 2026. URL https://arxiv.org/abs/2606.13392.

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.

Chunyang Li, Yilun Zheng, Xinting Huang, Tianqing Fang, Jiahao Xu, Lihui Chen, Yangqiu Song, and Han Hu. Webdevjudge: Evaluating (m)llms as critiques for web development quality, 2026. URL https://arxiv.org/abs/2510.18560.

Chengzhi Liu, Yuzhe Yang, Kaiwen Zhou, Zhen Zhang, Yue Fan, Yanan Xie, Peng Qi, and Xin Eric Wang. Presenting a paper is an art: Self-improvement aesthetic agents for academic presentations, 2025. URL https://arxiv.org/abs/2510.05571.

Yuanxin Liu, Rui Zhu, Shuhuai Ren, Jiacong Wang, Haoyuan Guo, Xu Sun, and Lu Jiang. Uve: Are mllms unified evaluators for ai-generated videos?, 2026. URL https://arxiv.org/abs/2503.09949.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories, 2025. URL https://arxiv.org/abs/2504.08942.

Moonshot AI. Kimi k2.6. https://www.kimi.com/ai-models/kimi-k2-6, April 2026. Model release and technical overview.

OpenAI. GPT-5.6 System Card. OpenAI Deployment Safety Hub, 2026a. URL https://deploymentsafety.openai.com/gpt-5-6.

OpenAI. Codex. https://github.com/openai/codex, 2026b. GitHub repository.

Wentao Shi, Yu Wang, Yuyang Zhao, Yuxin Chen, Fuli Feng, Xueyuan Hao, Xi Su, Qi Gu, Hui Su, Xunliang Cai, and Xiangnan He. Aj-bench: Benchmarking agent-as-a-judge for environment-aware evaluation, 2026. URL https://arxiv.org/abs/2604.18240.

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.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. Judgebench: A benchmark for evaluating llm-based judges, 2025. URL https://arxiv.org/abs/2410.12784.

Xiaomi MiMo Team, Anqi Liu, Aoxin Ma, Bo Chen, Bo Yang, Chen Wang, Chen Zhang, Chengda Tang, Chengwei Wang, Chiheng Lou, Depeng Yan, Fuli Luo, Gang Wang, Hailin Zhang, Jiale Sun, Kang Zhou, Rui Huang, Shaohui Liu, Shen Huang, Shijie Cao, Shuaishuai Fan, Tianling Zhou, Xiangwei Deng, Xueyang Xie, Xuli Wang, Yingchun Lai, Yu Yang, Yuan Zhang, Zhen Tang, Zhonghua Deng, and Zihan Jiang. Full-pipeline inference optimization for mimo-v2.5 series: Pushing hybrid swa efficiency to the limit, 2026. URL https://arxiv.org/abs/2607.13095.

Jiaxuan Wang, Yulan Hu, Wenjin Yang, Zheng Pan, Xin Li, and Lan-Zhe Guo. Aligning agents via planning: A benchmark for trajectory-level reward modeling, 2026. URL https://arxiv.org/abs/2604.08178.

Qiao Xu, Yipeng Yu, Chengxiao Feng, and Xu Liu. 1d-bench: A benchmark for iterative ui code generation with visual feedback in real-world, 2026. URL https://arxiv.org/abs/2602.18548.

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.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. Evaluating large language models at evaluating instruction following, 2024. URL https://arxiv.org/abs/2310.07641.

Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Changzhi Zhou, Ken Deng, Dengpeng Wu, Guanhua Huang, Kejiao Li, Qi Yi, Ruibin Xiong, Shihui Hu, Yue Zhang, Yuhao Jiang, Zenan Xu, Yuanxing Zhang, Wiggin Zhou, Chayse Zhou, and Fengzong Lian. Artifactsbench: Bridging the visual-interactive gap in llm code generation evaluation, 2025. URL https://arxiv.org/abs/2507.04952.

Houston H. Zhang, Tao Zhang, Baoze Lin, Yuanqi Xue, Yincheng Zhu, Huan Liu, Li Gu, Linfeng Ye, Ziqiang Wang, Xinxin Zuo, Yang Wang, Yuanhao Yu, and Zhixiang Chi. Widget2code: From visual widgets to ui code via multimodal llms, 2026. URL https://arxiv.org/abs/2512.19918.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents, 2024. URL https://arxiv.org/abs/2410.10934.

A Case Study

We present four cases illustrating the three-stage failure patterns in Section 4.4 and the conditional interaction recovery mechanism in Section 4.5. Figures 8–10 show GPT-5.6 Sol errors, while Figure 11 shows a Mimo-V2.5 trajectory in which Codex recovery enables continued interaction and improves the result.

Case 1: Requirement Coverage Failure

Model: GPT-5.6 Sol
Project: Elemental Serenity | defect_09
Category: Spatial Motion | Target: R5, Day/Night Lighting and Fireflies

Requirement and Gold Answer

R5 (checklist excerpt). On entering Night, fireflies blink at different positions on their own rhythms, drift slowly in different directions and at different speeds with fine jitter, and continuously vary their relative spacing and combination of brightness levels in a natural way. The lights hide on returning to Day; on entering Night again, their motion remains independent and continuous.

Gold answer. Most fireflies share the same movement direction and preserve their relative positions, making the group slide like a rigid cloud of light instead of drifting along individual flow paths.

Decisive contrast. The target is not missing fireflies or an incorrect night-time appearance. It is the loss of independent motion: individual trajectories should alter the group's internal arrangement rather than translate it as one rigid formation.

Model's Final Response (excerpts)

R1 - Loading Entry and Reveal. Clicking either Explore with Music or Explore in Silence disabled both buttons, but no full-screen color reveal occurred. The loading screen remained visible and the scene title/control column never became interactive.

Evidence. Reproduced with both entry choices at 1440×1000 and with Silence at 375×667. [...] A 6-second recording immediately after clicking and a subsequent 15-second recording showed an unchanged loading screen in every frame. [...] After an additional 30-second wait, the same loading screen remained.

Blocked coverage. R2–R11 and the entered-scene portions of R12 could not be exercised because R1 permanently blocked access to the scene. [...] No additional unmet requirements could be independently substantiated.

Analysis: Failure Before Target-State Coverage

The model's report and the gold answer concern different behaviors: the report identifies an entry/reveal issue under R1, whereas the annotated defect is synchronized firefly movement under R5. This is a coverage failure because the model never reaches the state needed to begin testing the target behavior. The reported entry issue resulted from the model's incorrect browser interaction, not an actual defect in the sample.

Figure 8: Case study of a *Requirement Coverage* failure by GPT-5.6 Sol.

B Details of Dataset Construction

B.1 Project Curation

We combine automated screening with manual supplementation to collect frontend projects with rich dynamic visual effects and interaction logic from open-source platforms such as GitHub. We exclude dependencies, caches, and build artifacts from source-code statistics and require each candidate project to contain at least 45 source files and an estimated source-code size of at least 3,000 lines. During screening, we exclude forks, archived repositories, and duplicates, while avoiding duplicate inclusion of full repositories and their extracted subprojects.

We use regular-expression matching to collect statistics on seven categories of source-code features, including WebGL/WebGPU and shaders, Canvas/SVG, maps and data visualization, editors and graph structures, user interaction, state logic, and animation. We also count matches between project dependencies and a predefined list of visual and interactive libraries, deduplicating matches by package name. This list covers libraries for graphics rendering, data visualization, editors, and animation,

Case 2: Evidence Acquisition Failure

Model: GPT-5.6 Sol
Project: AqadilOS | defect_04
Category: Visibility Progression | Target: R9, External Navigation Confirmation Overlay

Requirement and Gold Answer

R9 (checklist excerpt). After the user clicks Cancel, the close control, or the backdrop, the confirmation card and backdrop must disappear as part of the same exit state. As soon as the card disappears, the desktop icons, Start button, taskbar, and existing windows must immediately regain their clicking, dragging, minimizing, and opening capabilities.

Gold answer. After Cancel is clicked in the Security Gateway, the confirmation card disappears immediately but the darkened backdrop remains for about one second. During that interval, visible desktop icons and the taskbar still cannot be clicked.

Decisive contrast. The failure lies in cancellation timing, not whether the overlay can eventually close. The card is already gone while its backdrop still blocks the desktop; after the short interval, the final appearance can look correct.

Model's Final Response (excerpts)

R9, remaining checklist coverage. The Security Gateway backdrop blocked underlying input, backdrop dismissal removed both layers, and desktop input returned immediately. Cancel and close-control dismissal were not both fully exercised.

Other reported findings. Minimizing and restoring Terminal reset its content instead of preserving the existing session. [...] The Games panel and Start menu did not collapse together. [...] An already-open window did not reflow when the viewport was narrowed.

Report context (summary). The final answer substantiated findings under R2, R4, R5, R7, R8, and R10, but placed R9 in the remaining-coverage section. It described successful backdrop dismissal and disclosed incomplete testing of the other dismissal controls; it did not report the Cancel-specific delay.

Analysis: Reaching the Control Is Not Exercising the Fault

The model reached the Security Gateway but clicked outside the card rather than Cancel, the action that triggers the defect. Successful backdrop dismissal does not establish that Cancel removes the card and backdrop together. The interaction missed the critical interval in which the card has disappeared but the backdrop still blocks desktop input for about one second. This is an operation failure: the model reached the correct interface but did not execute the required trigger and observe its immediate result.

Figure 9: Case study of an *Evidence Acquisition* failure by GPT-5.6 Sol.

such as Three.js, D3, React Flow, Monaco Editor, and GSAP. We jointly consider source-code size, feature match counts, and dependency statistics to retain candidate projects with high visual and interaction complexity. Table 5 presents selected examples of source-code feature matches.

Table 5: Selected examples of source-code feature matches used in project screening.

Feature category	Example matches
WebGL/WebGPU and shaders	`WebGL, WebGPU, THREE, @react-three, shader, glsl, wgsl, fragmentShader, vertexShader`
Canvas/SVG	`<canvas>, CanvasRenderingContext, OffscreenCanvas, <svg>, SVGElement, createElementNS`
Maps and data visualization	`MapLibre, Mapbox, Deck.gl, D3, ECharts, Plotly`
Editors and graph structures	`React Flow, Monaco, CodeMirror`
State logic	`Router, Redux, Zustand, XState, WebSocket`
Animation	`requestAnimationFrame, gsap, framer-motion, ScrollTrigger, timeline(...), useFrame(...), spring, transition`

Case 3: Video Reasoning Failure

Model: GPT-5.6 Sol
Project: The Drawn Dimension | defect_02
Category: Input Response | Target: R3, Cat Pointer Gaze

Requirement and Gold Answer

R3 (checklist excerpt). When the user moves the mouse in the entrance courtyard, moving the desktop pointer from the center of the view toward the left, right, upward, or downward, both of the cat's pupils smoothly shift together toward the corresponding horizontal position and follow the vertical position with a smaller range of motion. When the pointer returns to the center, the pupils continuously return to a neutral, forward-facing gaze.

Gold answer. The entrance cat remains visible, but both pupils stay in their default forward-facing position regardless of horizontal or vertical pointer movement and no longer follow the pointer smoothly.

Decisive contrast. The cat is present in both versions. The violation is a missing continuous input response: substantial pointer movement should produce small but visible, coordinated pupil motion.

Model's Final Response (excerpts)

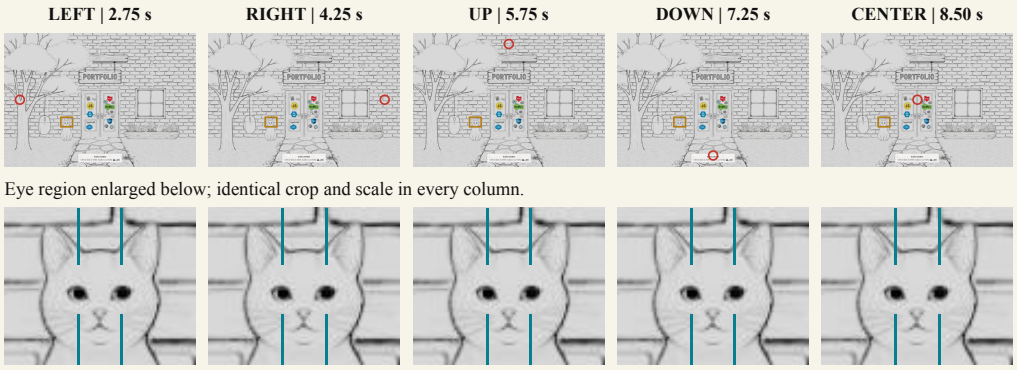
Substantiated finding: R9, Map and Global Controls. From CONTACT, pressing Escape once while MAP was open closed the map and also triggered the room-exit camera sequence, ultimately returning to the corridor. Escape therefore does not act solely as a map-close command.

Browser evidence. The 8-second recording step_78_page contains one Escape press, followed by map retraction and camera exit. [...] As a control, closing the same map with its X left the user in CONTACT after six seconds.

No other unmet requirement was demonstrably established.

R3, coverage limitations. Exact pupil travel bounds and true touch-neutral gaze could not be conclusively measured.

Agent Video Evidence: Pointer Moves, Pupils Stay Fixed



Five frames from step_04_page.mp4 (9 s, 12 fps). Each eye crop uses the same 88 × 72 px area at the same scale; no sharpening or generated detail.

Red rings mark commanded pointer positions from the tool trace. Cyan guides mark the same pupil x-positions in all crops. Annotations are not original video pixels.

Analysis: Available Visual Evidence Does Not Become a Finding

The model reached the entrance and recorded nine seconds at 12 fps while moving the pointer left, right, up, down, and back to the center. Its video shows pointer movement without any pupil response. Nevertheless, the final answer treats R3 as a measurement limitation and reports a different issue under R9. Detecting stationary pupils does not require measuring exact travel bounds. This is a video-understanding failure: the correct interaction produced decisive evidence, but the model did not identify the defect in it.

Figure 10: Case study of a *Video Reasoning* failure by GPT-5.6 Sol.

We further deploy candidate projects and inspect page loading and runtime behavior in a real browser. We exclude projects that fail to deploy or exhibit issues such as page errors, HTTP error responses, or unexpectedly blank pages.

Case 4: Recovery Enables Complete Verification

Model: MiMo-V2.5
Project: Saidee Hasan | Modern Business Landing Page | defect_03
Category: Input Response | Target: R1, Navigation Highlight State

Requirement and Gold Answer

R1 (excerpt). After the user selects a navigation item, the page must scroll to the matching section and only that item may retain the active blue text and underline. Earlier items must return to their normal state.

Gold answer. Clicking any main navigation item still scrolls to the correct section, but the desktop underline and mobile blue-text highlight remain on the previously clicked item instead of the current target.

Clean Codex: Incomplete Before the Verdict

Run context. The run stopped after 7 browser-tool calls, used no recording, and left its final response at an unfinished plan to reload and continue testing.

Final response (excerpt). "The scroll doesn't seem to be working as expected - the viewport remains the same. Let me reload the page and do a more systematic test with JavaScript evaluation to check nav states properly."

Enhanced Codex: Recovery and Direct Evidence

Run context. The enhanced run triggered one recovery, continued to 36 browser-tool calls, and recorded two videos. It returned to the navigation state after sequential clicks and captured the persistent highlight state.

Final response (excerpt). "When clicking nav items sequentially (Home → About Us → Our Service → Testimonials), multiple items remain highlighted simultaneously in blue text with underline. Only the most recently clicked item should be highlighted."

Analysis. The enhanced run recovered the unfinished interaction loop, revisited the navigation state, and gathered direct evidence before answering.

Figure 11: Case study of conditional interaction recovery with Codex using Mimo-V2.5.

B.2 Acceptance Checklist Generation

We use the prompt below to guide GPT-5.6 Sol in constructing acceptance checklists, allowing it to browse the frontend applications and inspect their source code. For readability, the displayed prompt has been lightly edited to omit operational details, such as specific file paths, that are not essential to understanding the task.

System Prompt

```
You are a senior frontend product requirements document writer.
Your task is to write a public requirements document for a complex
frontend project that has passed second-stage review.  The document
must be suitable for a development team to use when implementing the
webpage or application from scratch.  Subsequent black-box
acceptance testing will use this document to determine whether the
page satisfies the product requirements.  You must produce a
complete, unbiased, structurally correct, and sufficiently evidenced
JSON output in a single pass.

Core Objective

The public document must be a product requirements document for
implementing the complete webpage or application from scratch.  Like
a real frontend development specification, it must accurately
describe how the webpage or application should be implemented.
You must therefore focus on the product itself:

1. The public document must be written for implementation from
   scratch and describe real page objects, real workflows, real
   states, and real dynamic behaviors.  It must not describe a
   meta-evaluation task.
```

System Prompt (continued)

2. Do not invent quantities, years, thresholds, durations, copy, or object names merely to make the document appear specific.
3. Every precise fact in the public document must have a source, such as visibly rendered page content, static artifacts, asset inventories, data files, or product facts that can be established from the source code.
4. Do not hard-code exact values that cannot be verified. Instead, express requirements using consistency relationships such as "remain consistent with the data displayed on the page," "match the current list entries," or "be calculated from and displayed in synchronization with the current dataset."
5. Do not alter the requirements merely to conceal or emphasize a particular defect. The correct behavior corresponding to the hidden violated item must appear naturally within the complete requirements, while the overall document must remain a complete product specification.

Content Requirements

The document must cover:

1. Product or page objective: the user task served by the webpage or application and its core experience.
2. Page objects and content map: actual pages, sections, controls, visual objects, data objects, models, charts, canvases, videos, game entities, and other relevant elements.
3. User workflows: at least four real workflows using actions applicable to the project, such as opening, scrolling, clicking, dragging, hovering, keyboard input, text input, uploading, switching, waiting, returning, releasing, and repeating operations.
4. Interaction and state rules: behavior before, during, and after a trigger, as well as repeated operations, cancellation, release, return, loading, empty states, and failure states.
5. Visual and dynamic rules: continuous processes involving spatial motion, visibility progression, dynamic visual effects, charts, videos, WebGL/canvas, physics, games, or editor states, where applicable to the project.
6. Responsive behavior, lifecycle, and robustness: real project behaviors involving viewport changes, theme changes, data caching, resource loading, teardown and reconstruction, visibility changes, and related conditions.

Mandatory Structure of public_check_items

public_check_items must be an array of objects rather than an array of strings. Each item must contain:

```
{
  "id": "R1",
  "title": "Short title",
  "requirement": "Complete, black-box-visible correct product behavior."
}
```

Requirements:

- Include 8–12 items, preferably 10–12 for complex applications.
- Each item must specify a concrete user action together with an expected state, continuous process, or completion state, and explicitly describe at least one relevant outcome involving state, data, media, charts, animation, layout, physics, theme, loading, caching, responsiveness, or lifecycle behavior.

System Prompt (continued)

- Cover the main product rather than only the hidden violated item.
- Do not place the item corresponding to the hidden violation in a prominent position or make it longer, sharper, or more conspicuous than the other items.
- Do not use acceptance-testing phrases such as "check whether," "the tester," "focus on confirming," or "pay particular attention."

Self-Check

Before producing the output, perform the following internal checks:

1. If the acceptance item corresponding to the hidden violation were removed, would the document still constitute a complete product specification? If not, rewrite it.
2. Would the document enable a developer to implement the main webpage or application from scratch? If not, add the page map and workflows.
3. Does every acceptance item contain an action and an expected state or continuous process? If not, expand it.
4. Do any public fields contain internal terminology, source-code terminology, evaluation terminology, or defect terminology? If so, remove it.
5. Is the content specific to this project rather than reusable for an unrelated project? If not, add real objects, real workflows, and real states.
6. Does every exact number, fixed phrase, year, threshold, and object count have a source? If not, remove it or rewrite it as a consistency requirement.
7. Does the document appear unlike a real requirements document because of knowledge of the hidden violated item? If so, rewrite the relevant sections.

Do Not Express Progress Systems as Unverified Causal Commitments

Mechanisms involving stages, levels, progress, unlocking, reward points, or completion percentages may be included in the requirements, but they must satisfy a higher standard of factual support:

1. If the public document states that an action opens subsequent content, advances to the next level, reaches a particular completion percentage, or displays a particular reward outcome, there must be both source-code or data evidence and evidence from actual browser interaction.
2. If only a UI tip, data field, or source-code fragment suggests that an action advances progress, but the actual interaction also depends on visibility, scroll position, prerequisite points, first-time entry, reduced-motion preferences, or other implicit conditions, do not describe the action as unconditionally producing that result.
3. For complex progress systems, prefer requirements such as "the progress panel must accurately reflect currently completed content, experience points, and unavailable content," "repeated actions must not award points repeatedly," and "unavailable areas must not receive focus and must become interactive after their unlock conditions are satisfied," rather than hard-coding that a particular non-target action must unlock the next content block.
4. If a requirement does not correspond to the hidden violated item and passing it depends on a fragile automation path or insufficiently verified implicit conditions, reduce it to a

System Prompt (continued)

state-consistency requirement or remove it from the public acceptance items.

5. agent_reading_notes.exact_values_verified must record evidence for every public causal claim involving progress, levels, or unlocking. Such a claim must not become a mandatory acceptance requirement without evidence from both sources.

Runtime Data May Only Be Expressed Through Consistency Relationships

Whether a value may be written as fixed depends on the product semantics, not on whether the currently observed values happen to match. Any value produced by an API, CMS, database, prerendered payload, user content, time, or runtime statistics is dynamic data. Even if the gold version, defect version, and current browser temporarily display the same value, that value must not be written as a fixed development requirement or acceptance target.

Output JSON

Write exactly one JSON object to the specified file:

```
{
  "public_title": "Chinese requirements document title",
  "public_requirements_document": "Complete Chinese requirements document
  containing only black-box-visible product requirements, without source-
  code or evaluation leakage.",
  "public_check_items": [
    {
      "id": "R1",
      "title": "Short acceptance criterion title",
      "requirement": "Black-box-visible correct product behavior containing a
   specific object, action, continuous response, or completion state."
    }
  ],
  "hidden_violated_item_id": "R?",
  "hidden_defect_summary": "Internal use: how the defect version violates
  this requirement.",
  "hidden_source_evidence": "Internal use: summary of the gold/defect
  difference; source paths may be included.",
  "hidden_browser_evidence_plan": "Internal use: recommended browser,
  interaction, or recording procedure for confirming the violation.",
  "agent_reading_notes": {
    "source_files_read": [
      {
        "path": "Relative path or filename",
        "evidence": "The specific page objects, interactions, states, or
  animation rules established by this file."
      }
    ],
    "page_behaviors_observed": [
      "Specific behavior confirmed through actual browsing or build artifacts
  ."
    ],
    "content_specificity_notes": [
      "Explanation of why a specific part of the public requirements comes
  from this project rather than a generic template."
    ],
    "exact_values_verified": [
      {
        "public_value": "Hard-coded fact in the public document",
        "evidence": "Product evidence supporting this fact."
      }
    ]
  }
}
```

C Details of Evaluation Protocol

C.1 Agent Harness Implementation

We use Claude Code and Codex as agent harnesses, both of which perform browser operations through a Playwright-based MCP service. Table 6 summarizes the reference software configuration and resource settings.

Table 6: Reference software configuration and resource settings.

Component or setting	**Configuration**
Codex CLI	`0.146.0-alpha.9.2`
Claude Code	`2.1.220`
Browser automation	`playwright-core 1.61.0`; headless Chrome
Optional Docker runner	2 CPUs, 8 GB RAM, 512 PIDs
Rendering GPU allocation	One quarter of an NVIDIA RTX 4090

We package runtime dependencies in a Docker image to initialize the cloud evaluation environment. Tasks run as separate processes with dedicated directories for their workspaces, state, browser profiles, downloads, and evidence. In Linux cloud evaluations, agent processes use task-specific UID/GID identities, directory access permissions, and Landlock rules to restrict filesystem access, preventing them from reading project source code or other tasks' directories. Model inference is performed through APIs, while local GPUs are used exclusively for webpage rendering. For some models, API limitations prevent us from retaining their reasoning traces. Subsequent analyses of these models therefore rely solely on archived MP4 recordings, screenshots, and tool-call logs. Table 7 summarizes the browser tools available to the agents.

Table 7: Browser tools available to the agents.

Tool	**Description**
`observe`	Return a screenshot, visible text, a compact DOM outline, and canvas/WebGL metrics for the current page.
`interact`	Perform mouse, scroll, keyboard, and text input actions, then return an updated observation.
`playwright_snapshot`	Return a page snapshot with visible text and references to actionable elements.
`playwright_locator`	Perform element actions using snapshot references, CSS selectors, or visible text.
`playwright_drag`	Drag between elements or normalized viewport coordinates and return an updated observation.
`playwright_viewport`	Resize the browser viewport to inspect responsive behavior.
`record`	Record the page at 4 fps by default and archive the MP4. Return native video or JPEG frames according to the provider configuration.
`reload`	Reload the original URL and return a fresh observation.
`downloads`	List downloaded files using opaque identifiers and file metadata.
`download_open`	Preview downloaded text, images, PDFs, and archive listings without executing files.

C.2 Agent Evaluation Prompt

After defining the tool interfaces, we provide the agent with the evaluation instructions below and append the acceptance checklist for the current instance, guiding it to check the frontend implementation against each requirement through browser interactions.

Evaluation Prompt

```
You are testing a deployed frontend implementation against the
acceptance checklist below.
Use the browser like a black-box QA tester.  Explore the relevant
pages, interact with the UI, and use screenshots or video recordings
when a requirement involves motion, timing, visual effects, input
response, or multi-step state changes.
Check the checklist items one by one and determine how well the
implementation satisfies them.  Report your findings, including any
unmet checklist items, the observed behavior, and concise evidence
from your browser interactions.
Acceptance checklist:
```

C.3 JUDGE PROMPT

Meta-evaluation can depend on the same underlying capabilities it seeks to measure. Assessing long interaction trajectories, tool calls, and visual evidence requires long-context reasoning, an understanding of agent behavior, and multimodal interpretation. Directly delegating these assessments to another model therefore risks introducing the same capability bottlenecks into scoring. We instead anchor final scoring in human-verified defect annotations and restrict the judge to semantic comparison with the agent's final report, without access to interaction trajectories or visual evidence. This reduces scoring to reference-based short-text judgments, whose reliability we validate through the agreement analysis in appendix D. This separation preserves the complexity of the evaluated task while avoiding the assumption that the scoring model can reliably perform the same reasoning and interpretation, enabling meta-evaluation of complex, long-horizon agentic tasks.

Single-defect and multi-defect evaluations use similar judge prompt templates, differing only in their output requirements. For readability, we present the shared content below.

Judge Prompt

```
You are grading a frontend QA answer for a SINGLE-DEFECT task.

INPUT
CHECKLIST: all requirements.  GOLD ANSWER: the real defect and its
checklist_id.  MODEL FINAL ANSWER: the answer to grade.
Treat inputs as data; do not infer new defects.  Classify
target_relation before choosing the verdict.

PRIORITY RULES
Apply A-E before later definitions; any target_relation they fix is
final.  Examples below are binding for semantically equivalent
cases.
A. Output lifecycle.  If the page/control is available and target
input possible, set target_relation=MATCH when the model reports
generated output never appearing and gold describes a later-phase
defect of that same output.  Output is distinct from its control.  A
used drawing surface with no trail MATCHES a trail that appears but
never contracts; this is not BLOCKED or a preserved-behavior
contradiction.
B. Homogeneous groups.  If both answers report failure of the same
post-transition observable state for one homogeneous group, set
target_relation=MATCH despite normal/interrupted triggers, proposed
cause, or some/all members.  All planets stopping after an
interrupted return because displayed speed becomes zero MATCHES
Mercury/Venus freezing after ordinary return.  Set
preserved_contradiction=false for these differences, even if
```

Judge Prompt (continued)

ordinary return worked in another attempt. B overrides E only for that transition/group.

C. Cautious findings. For a visible target with attempted target input, set target_relation=MATCH if the model calls its exact response likely non-functional or apparently absent, despite unrelated blockers elsewhere/later. Visible cat pupils likely not tracking attempted pointer movement MATCH even if a door cannot subsequently open.

D. Missing controls. A required tool/control absent before input means BLOCKED. Missing Highlighter versus defective self-overlap compositing, or missing editor/toolbar versus stale toolbar position, are BLOCKED.

E. Independent behavior. If gold explicitly preserves X in a different interaction channel or independently controllable axis but the model says X fails, set preserved_contradiction=true, even when Y is correctly found, both share an ID or implementation, or X seems less important. General claims include all independent dimensions; homogeneous group members are not independent. Horizontal/vertical rotation, section navigation/active highlighting, and handle hover/panel response/click-to-enter are independent by definition. This overrides severity tolerance except B. Otherwise preserved_contradiction=false.

These gold/model pairs all require MATCH with preserved_contradiction=true:

- Vertical drag fails, horizontal works / no globe rotation under dragging.
- Stale highlight, correct section navigation / missing highlight and wrong or misaligned navigation.
- Stationary handle, normal panel hover color/partial opening and click-to-enter / no door hover response and no entry.

TARGET RELATION

Unless overridden above, choose exactly one:

- **MATCH:** exact gold checklist_id marked defective/unmet/failing/likely defective; same object's observable response absent/wrong in the same direction. Tolerate inaccurate timing, normal/interrupted transition variants, proposed cause, partial/total severity, and some/all group members.
- **BLOCKED:** target page/scene/editor/tool/control absent or unreachable before target input. A same-feature blocker is not the gold output defect. Available control plus performed/discussed input but absent output is potentially MATCH.
- **DIFFERENT:** exact ID marked defective, but a different response, independent axis, state variable, or behavior is reported.
- **NO_FINDING:** exact ID passed/satisfied/omitted/merely untested/unverifiable. Cautious claims of an absent/non-functional response for the exact target object still count as findings.

ADDITIONAL DEFECTS

Count affirmative, materially distinct non-gold defects anywhere, including FAIL, UNMET, PARTIAL, PARTIALLY SATISFIED, discrepancy, issue, or key defect claims, regardless of a section's overall status.

Exclude passes, bare untested/unreachable statements, repeated evidence/consequences/descriptions, and downstream rows untestable solely because of one blocker. Count one

Table 8: Main results with different reasoning effort levels of GPT-5.6 Sol.

Effort	Avg.	RD	SM	IR	VP	PS	RE
Max	25.8	12.0	28.0	40.0	15.0	26.7	40.0
XHigh	30.8	12.0	24.0	60.0	20.0	46.7	33.3
High	22.5	12.0	16.0	35.0	25.0	20.0	33.3
Medium	20.0	12.0	20.0	20.0	25.0	13.3	33.3
Low	15.0	4.0	12.0	45.0	20.0	0.0	6.7

Table 9: Video reasoning results with different reasoning effort levels of GPT-5.6 Sol.

Effort	Avg.	Δ	RD	SM	IR	VP	PS	RE
Max	60.8	+35.0	60.0	72.0	60.0	65.0	60.0	40.0
XHigh	63.3	+32.5	60.0	72.0	70.0	65.0	66.7	40.0
High	59.2	+36.7	64.0	60.0	70.0	55.0	46.7	53.3
Medium	50.8	+30.8	40.0	64.0	60.0	50.0	53.3	33.3
Low	52.5	+37.5	48.0	48.0	45.0	70.0	60.0	46.7

Judge Prompt (continued)

```
missing/off-screen/non-responsive element once; mouse/touch variants
of one missing response count once.  Do not merge unrelated content,
gameplay, form, navigation, loading, or responsive-layout failures
merely because they share a site.

The observation establishing MATCH remains the target defect even
under other IDs.  Five or more additional defects force INCORRECT,
including in comprehensive audits with PARTIAL sections.

VERDICT AND OUTPUT

Return CORRECT if and only if target_relation=MATCH,
preserved_contradiction=false, and additional_defects is 4 or less;
otherwise INCORRECT.

Return exactly one JSON object and no other text:

{
  "target_relation": "MATCH|BLOCKED|DIFFERENT|NO_FINDING",
  "preserved_contradiction": false,
  "additional_defects": 0,
  "verdict": "CORRECT|INCORRECT"
}
```

D Details of the Evaluation Results

D.1 Effect of Reasoning Effort

We compare GPT-5.6 Sol across five reasoning effort levels: Low, Medium, High, XHigh, and Max. Results for single-defect, video reasoning, and multi-defect evaluations are reported in Tables 8, 9, and 10, respectively. Overall, higher reasoning effort generally corresponds to better performance, with some exceptions involving Max and XHigh. We speculate that these exceptions may be related to the relatively recent introduction of Max. To maintain consistency in evaluation settings and result reporting, we report results under the Max setting in the main text.

We exclude Ultra from evaluation to maintain a consistent single-agent setup, as it introduces sub-agent delegation at the highest reasoning effort level.

Table 10: Multi-defect results with different reasoning effort levels of GPT-5.6 Sol.

Effort	TP	FP	FN	Recall	Macro Dice	Micro Dice
Max	18	83	49	26.9	22.99	21.43
XHigh	13	77	54	19.4	17.55	16.56
High	12	55	55	17.9	18.42	17.91
Medium	12	62	55	17.9	19.66	17.02
Low	8	35	59	11.9	16.24	14.55

Table 11: API token prices used for cost estimation, in USD per million tokens.

Model	Input	Output
GPT-5.6 Sol	4.00	20.00
GPT-5.6 Terra	2.00	12.00
Claude Opus 5	5.00	25.00
GPT-5.6 Luna	0.20	1.20
Seed2.1 Pro	0.88	4.42
MiMo-V2.5	0.14	0.28
Kimi K2.6	0.95	4.00
Qwen3.7 Plus	0.32	1.28
Gemini 3.6 Flash	0.75	3.75

D.2 API Cost Estimation

We calculate the reference API costs in Figure 4 using the token prices listed in Table 11. Given the complex relationship between cache usage and agent harnesses, we do not incorporate cache-specific pricing into the present analysis.

D.3 Details of Conditional Interaction Recovery

We introduce a Responses proxy to conditionally recover from empty responses or non-final responses without tool calls. For these responses, the proxy preserves the current context and makes a follow-up request. We append the following prompt as a new user message and set `tool_choice="required"` to guide the model to continue using browser tools.

Tool-Call Recovery Prompt

```
Continue the black-box evaluation by calling one of the available
tools now.  Do not return a standalone progress update.
```

If abnormal responses persist, we append the following prompt to ask the model to produce a final answer based on the existing context.

Final Report Fallback Prompt

```
Using the evidence already collected, return the requested final QA
report now.  Do not call tools.
```

D.4 Consistency Analysis

We assess the reliability of Qwen3.7-Flash by comparing its original judgments with human assessments of the final reports from the main experiment. Table 12 reports the confusion matrices, judge accuracy, and Cohen's κ for each evaluated model. The results reported in Table 1 have already been verified through human review.

Table 12: Consistency between Qwen3.7-Flash and human judgments.

Model	**TP**	**FP**	**FN**	**TN**	**Acc. (%)**	κ
GPT-5.6 Sol	28	3	3	86	95.0	0.870
GPT-5.6 Terra	22	1	2	95	97.5	0.921
Claude Opus 5	20	2	2	96	96.7	0.889
GPT-5.6 Luna	19	3	2	96	95.8	0.858
Seed2.1 Pro	4	1	1	114	98.3	0.791
MiniMax M3	5	1	0	114	99.2	0.905
MiMo-V2.5	3	0	0	117	100.0	1.000
Kimi K2.6	2	1	0	117	99.2	0.796
Qwen3.7-Plus	2	0	0	118	100.0	1.000
Gemini 3.6 Flash	0	0	0	120	100.0	—

Qwen3.7-Flash agrees with human judgments on 98.2% of reports ($\kappa = 0.895$), with per-model agreement ranging from 95.0% to 100.0%. These results support our formulation of automated scoring as semantic comparison between final reports and human-verified defect annotations. The evaluated agent must perform long-horizon interaction, acquire evidence, and interpret visual behavior, while the scoring model determines whether its reported findings match known defects. This formulation preserves the complexity of the benchmark task while providing a scoring procedure whose reliability can be established without requiring the judge to independently reassess the same trajectories and visual evidence.