WebRISE: Requirement-Induced State Evaluation for MLLM-Generated Web Artifacts

Yuxin Meng[1,2*] Yuhan Suo[1,2*] Junjie Wang[1*] Yuhan Sun[3*]
Yiyao Yu[1] Ruixu Zhang[1] Ruining Hu[4] Yubin Wang[2] Shouwei Ruan[5]
Bin Wang[2] Yuxiang Zhang[2†] Yujiu Yang[1†]

[1]Tsinghua University [2]Huawei Noah's Ark Lab [3]East China Normal University
[4]Tongji University [5]Institute of Artificial Intelligence, Beihang University

{meng-yx25, suoyh25}@mails.tsinghua.edu.cn, joel0495@asagi.waseda.jp, yang.yujiu@sz.tsinghua.edu.cn
https://iigroup.github.io/WebRISE

Abstract

Existing benchmarks for MLLM-generated web artifacts assess interaction through local evidence and miss the requirement-induced states and transitions that determine whether a page works. We introduce WebRISE, which compiles task requirements into Interaction Contract Graphs (ICGs) of observable states, user-intent transitions, and DOM/visual assertions for implementation-agnostic browser execution. WebRISE spans 442 tasks across five input modalities (Text, Markdown, Sketch, Image, Video), with 5,495 transitions and 5,271 requirement checks that separate user-stated functions from implicit product-level constraints. Across 14 MLLMs, even the strongest model reaches only 65.6% transition validity and 66.3% requirement coverage, and visual quality is no proxy for behavior (Qwen3.6-35B-A3B on Markdown: V=80.8 yet T=15.5). Video gives the strongest interaction signal (+10.6 pp implicit coverage over Text), while implicit constraints persist; defect injection shows ICG-based scoring detects state errors at 2–16× the rate of checkpoint-style evaluation.

1 Introduction

Multimodal large language models (MLLMs) are increasingly asked to generate executable web artifacts from multimodal specifications, including textual requirements, Markdown structures, sketches, screenshots, and interaction videos (Yin et al., 2024; Si et al., 2025; Chen et al., 2026; Liu et al., 2026). This shift raises a basic benchmark question: *when is a generated webpage usable, rather than merely visually plausible?* In real use, a page can fail even when the expected controls are present: a filter may leave the item list unchanged, or a cart update may not propagate to the total price. Evaluating MLLM-generated web artifacts therefore requires testing requirement-implied state transitions and state-consistency constraints, rather than only initial appearance or isolated action outputs.

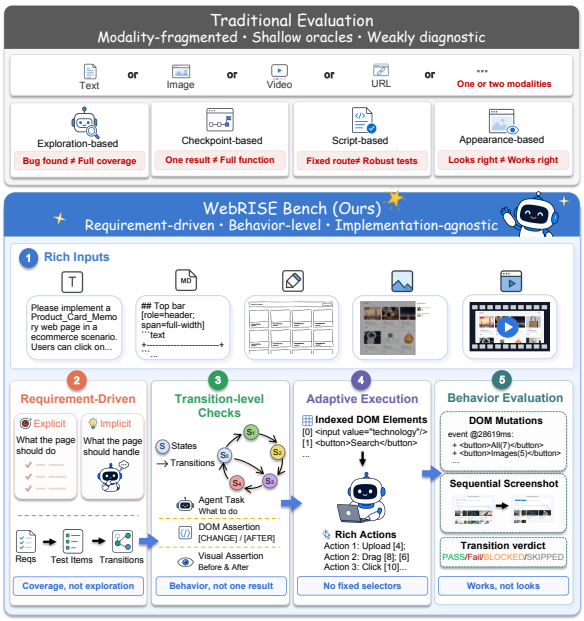

Figure 1: Overview of WebRISE. Top: representative prior evaluation protocols often rely on modality-fragmented inputs and local evidence, such as appearance, scripts, checkpoints, or open-ended exploration. Bottom: WebRISE evaluates generated web artifacts through a requirement-induced interaction contract: it supports five input modalities (❶), maps explicit and implicit requirements to test items and transitions (❷), defines DOM/visual transition checks (❸), executes them with a contract-guided agent (❹), and records transition-level verdicts with structured evidence (❺).

Recent benchmarks for web, UI, and artifact generation have moved beyond static visual fidelity by incorporating interaction evidence, such as dynamic screenshots and MLLM-as-a-judge checklists (Zhang et al., 2025), predefined scripts (Zhu et al., 2025), web-navigation agents (Lu et al., 2025b), real user requirements (Liu et al., 2026), and interaction videos (Chen et al., 2026). These efforts establish interaction as a central dimension of web generation evaluation. However, existing

*Equal contribution.
†Corresponding authors.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 12360–12385
October 24-29, 2026 ©2026 Association for Computational Linguistics

protocols still tend to operationalize interaction through local evidence rather than requirement-level state obligations. This creates two limitations. (i) **Event-centric evaluation**: screenshots, script steps, video trajectories, or expected-result checkpoints can verify whether a local action produces a response, but they do not explicitly define which requirement-induced states and transitions should be covered. (ii) **State-consistency gap**: a local response may pass even when the page violates cross-component, cross-view, or cross-step constraints, such as filter–pagination synchronization, count updates after deletion, or hidden-state preservation after navigation. In short, existing benchmarks make interaction observable, but not yet fully enumerable or attributable as a requirement-induced state space. Fig. 1 summarizes this contrast.

To address these limitations, we introduce `WebRISE`, a benchmark that evaluates MLLM-generated web artifacts as *requirement-induced observable state-transition conformance*. `WebRISE` derives a finite interaction contract from task requirements, consisting of observable UI states, user-intent transitions, and DOM/visual assertions, and tests whether a generated page conforms to this contract under browser execution. It is built on two design choices: *requirement-conditioned state modeling*, which represents each task as an Interaction Contract Graph (ICG), and *conformance-based diagnostic evaluation*, which links transition outcomes back to explicit requirements and implicit state-consistency constraints.

Concretely, `WebRISE` converts explicit and implicit requirements into Test Data Contracts and test items, then compiles them into an Interaction Contract Graph (ICG). ICG states are requirement-relevant observable UI configurations rather than full DOM snapshots, while transient behaviors such as loading, saving, debounce, and temporary disabled states are verified as transition-level DOM evidence. Each task is instantiated under Text, Markdown, Sketch, Image, and Video inputs, and models generate self-contained executable HTML pages. During evaluation, the ICG specifies *what* to verify, a contract-guided agent decides *how* to execute each transition, and a DOM/visual dual oracle verifies process evidence and user-visible outcomes. The resulting reports are aggregated into state-, transition-, and requirement-level diagnostics, including $S\%$, $T\%$, $Re\%$, $Ri\%$, and $R\%$.

We evaluate `WebRISE` on 442 tasks, 5 input modalities, and 14 representative models, and obtain three main findings. First, interactive web generation remains far from solved: even the strongest model, GPT-5.5, reaches only $T = 65.6\%$ and $R = 66.3\%$ under its best modality, leaving roughly one third of required transitions or requirement checks unsatisfied. Second, multimodal specifications improve interaction quality, with Video being the strongest modality: compared with Text, it improves T, R, and R_i by 8.8, 8.3, and 10.6 percentage points, respectively. Third, implicit state constraints remain a consistent bottleneck: explicit requirements are easier across models, and hard tasks are enriched with feedback, error, edge-state, and boundary-condition failures. As an additional evaluator sanity check, defect injection on GT-validated pages shows that ICG-based evaluation detects $16/25$ injected state-related defects, compared with $8/25$ under a broad checkpoint-style WebGen criterion and $1/25$ under a strict one.

Our contributions are threefold:

- We introduce `WebRISE`, a benchmark that reframes MLLM-generated web artifact evaluation as requirement-induced observable state-transition conformance, covering 442 tasks, five input modalities, and explicit/implicit requirement contracts.
- We develop a contract-guided evaluation protocol that represents each task with an Interaction Contract Graph, executes transitions with an adaptive browser agent, and verifies process and outcome evidence through DOM/visual oracles.
- We conduct a large-scale evaluation of 14 representative models, revealing that current systems remain far from solving interactive web generation, that Video provides the strongest interaction signal, and that implicit state constraints remain a major bottleneck.

2 Related Work

MLLM-generated web artifacts.

Multimodal large language models are increasingly moving from UI understanding and static code generation toward executable web artifact generation (Yin et al., 2024). Early UI-to-code, design-to-code, and sketch-to-code studies mainly evaluate whether models can recover layout, visual structure, and front-end code from textual or visual specifications (Si et al., 2025; Jain et al., 2019; Periasami et al., 2026). Recent work further expands this setting to automated functional testing (Zhu et al., 2025; Lu et al., 2025b), dy-

Benchmark	Verified Units	Main Verdict	Interact.	Vision	Safety	Exp.Req	Imp.Req	Input Modality
WebCoderBench (Liu et al., 2026)	1,572 reqs; 24 metrics	Static evaluation	✗	✓	✗	✓	✗	2 (Text, Image)
VibeCodeBench (Tran et al., 2026)	100 apps; 964 workflows	Browser agent	✓	✗	✗	✓	✗	1 (Text)
Interaction2Code (Xiao et al., 2025)	127 pgs; 374 inter.	Human evaluation	✓	✓	✗	✓	✗	1 (Image)
FrontendBench (Zhu et al., 2025)	148 tasks	Script assertion	✓	✗	✗	✓	✗	1 (Text)
WebGen-Bench (Lu et al., 2025b)	101 instr.; 647 cases	Checkpoint	✓	✓	✗	✓	✗	1 (Text)
IWR-Bench (Chen et al., 2026)	113 tasks; 1,001 acts; 403 asserts	MLLM assertion	✓	✓	✗	✓	✗	1 (Video)
WebRISE (ours)	**442 tasks; 5,271 reqs; 5,081 states; 5,495 trans.; 12,441 asserts**	**DOM/VLM assertion**	✓	✓	✓	✓	✓	**5** (Text, Markdown, Sketch, Image, Video)

Table 1: Comparison with related web generation benchmarks. Verdict: the mechanism used for pass/fail judgment. Exp.Req / Imp.Req: whether the benchmark includes explicit (user-stated) and implicit (unstated product-level) requirements separately. Input Modality: number of supported modalities with types listed.

namic visual-interactive evaluation (Zhang et al., 2025), real user requirements with interpretable metrics (Liu et al., 2026), interactive webpage reconstruction from video (Chen et al., 2026), and agentic interactive verification (Xu et al., 2025).

This shift changes what should be evaluated. For static pages or local components, visual fidelity, structural similarity, and code executability are natural targets. For interactive web artifacts, however, the key question is whether the page responds correctly to user actions and preserves task-implied state constraints. Accordingly, WebRISE evaluates MLLM-generated web artifacts as executable, stateful interfaces rather than merely rendered pages or code.

Interactive web evaluation. Existing web generation benchmarks increasingly evaluate interaction through scripts, agents, visual judges, or demonstrated trajectories. Script-based protocols such as FrontendBench (Zhu et al., 2025) provide reproducible functional checks but often depend on implementation-specific selectors or entry points. Checkpoint-style protocols such as WebGen-Bench (Lu et al., 2025b) use web-navigation agents to verify expected results, but still focus on local action–result pairs. MLLM-judge and video-based protocols, such as Artifacts-Bench (Zhang et al., 2025) and IWR-Bench (Chen et al., 2026), assess rendered evidence or trajectory reproduction. Beyond generation benchmarks, agent-based web testing systems such as WebProber (Ye et al., 2025) and UXAgent (Lu et al., 2025a) explore websites to identify bugs or usability issues. These protocols make interaction observable, but typically operationalize it through scripts, checkpoints, trajectories, visual evidence, or exploration traces. WebRISE instead formulates interaction evaluation as requirement conformance: an ICG defines requirement-linked states, transitions, and assertions, and an adaptive agent executes them on each generated page, supporting diagnosis beyond pass/fail outcomes.

3 WebRISE: Benchmark Design

Fig. 2 summarizes the benchmark pipeline. WebRISE converts task requirements into executable interaction contracts and evaluates generated HTML through browser-based conformance checks.

3.1 Task Definition

WebRISE evaluates whether an MLLM can generate an executable web artifact that satisfies the interaction behavior of a user-facing task. For each task τ, we define a requirement set R_τ and five modality-specific specifications x_τ^m, where

$$m \in \mathcal{M} = \{\text{Text, Markdown, Sketch, Image, Video}\}. \quad (1)$$

Given x_τ^m, a model f_θ generates a self-contained HTML artifact:

$$h_{\theta,\tau}^m = f_\theta(x_\tau^m). \quad (2)$$

The artifact must be directly executable in a browser and include the required HTML, CSS, and JavaScript without external back-end services or manually prepared runtime state.

For each task, WebRISE derives a requirement-induced interaction contract G_τ from R_τ. The core evaluation asks whether $h_{\theta,\tau}^m$ satisfies G_τ under browser execution, rather than whether it matches a reference DOM, follows a fixed selector path, or reproduces a single visual snapshot. Since G_τ is shared across modalities, WebRISE compares how textual, structural, visual, and temporal specifications affect generation of the same required interactive behavior. Detailed modality construction procedures, prompt templates, and Image/Video specification rules are provided in Appendix A.2.

Ground-truth HTML pages validate contract executability and, when needed, provide Image/Video specifications, but are not treated as unique reference implementations.

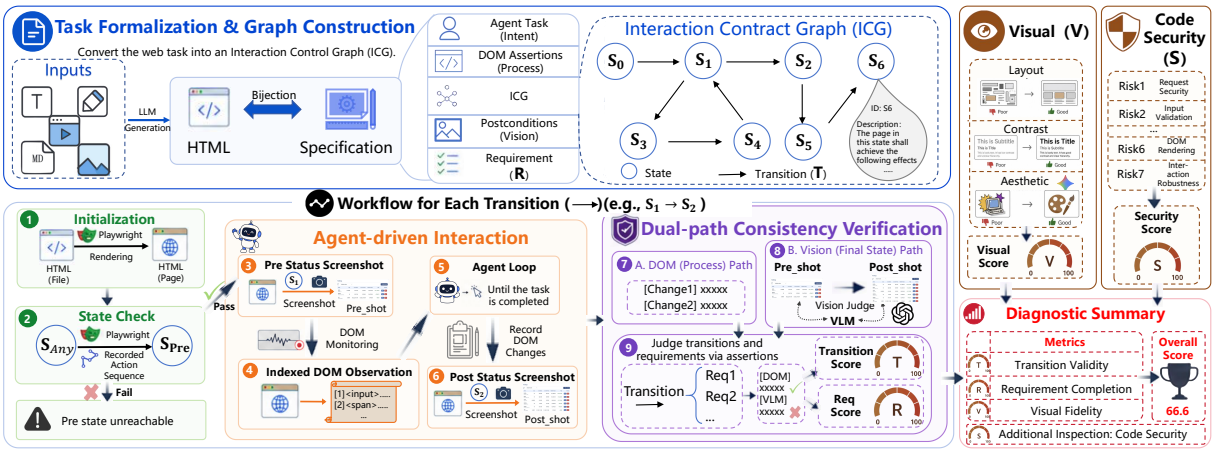

Figure 2: Overview of WebRISE. WebRISE converts multimodal web generation tasks into Interaction Contract Graphs (ICGs), executes each state transition with a contract-guided agent, verifies process and outcome evidence with DOM/visual oracles, and aggregates transition-level verdicts into diagnostic scores.

3.2 Requirement-Induced Interaction Contracts

For each task τ, WebRISE derives an interaction contract from the requirement set R_τ and represents it as an Interaction Contract Graph (ICG):

$$G_\tau = (S_\tau, T_\tau, \Phi_\tau, M_\tau). \tag{3}$$

Here, S_τ denotes stable and replayable UI states, T_τ denotes user-intent-driven transitions, Φ_τ denotes observable DOM/visual predicates, and M_τ maps requirements to test items, transitions, and assertions.

The states in S_τ are requirement-relevant observable UI configurations, rather than full DOM snapshots. Transient effects such as loading indicators, saving states, toasts, debounce effects, and temporary disabled controls are not modeled as standalone states; they are attached to transitions as process-level predicates. This keeps the state space finite and stable while preserving evidence for intermediate interaction behavior.

Each transition in T_τ specifies a user-intent state change, describing the desired outcome rather than a selector-level action sequence. Predicates in Φ_τ verify the transition through DOM evidence for structural or process-level signals and visual evidence for final user-visible outcomes, allowing the same contract to apply across diverse implementations.

The mapping M_τ connects transition-level evidence back to the original requirements. Explicit requirements describe user-stated functional affordances, whereas implicit requirements capture product-level constraints such as state synchronization, boundary feedback, pagination reset, loading feedback, and stale-state removal. Consequently, the contract specifies not only which interactions should be executed, but also how their evidence contributes to requirement-level evaluation.

3.3 Contract Construction Pipeline

WebRISE constructs one interaction contract for each task and applies it to all model outputs across modalities. The pipeline starts from expert-provided task materials and converts them into executable, requirement-attributable interaction contracts through four steps.

Step 1: Expert-informed task collection. We design collection templates specifying the target domain, scenario, and expected web application setting. Anonymous industry practitioners provide domain-grounded task materials, including user-facing requirements, representative interaction goals, and task-relevant data assumptions. These materials serve as raw task sources, rather than executable evaluation specifications.

Step 2: Requirement normalization. We normalize the collected materials into a requirement set R_τ for each task τ. Each set contains explicit requirements for user-stated functional affordances, such as search, filtering, sorting, dragging, and navigation, and implicit requirements for product-level interaction constraints, such as state synchronization, boundary feedback, pagination reset, loading feedback, and stale-state removal.

Step 3: Test Data Contract and test items. From R_τ, WebRISE derives a Test Data Contract specifying the minimal functional readiness for evalua-

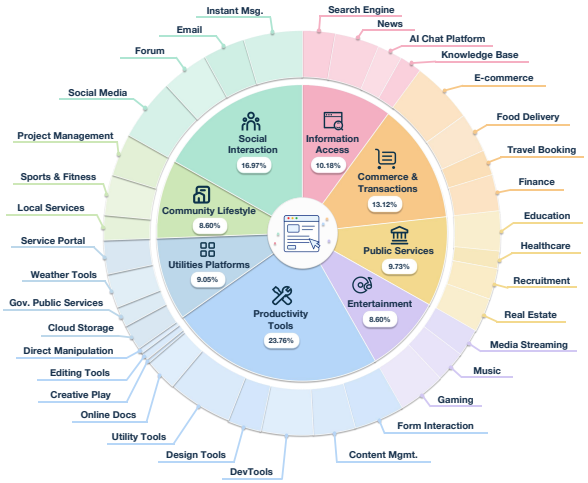

Figure 3: Domain and scenario distribution of WebRISE. Tasks cover 8 domains and 35 scenarios, such as Productivity Tools (23.76%) and Social Interaction (16.97%).

tion, such as initial data, filters, navigation entries, or loadable content, without constraining layout, DOM hierarchy, style, or exact element counts. It derives test items that describe user-triggered behaviors and expected semantic outcomes, rather than CSS selectors, DOM paths, or click sequences.

Step 4: ICG compilation. The Test Data Contract and test items are compiled into the Interaction Contract Graph G_τ. Stable configurations become states, user-triggered behaviors become transitions, and expected outcomes become DOM assertions or visual postconditions. WebRISE also constructs the coverage mapping M_τ, linking requirements to test items, transitions, and assertions.

This pipeline separates domain task authoring from executable evaluation design. Practitioners provide realistic task content, while WebRISE converts it into an interaction contract that defines what should be evaluated; Sec. 4 describes how the contract is executed on generated pages.

3.4 Benchmark Statistics and Quality Control

Fig. 3 shows that WebRISE spans diverse web application settings, with detailed construction statistics reported in Appendix A.1.

After constructing each ICG, we validate it with a ground-truth HTML page generated from the full requirement set. A task is retained only when the ground-truth page, the ICG, and the evaluator form a stable executable loop. We also run schema checks over requirements, test items, states, transitions, assertions, and coverage mappings. Human consistency validation is provided in Appendix A.3.

Algorithm 1 Contract-Guided Evaluation

Require: Page H, transitions $\mathcal{T}$, budget K, settle delay Δ
Load H; initialize replay cache $\Pi \leftarrow \emptyset$
for $t_j = (s_j^{\text{from}}, s_j^{\text{to}}, g_j, P_j, A_j^{\text{dom}}, A_j^{\text{vis}}) \in \mathcal{T}$ **do**
  Restore s_j^{from} by replaying Π
  if restore fails **then**
    $o_j \leftarrow$ SKIPPED; record evidence; **continue**
  end if
  Capture img_{pre}; check P_j
  if any precondition fails **then**
    $o_j \leftarrow$ FAIL; record evidence; **continue**
  end if
  Monitor DOM events; run agent on g_j with budget K
  Wait Δ; capture img_{post}; freeze event log $\mathcal{L}$
  $r_{\text{dom}} \leftarrow \text{SCOREDOM}(A_j^{\text{dom}}, \mathcal{L})$
  $r_{\text{vis}} \leftarrow \text{SCOREVISUAL}(A_j^{\text{vis}}, \text{img}_{\text{pre}}, \text{img}_{\text{post}})$
  $o_j \leftarrow \text{AGGREGATE}(\text{agent status}, r_{\text{dom}}, r_{\text{vis}})$
  if $o_j = \text{PASS}$ **then**
    Update Π with the trajectory reaching s_j^{to}
  end if
  Record evidence $\mathcal{E}_j$
end for

4 Evaluation Protocol

Given a generated HTML artifact H and its Interaction Contract Graph G_τ, WebRISE evaluates contract conformance under browser execution. The ICG specifies *what* to verify, while a contract-guided agent determines *how* to execute each transition on the generated page.

4.1 Protocol Overview

Each transition is represented as

$$t_j = (s_j^{\text{from}}, s_j^{\text{to}}, g_j, P_j, A_j^{\text{dom}}, A_j^{\text{vis}}), \tag{4}$$

where s_j^{from} and s_j^{to} are source and target states, g_j is the natural-language agent goal, P_j is the precondition set, and A_j^{dom}, A_j^{vis} are DOM assertions and visual postconditions. This transition-level formulation supports branching state graphs and localizes evidence to requirement-linked state changes.

Algorithm 1 summarizes the evaluation loop. A transition is marked as PASS only if the source state is reachable, the agent completes the intended interaction, and all required DOM/visual checks hold. The resulting reports are aggregated into the diagnostic metrics in Sec. 4.4.

4.2 Contract-Guided Agent Execution

WebRISE uses an adaptive browser agent rather than a precompiled script. At each step, the page is serialized into an indexed DOM observation containing interaction-relevant controls, state fields, newly appeared elements, scroll context, and editable text selections. Because indices are regenerated after

each action, execution depends on the current page state rather than fixed selectors or reference DOM paths. For branching ICGs, source states are restored by replaying previously verified trajectories, which isolates transitions and separates unreachable states from executable contract violations.

4.3 DOM/Visual Oracle and Evidence

Each transition is verified with a dual-channel oracle. DOM assertions score process-level or element-level evidence from the event log, with `[CHANGE]` checking transient evidence during execution and `[AFTER]` checking the final stable DOM state. Visual postconditions compare pre/-post screenshots to verify final user-visible outcomes such as list updates, sorting changes, moved cards, opened panels, or empty states. For auditability, `WebRISE` records the agent trace, DOM log, screenshots, assertion verdicts, and final transition outcome. Details are provided in Appendix B.

4.4 Diagnostic Metrics

`WebRISE` reports diagnostics as different projections of the same interaction contract. After evaluation, each transition receives one outcome in $\{\textsc{Pass}, \textsc{Fail}, \textsc{Blocked}, \textsc{Skipped}\}$. Only PASS is counted as successful, which avoids giving credit to incomplete interactions or unreachable states.

State and transition metrics. Let S_τ and T_τ denote the state and transition sets in G_τ. Let S_τ^{reach} be the set of reached states, where the initial state is reachable only when its preconditions hold and any other state is reachable only through a passed incoming transition. Let T_τ^{pass} be the set of transitions marked as PASS. We define:

$$S\%(\tau) = \frac{|S_\tau^{\text{reach}}|}{|S_\tau|} \times 100, \tag{5}$$

$$T\%(\tau) = \frac{|T_\tau^{\text{pass}}|}{|T_\tau|} \times 100. \tag{6}$$

Here, $S\%$ measures state reachability, while $T\%$ measures transition-level interaction correctness.

Requirement coverage. Let R_τ^{exp} and R_τ^{imp} denote explicit and implicit requirements, with $R_\tau = R_\tau^{\text{exp}} \cup R_\tau^{\text{imp}}$. Using the coverage mapping M_τ, each requirement r is linked to the transitions and assertions that verify it. We set $\text{sat}(r) = 1$ if all mapped checks for r pass, and 0 otherwise. For any requirement subset $\hat{R} \in \{R_\tau^{\text{exp}}, R_\tau^{\text{imp}}, R_\tau\}$, we define:

$$\mathcal{C}(\hat{R}) = \frac{1}{|\hat{R}|} \sum_{r \in \hat{R}} \text{sat}(r) \times 100. \tag{7}$$

Applying $\mathcal{C}$ to R_τ^{exp}, R_τ^{imp}, and R_τ gives $Re\%$, $Ri\%$, and $R\%$, respectively. $Re\%$ measures user-stated functional affordances, while $Ri\%$ measures implicit state-consistency constraints such as synchronization, boundary feedback, reset behavior, and stale-state removal.

Aggregation. All metrics are computed at the task level and macro-averaged over tasks:

$$\bar{q}(\theta, m) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} q(\theta, \tau, m), \tag{8}$$

where $q \in \{S\%, T\%, Re\%, Ri\%, R\%\}$. This prevents tasks with more transitions or assertions from dominating the aggregate score.

5 Experiments and Findings

5.1 Experimental Setup

We evaluate `WebRISE` on 14 representative models. The model set includes 7 open-weight models and 7 proprietary models. The open-weight models are Qwen3.5-27B (Qwen Team, 2026a), Qwen3.5-122B-A10B, Qwen3.5-397B-A17B, Qwen3.6-27B (Qwen Team, 2026b), Qwen3.6-35B-A3B, Kimi K2.5 (Kimi Team, 2026), and Kimi K2.6 (Moonshot AI, 2026). The proprietary models are GPT-5.4 (OpenAI, 2026a), GPT-5.5 (OpenAI, 2026b), Claude Opus 4.6 (Anthropic, 2026a), Claude Opus 4.7 (Anthropic, 2026b), Gemini 3 Flash (Google DeepMind, 2025), Gemini 3.1 Pro (Google DeepMind, 2026), and Qwen3.6-Plus.

5.2 Overall Model Performance

Table 2 shows that interactive web artifact generation remains far from saturated. Although GPT-5.5 achieves the highest compact Overall score, even its best modality, Video, reaches only $T = 65.6$ and $R = 66.3$, leaving roughly one third of required transitions or requirement checks unsatisfied.

Proprietary models lead, but open-weight models remain competitive. GPT-5.5 and GPT-5.4 obtain the top two Overall scores, 69.1 and 66.8. However, the gap is not determined solely by model access type. Kimi-K2.6 achieves the best open-weight Overall score (63.3), surpassing several proprietary systems and performing especially well under Image and Video. Qwen3.6-27B also reaches a competitive Overall score (62.5), with strong Markdown and Sketch results. These trends suggest that modality handling and stateful interaction reasoning contribute substantially to model ranking.

Visual quality is not a proxy for interaction correctness. High visual scores can coexist with

Model	Text			MD			Sketch			Image			Video			Overall
	T	R	V	T	R	V	T	R	V	T	R	V	T	R	V	
Open-Source																
Qwen3.6-35B-A3B	26.8	30.5	78.2	15.5	19.2	80.8	41.2	45.4	77.0	46.6	49.6	71.7	49.5	52.2	72.8	50.5
Qwen3.5-122B-A10B	38.0	41.2	56.8	42.5	45.9	72.0	38.0	42.3	74.0	40.2	43.8	70.7	42.8	47.1	71.3	51.1
Qwen3.5-27B	36.3	40.0	59.9	41.7	45.5	72.1	38.6	42.7	76.8	42.6	46.7	70.6	43.1	46.9	71.8	51.7
Qwen3.5-397B-A17B	45.7	49.2	64.8	51.1	54.5	75.7	46.8	50.5	78.9	48.4	51.4	72.8	49.3	52.8	72.1	57.6
Kimi-K2.5	**48.5**	**51.9**	68.9	57.0	59.6	73.8	47.8	50.4	79.9	56.9	59.1	72.6	58.6	60.3	72.9	61.2
Qwen3.6-27B	47.9	50.9	75.3	**57.5**	**60.1**	83.0	**50.4**	**53.3**	**87.2**	55.2	57.8	**74.1**	54.2	57.2	**74.1**	62.5
Kimi-K2.6	44.6	47.3	**83.1**	51.7	54.9	**87.1**	47.8	51.5	86.3	**58.5**	**60.4**	73.2	**63.7**	**65.4**	73.5	**63.3**
Proprietary																
Claude Opus 4.6	43.3	45.5	56.6	54.3	56.3	73.9	52.3	55.0	72.2	57.7	59.5	70.2	52.6	54.9	70.7	58.3
Gemini 3 Flash	44.7	48.2	71.9	50.0	54.1	79.3	46.1	49.3	85.4	54.1	57.5	72.4	45.6	48.5	70.8	58.5
Claude Opus 4.7	48.8	50.9	68.3	54.5	56.5	76.2	49.7	52.4	77.4	57.0	58.5	70.5	65.0	66.1	72.7	61.6
Gemini 3.1 Pro	50.7	53.6	69.7	58.9	61.5	79.2	52.2	54.9	84.8	54.5	57.1	72.2	52.0	54.9	71.6	61.9
Qwen3.6-Plus	49.3	51.9	68.2	51.7	54.6	74.5	53.8	56.4	86.3	57.5	59.4	73.8	61.7	63.4	**74.8**	62.5
GPT-5.4	59.7	61.4	78.4	60.5	62.2	79.8	57.8	60.3	**86.6**	60.0	62.1	71.5	63.1	64.8	73.7	66.8
GPT-5.5	**60.3**	**62.3**	**85.6**	**64.4**	**66.1**	**83.3**	**60.6**	**62.9**	86.1	**61.8**	**63.4**	**74.1**	**65.6**	**66.3**	73.9	**69.1**

Table 2: Overall model performance on WebRISE across five input modalities. We report transition validity (T), overall requirement coverage (R), and auxiliary visual quality (V); Overall is a compact average of T, R, and V across modalities. **Bold** and underline denote the best and second-best results within each model group.

Model	Pass (%)	Model	Pass (%)
Gemini-3 Flash	24.9	Qwen3.5-27B	29.8
Kimi K2.5	28.0	Qwen3.5-397B	30.0
Qwen3.6-27B	28.0	Qwen3.5-122B	30.2
Kimi K2.6	28.2	Gemini-3.1 Pro	31.0
Qwen3.6-35B-A3B	28.3	Claude Opus 4.7	31.6
Claude Opus 4.6	28.8	GPT-5.4	34.9
Qwen3.6-Plus	29.7	GPT-5.5	**41.3**

Table 3: Auxiliary safety and robustness diagnostic results by model. Pass rates are computed over applicable check instances; higher is better.

weak executable behavior: Qwen3.6-35B-A3B obtains a strong Markdown visual score ($V = 80.8$), but much lower interaction scores ($T = 15.5$, $R = 19.2$). This mismatch reinforces the need to evaluate generated web artifacts through state transitions and requirement satisfaction, rather than visual plausibility alone.

5.3 Analysis

5.3.1 Safety and Robustness Diagnostics

As an auxiliary diagnostic, we evaluate basic HTML safety and robustness checks. Table 3 shows uniformly low pass rates: even GPT-5.5 reaches only 41.3%, while most models cluster within 25–32%. The flat model ranking and small cross-modality variation suggest that safer HTML generation is not automatically induced by stronger models or richer input specifications.

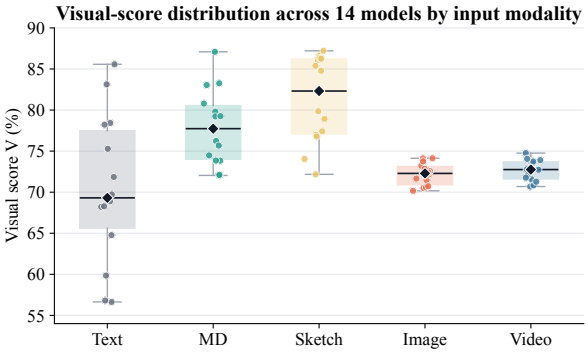

Figure 4: Visual-score distributions across input modalities. Points denote models and boxes show distribution.

5.3.2 Modality Effects

Fig. 4 shows that visual quality and interaction performance follow different patterns. Text has the largest cross-model variance, while Sketch obtains high visual scores due to strong spatial constraints from wireframes. However, Image and Video have similar visual-score distributions, whereas Video leads in interaction-oriented metrics in Appendix C.3. This indicates that Video's advantage is better explained by temporal interaction evidence than by static visual fidelity, reinforcing that visual quality should remain an auxiliary signal. The visual scoring procedure is described in Appendix B.5.

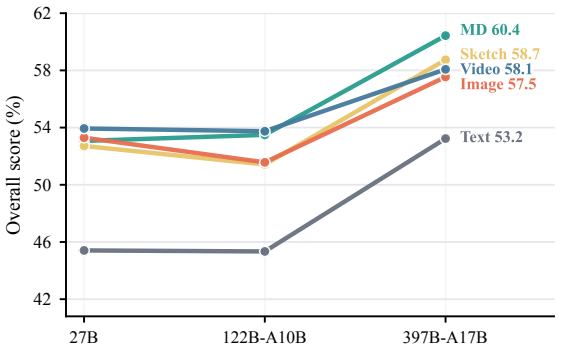

Figure 5: Scaling behavior of the Qwen3.5 family across input modalities. Performance is largely flat from 27B to 122B-A10B, but increases sharply at 397B-A17B.

Evaluator	Signal	Det.	DR(%)
ICG	$T < 100\%$	16/25	64.0
WG-broad	any non-YES	8/25	32.0
WG-strict	$\geq$1 NO	1/25	4.0

Table 4: Defect injection meta-evaluation. We compare ICG-based evaluation with checkpoint-style WebGen (WG) signals on 25 injected state-related defects. Det. denotes detected defects and DR denotes detection rate. ICG detects defects at $2\times$ the rate of WG under the broad criterion and $16\times$ under the strict criterion.

5.3.3 Model Scaling Effects

Fig. 5 shows a non-linear scaling trend within the Qwen3.5 family: performance is largely flat from 27B to 122B-A10B, but improves clearly at 397B-A17B. The gains are strongest under Text and Markdown, where layout, interaction logic, and state behavior must be inferred from weaker specifications. This pattern suggests a scaling knee for stateful web artifact generation, where sufficient model capacity becomes important for jointly modeling layout, interaction logic, and state behavior.

5.3.4 Defect Injection Meta-Evaluation

To assess evaluator sensitivity, we inject state-related defects into GT-validated pages and rerun the same pipeline. Table 4 shows that ICG-based evaluation detects substantially more defects than checkpoint-style WebGen signals, suggesting that explicit state-transition contracts are more sensitive to state corruptions missed by local checkpoints. The remaining missed cases show that defect-sensitive evaluation is not yet exhaustive.

5.3.5 Failure Attribution.

Fig. 6 groups direct failed transitions into four functional error types. GPT-5.5 and Kimi-K2.6 show similar profiles: *State & Logic* dominates, followed by *Feedback & Boundary*. Therefore, many failures occur after required controls or interaction paths are exposed, indicating that the main bottleneck is maintaining correct state updates, result logic, validation behavior, and boundary feedback under user actions.

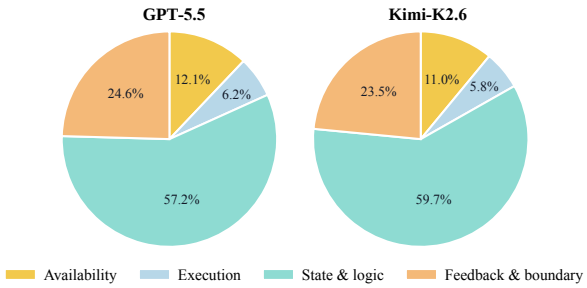

Figure 6: Failure attribution (GPT-5.5 and Kimi-K2.6).

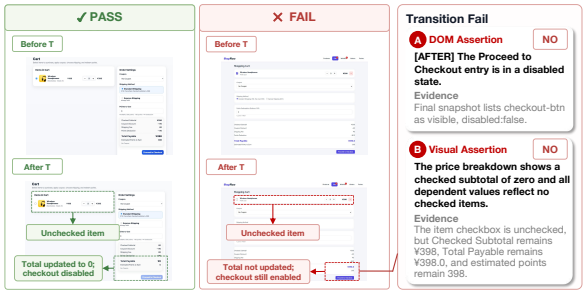

Figure 7: Case study of WebRISE's transition-level diagnosis on a shopping-cart interaction. After the only checked item is unchecked, the passing artifact resets the totals to zero and disables checkout. The failing artifact changes the item checkbox state but leaves the price breakdown and checkout availability stale; WebRISE localizes the error with failed DOM and visual assertions.

5.3.6 Case Study

Fig. 7 illustrates transition-level diagnosis on a shopping-cart interaction. The failing artifact accepts the user click but fails to propagate the resulting state change to dependent totals and checkout availability, exposing a state-consistency error rather than a click-execution failure.

6 Conclusion

We introduced WebRISE, a benchmark that evaluates MLLM-generated web artifacts through requirement-induced observable state-transition conformance. WebRISE represents each task with an Interaction Contract Graph, enabling implementation-agnostic browser execution and state-, transition-, and requirement-level diagnostics over explicit functions and implicit state-consistency constraints. Experiments on 442 tasks, five input modalities, and 14 models show that current systems remain far from solving interactive

web generation: Video provides the strongest interaction signal, while implicit state constraints remain a persistent bottleneck. These results highlight the need to evaluate generated web artifacts by requirement-level state behavior, rather than visual plausibility or isolated action success alone.

Limitations

WebRISE focuses on self-contained HTML artifacts executed in a controlled browser environment. This enables consistent comparison across models and modalities, but does not cover full production web systems involving back-end services, authentication, external APIs, persistent databases, multi-user concurrency, or long-lived sessions. Accordingly, WebRISE should be interpreted as measuring front-end interaction conformance rather than deployment readiness. A natural extension is to augment Interaction Contract Graphs with sandboxed API contracts, persistent data fixtures, and session-level state transitions.

WebRISE evaluates generated pages against requirement-induced interaction contracts. Although the contracts are validated through ground-truth execution, schema checks, human consistency studies, and defect injection, their coverage is still bounded by the specified requirements, generated test items, and DOM/visual assertions. Therefore, WebRISE provides diagnostic evidence of conformance to the defined interaction contract, rather than an exhaustive characterization of all possible user behaviors. Future work can broaden coverage by expanding contract templates, adding richer defect suites, incorporating multiple evaluator agents and selectively auditing uncertain cases.

Ethical Considerations

WebRISE is a diagnostic benchmark, not a deployable system. Contributors and annotators participated under informed consent with aggregated reporting. Because contributors are drawn primarily from a single region, regional product conventions shape what counts as expected interaction, and applications targeting other markets should treat our metrics as a baseline and extend the contract set with locale-specific affordances. LLM-judge scoring is validated against human judgments ($\kappa = 0.74$, Appendix A.3) and defect injection, but remains susceptible to prompt sensitivity and API version drift; reported scores should be read as stable rank-orderings rather than absolute measurements. We release all judge prompts, configurations, and per-assertion verdicts to support independent re-scoring.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (Grant No. 62576191), the Shenzhen Science and Technology Program (ZDCY20250901103533010).

References

Anthropic. 2026a. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6. Accessed: 2026-05-25.

Anthropic. 2026b. Introducing claude opus 4.7. https://www.anthropic.com/news/claude-opus-4-7. Accessed: 2026-05-25.

Yang Chen, Minghao Liu, Yufan Shen, Yunwen Li, Tianyuan Huang, Xinyu Fang, Tianyu Zheng, Wenxuan Huang, Cheng Yang, Licheng Wen, Xuemeng Yang, Daocheng Fu, Jianbiao Mei, Rong Wu, Song Mao, Qunshu Lin, Zhi Yu, Yongliang Shen, Yu Qiao, and Botian Shi. 2026. IWR-Bench: Can LVLMs reconstruct interactive webpage from a user interaction video? In *The Fourteenth International Conference on Learning Representations*.

Google DeepMind. 2025. Gemini 3 flash model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf. Accessed: 2026-05-25.

Google DeepMind. 2026. Gemini 3.1 pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/. Accessed: 2026-05-25.

Vanita Jain, Piyush Agrawal, Subham Banga, Rishabh Kapoor, and Shashwat Gulyani. 2019. Sketch2Code: Transformation of sketches to UI in real-time using deep neural network. *arXiv preprint arXiv:1910.08930*.

Kimi Team. 2026. Kimi K2.5: visual agentic intelligence. *CoRR*, abs/2602.02276.

Chenxu Liu, Yingjie Fu, Wei Yang, Ying Zhang, and Tao Xie. 2026. 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)*, pages 11632–11666. Association for Computational Linguistics.

Yuxuan Lu, Bingsheng Yao, Hansu Gu, Jing Huang, Zheshen Jessie Wang, Yang Li, Jiri Gesi, Qi He, Toby Jia-Jun Li, and Dakuo Wang. 2025a. UXAgent: An

LLM agent-based usability testing framework for web design. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 545:1–545:12. Association for Computing Machinery.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2025b. WebGen-Bench: Evaluating LLMs on generating interactive and functional websites from scratch. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc.

Moonshot AI. 2026. Kimi-k2.6. https://huggingface.co/moonshotai/Kimi-K2.6. Accessed: 2026-05-25.

OpenAI. 2026a. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/. Accessed: 2026-05-25.

OpenAI. 2026b. Introducing gpt-5.5. https://openai.com/index/introducing-gpt-5-5/. Accessed: 2026-05-25.

Ajay Vikram Periasami, Junlin Wang, and Bhuwan Dhingra. 2026. Vision2Code: A multi-domain benchmark for evaluating image-to-code generation. *arXiv preprint arXiv:2605.11307*.

Qwen Team. 2026a. Qwen3.5: Towards native multimodal agents.

Qwen Team. 2026b. Qwen3.6. https://github.com/QwenLM/Qwen3.6. Accessed: 2026-05-25.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2025. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3956–3974. Association for Computational Linguistics.

Hung Tran, Langston Nashold, Rayan Krishnan, Antoine Bigeard, and Alex Gu. 2026. Vibe Code Bench: Evaluating AI models on end-to-end web application development. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pages 514–536. Association for Computing Machinery.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R. Lyu. 2025. Interaction2Code: Benchmarking MLLM-based interactive webpage code generation from interactive prototyping. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 241–253. IEEE.

Mingde Xu, Zhen Yang, Wenyi Hong, Lihang Pan, Xinyue Fan, Yan Wang, Xiaotao Gu, Bin Xu, and Jie Tang. 2025. WebVIA: A web-based vision-language agentic framework for interactive and verifiable UI-to-code generation. *arXiv preprint arXiv:2511.06251*.

Naimeng Ye, Xiao Yu, Ruize Xu, Tianyi Peng, and Zhou Yu. 2025. AI agents for web testing: A case study in the wild. *arXiv preprint arXiv:2509.05197*.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2024. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403.

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. 2025. ArtifactsBench: Bridging the visual-interactive gap in LLM code generation evaluation. *arXiv preprint arXiv:2507.04952*.

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

Appendix

A Additional Benchmark Details

A.1 Benchmark Statistics

Table 5 reports additional construction statistics of WebRISE. The benchmark contains 442 tasks across 8 domains and 35 scenarios, instantiated under five input modalities. At the interaction-contract level, it includes 5,081 states, 5,495 transitions, and 5,271 requirement checks, covering both explicit user-stated requirements and implicit product-level constraints.

Statistic	Value
Domains / Scenarios / Tasks	8 / 35 / 442
Input modalities	5
Task–modality instances	2,210
States	5,081
Transitions	5,495
Requirement checks	5,271
Explicit requirements	2,276
Implicit requirements	2,995
Avg. transitions / task	12.4
Avg. requirement checks / task	11.9

Table 5: Benchmark construction statistics of WebRISE. The table summarizes task coverage, modality instantiation, interaction-contract scale, and requirement-check composition.

A.2 Input Modality Construction

WebRISE instantiates each task under five input modalities to simulate different specification conditions in practical web artifact generation. The task τ and its interaction contract G_τ are fixed across modalities, while the input specification x_τ^m varies. Table 6 summarizes the information provided by each modality and its intended evaluation role.

A.3 Human Consistency Validation

We conduct human consistency validation to examine whether the constructed interaction contracts and automatic evaluators align with human judgements. The validation covers two aspects: (i) the requirement-to-ICG construction and agent-based functional evaluation, and (ii) the modality-specific visual evaluation. This study is used only as a consistency check for benchmark construction and evaluator reliability; it is not used to tune model outputs or change the main evaluation results.

Annotation setup. We sample 300 interaction cases from WebRISE, stratified across domains, input modalities, and task difficulty levels. Each interaction case contains the original task requirement, the corresponding test item or ICG transition, the generated page execution trace, and the automatic verdict. Human annotators judge whether the transition correctly reflects the intended requirement and whether the generated page satisfies the expected functional interaction. For the visual validation, we sample 300 generated HTML pages across the five input modalities. Annotators evaluate visual quality according to the modality-specific criterion: single-page visual quality for Text, reference-page similarity for Image and Video, sketch similarity for Sketch, and Markdown-structure consistency for Markdown.

Annotator disclosure and privacy. The annotators were informed about how the benchmark data were collected and how their annotations would be used in this research. The annotation process does not require releasing private personal information. For privacy reasons, we do not disclose additional identifying information about individual participants, such as names, employers, or detailed personal profiles. All reported results are aggregated.

Metrics. We report accuracy, mean absolute error (MAE), Spearman correlation, Pearson correlation, and Cohen's κ. Accuracy and Cohen's κ measure agreement on binary pass/fail judgements. MAE and correlation metrics are computed over normalized scores when graded judgements are available. For each validation setting, we compare the automatic result against the human-majority judgement, and also report human–human agreement as a reference.

Interaction consistency. As shown in Table 7, the automatic requirement-to-ICG construction achieves 0.86 accuracy and a Cohen's κ of 0.78 against the human majority. The agent-based functional evaluator achieves 0.84 accuracy, 0.86 Spearman correlation, 0.84 Pearson correlation, and a Cohen's κ of 0.74. These scores are close to the human–human agreement, suggesting that both the constructed interaction contracts and the automatic functional evaluation provide stable signals for requirement-level interaction correctness.

Visual consistency. Table 8 reports the consistency of the visual evaluator. The overall visual evaluator obtains 0.81 accuracy, 0.80 Spearman correlation, 0.78 Pearson correlation, and a Cohen's κ of 0.69 against the human majority. The agreement is

Modality	Specification Signal	Evaluation Role
Text	Scenario description, task name, explicit requirements, and Test Data Contract.	Tests whether the model can implement user-stated functions and infer unstated product-level behavior from text-only requirements.
Markdown	Text specification augmented with symbolic layout structure, e.g., header, sidebar, main area, list, card, modal, or toolbar regions.	Tests whether structured layout cues help the model organize the page and ground interaction affordances.
Sketch	Low-fidelity wireframe preserving spatial layout while removing semantic content, copy, color, and visual details.	Tests whether the model can recover UI structure and affordances from coarse visual layout.
Image	Selected high-fidelity screenshot with visually inferable explicit requirements omitted from the accompanying text.	Tests whether visual grounding can recover visible components and complement text-only interaction logic.
Video	Interaction demonstration generated from a validated ICG transition chain.	Tests whether the model can infer interaction flow, process feedback, and temporal state changes from demonstrations.

Table 6: Input modalities in WebRISE. All modalities share the same task and interaction contract, but expose different specification signals to the model.

Comparison	#Samples	Acc.↑	MAE↓	Spearman↑	Pearson↑	κ↑
Human (A) vs. Human (B)	300	0.88	0.18	0.91	0.89	0.81
Human majority vs. requirement-to-ICG construction	300	0.86	0.20	0.88	0.86	0.78
Human majority vs. agent-based functional evaluation	300	0.84	0.23	0.86	0.84	0.74

Table 7: Human consistency validation for interaction-contract construction and functional evaluation. The automatic requirement-to-ICG construction and agent-based evaluation are compared with human-majority judgements, with human–human agreement reported as a reference.

Comparison / Setting	#Samples	Acc.↑	MAE↓	Spearman↑	Pearson↑	κ↑
Human (A) vs. Human (B)	300	0.85	0.17	0.88	0.86	0.77
Human majority vs. Text visual evaluator	60	0.82	0.24	0.83	0.80	0.70
Human majority vs. Image/Video visual evaluator	120	0.81	0.25	0.81	0.79	0.69
Human majority vs. Sketch/Markdown visual evaluator	120	0.80	0.26	0.79	0.77	0.67
Human majority vs. overall visual evaluator	300	0.81	0.25	0.80	0.78	0.69

Table 8: Human consistency validation for modality-specific visual evaluation. The visual evaluator is compared with human-majority judgements under modality-specific criteria, including single-page visual quality for Text, reference-page similarity for Image/Video, sketch similarity for Sketch, and structure consistency for Markdown.

slightly lower than the functional interaction evaluation, which is expected because visual assessment involves more subjective judgement. Nevertheless, the results indicate that the visual evaluator provides a stable auxiliary signal for modality-specific layout quality, visual consistency, and reference alignment.

B Additional Evaluation Protocol Details

B.1 Agent Observation and Action Space

WebRISE provides the evaluation agent with a compact, action-oriented view of the live webpage rather than the full HTML document. At each interaction step, the browser state is converted into an indexed DOM observation that exposes only interaction-relevant elements and state fields. Each actionable element receives an ephemeral `index`, which is local to the current observation and regenerated after the next browser action. This design allows the agent to act on the current page state without relying on persistent CSS selectors, fixed DOM paths, or reference-specific implementation details.

Indexed DOM observation. For each serialized element, WebRISE records its tag, accessibility role, visible text, key attributes, and interaction states. Typical fields include `placeholder`, `value`, `href`, `type`, `checked`, `selected`, `expanded`, `pressed`, `disabled`, `aria-disabled`, and `pointer-events`. For structured or stateful widgets, the observation additionally records option

lists, slider values, scroll offsets, and cursor or selection ranges for editable regions. These fields support fine-grained interactions such as selecting text spans, operating custom dropdowns, restoring scroll context, and performing drag-and-drop transitions.

Non-standard components. Generated pages often implement interactive elements with custom DOM structures rather than native controls. Therefore, WebRISE includes not only native buttons, links, inputs, selects, and text areas, but also elements with interactive ARIA roles, non-negative tabindex, event listeners, pointer or text cursors, contenteditable, or hover-revealed subtrees. Newly appeared elements are marked through cross-step DOM diffing, and hidden or non-interactable elements are explicitly annotated. This makes the agent interface robust to diverse MLLM-generated implementations while avoiding the cost and brittleness of exposing the full DOM.

Action space. The agent action space covers common web operations and interaction-heavy behaviors. It includes pointer actions, keyboard and text actions, form-control actions, spatial actions, and navigation/lifecycle actions: Click, Hover, Type, Clear, PressKey, SelectOption, ToggleCheck, SetSliderValue, Scroll, DragAndDrop, UploadFile, CanvasClickAt, Back, Refresh, WaitFor, and Done. We additionally support SelectText for selecting contiguous text spans inside input, textarea, and contenteditable regions. These actions allow WebRISE to evaluate interactions that cannot be expressed by simple click/type scripts, including anchored text editing, drag-and-drop reordering, file upload, slider control, canvas selection, and browser navigation recovery.

B.2 DOM and Visual Assertion Scoring

WebRISE scores each transition with two complementary assertion channels. DOM assertions operate on structured browser evidence, including the initial DOM snapshot, the final DOM snapshot, and the event log collected during agent execution. Visual postconditions operate on pre- and post-interaction screenshots. This separation lets WebRISE capture transient process evidence and element-level states through DOM signals, while using visual evidence for final user-visible outcomes.

DOM assertion scoring. Each DOM assertion is prefixed with a temporal operator. [CHANGE] requires the condition to hold at some point during the execution timeline, and is used for transient signals such as loading, saving, progress, debounce, confirmation feedback, or temporary disabled states. [AFTER] requires the condition to hold in the final stable DOM state, and is used for persistent outcomes such as selected filters, disabled controls, removed items, restored buttons, or updated ARIA states.

To reduce free-form interpretation, WebRISE applies deterministic priority rules for common state predicates. For non-interactivity, the scorer first checks pointer-events: none, then native disabled or aria-disabled="true", and then state-indicative class tokens such as disabled, inactive, locked, or readonly. For selection or activation, the scorer prioritizes aria-selected, aria-pressed, and aria-checked, followed by class tokens such as selected, active, highlighted, or current. For expansion, it uses aria-expanded and visibility changes in the corresponding container subtree.

Element localization uses visible text, role, aria-label, placeholder, attributes, and child-structure summaries. When multiple candidates match the target and the evidence is insufficient to disambiguate them, the scorer returns UNCERTAIN rather than selecting a target arbitrarily. Only YES is treated as passing when aggregating assertion-, transition-, and requirement-level scores.

Visual postcondition scoring. Visual postconditions compare the screenshots before and after a transition. They are written as behavioral conditions rather than pixel-level constraints, so different implementations can pass if they satisfy the same user-visible semantics. Typical postconditions include list updates, sorting changes, panel expansion, drag-and-drop placement, empty-state display, stale-state removal, and visible value updates.

The visual scorer uses before/after differences to judge the requested semantic change. For conditional assertions, it first determines which branch applies from the screenshots and evaluates only that branch. If relevant content is clipped by the viewport or a scrollable container, the scorer relies only on fully visible evidence. For search or filter assertions, an empty result may pass when the filter is visibly active and the page shows a valid empty state. Ambiguous or unsupported visual evidence is marked as UNCERTAIN, and does not count as a passing assertion.

B.3 Transition Outcomes and Evidence

Each evaluated transition receives one of four outcomes. PASS indicates that the source state is reachable, the agent completes the intended interaction, and all required DOM/visual checks pass. FAIL indicates that the transition is executable but at least one required assertion or postcondition is violated. BLOCKED indicates that the agent cannot complete the interaction within the budget, typically because the required affordance is absent, hidden, or non-functional. SKIPPED indicates that the source state cannot be restored, usually because a prerequisite transition failed or the replay path is unavailable. This taxonomy separates contract violations from execution failures and prevents a single upstream defect from being counted repeatedly across downstream transitions.

For auditability, WebRISE stores a structured evidence bundle for every transition. The bundle includes the transition identifier, source and target state descriptors, the natural-language agent goal, pre- and post-interaction screenshots, the agent action trace, the DOM event log, initial and final DOM snapshots, per-assertion verdicts, the final transition outcome, and the replay path when state replay is used. Each per-assertion record stores the verdict, supporting evidence fragments, and scorer version. The evidence bundle allows each reported error to be traced to the relevant phase, such as source-state restoration, agent execution, DOM assertion scoring, visual postcondition scoring, or replay. It also supports manual auditing, phase-level error analysis, and defect-injection meta-evaluation.

B.4 Additional Metric Details

In addition to the main metrics, WebRISE records test-item-level and assertion-level signals for diagnostic analysis. These signals are not used as primary leaderboard metrics, but help localize errors between user-facing behaviors, transition checks, and individual evidence channels.

Test-item coverage. A test item corresponds to a user-triggered behavior and its expected semantic outcome. Using the coverage mapping M_τ, each test item is linked to the transitions and assertions that verify it. We mark a test item as satisfied only when all mapped transitions and required assertions pass:

$$TI\%(\tau) = \frac{1}{|I_\tau|} \sum_{i \in I_\tau} \mathrm{sat}(i) \times 100, \tag{9}$$

where I_τ is the set of test items for task τ and $\mathrm{sat}(i) \in \{0, 1\}$. Because test items are closer to user-facing behaviors than raw transitions, $TI\%$ is mainly used for qualitative error analysis.

Assertion-level verdicts. Each DOM assertion and visual postcondition receives a verdict in $\{\text{YES}, \text{NO}, \text{UNCERTAIN}\}$. Only YES is treated as passing when aggregating assertion-, transition-, test-item-, and requirement-level scores. NO indicates contradicted evidence, while UNCERTAIN indicates insufficient or ambiguous evidence. This conservative rule prevents ambiguous observations from inflating final scores.

Aggregation convention. Unless otherwise specified, model- and modality-level scores are computed by macro-averaging task-level scores. This gives each task equal weight and prevents tasks with more transitions, assertions, or requirements from dominating aggregate results. Assertion-level and test-item-level metrics are used for debugging, case studies, and failure attribution, while the main paper focuses on state reachability, transition validity, and explicit/implicit requirement coverage.

Compact overall score. For leaderboard readability, we report an auxiliary Overall score:

$$O(\theta) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \frac{T(\theta, m) + R(\theta, m) + V(\theta, m)}{3}, \tag{10}$$

where V is the modality-specific auxiliary visual score. Overall is used only as a compact summary; the primary analysis relies on the diagnostic interaction and requirement metrics, especially $T\%$, $R_e\%$, $R_i\%$, and $R\%$.

B.5 Visual Quality Evaluation Details

WebRISE reports visual quality as an auxiliary signal, complementary to executable interaction metrics. The visual evaluator combines three components: layout structure, color accessibility, and perceptual aesthetics, with modality-specific aggregation.

Layout and color. The layout module performs coarse-grained block modeling over the rendered page, measuring alignment, structural clarity, and floating-element artifacts. When a visual reference is available, it also measures cross-page structural consistency using row-level signatures and grid-distribution similarity. The color module checks text contrast against WCAG thresholds ($\geq$4.5:1 for normal text and $\geq$3:1 for large text), and for Image/Video additionally compares palette and contrast-profile similarity to the reference page.

Model	Avg.	Text	MD	Sketch	Image	Video
Kimi-K2.6	80.6	83.1	87.1	86.3	73.2	73.5
GPT-5.5	80.6	85.6	83.3	86.1	74.1	73.9
Qwen3.6-27B	78.7	75.3	83.0	87.2	74.1	74.1
GPT-5.4	78.0	78.4	79.8	86.6	71.5	73.7
Qwen3.6-35B-A3B	76.1	78.2	80.8	77.0	71.7	72.8
Gemini 3 Flash	76.0	71.9	79.3	85.4	72.4	70.8
Qwen3.6-Plus	75.5	68.2	74.5	86.3	73.8	74.8
Gemini 3.1 Pro	75.5	69.7	79.2	84.8	72.2	71.6
Kimi-K2.5	73.6	68.9	73.8	79.9	72.6	72.9
Claude Opus 4.7	73.0	68.3	76.2	77.4	70.5	72.7
Qwen3.5-397B-A17B	72.9	64.8	75.7	78.9	72.8	72.1
Qwen3.5-27B	70.2	59.9	72.1	76.8	70.6	71.8
Qwen3.5-122B-A10B	69.0	56.8	72.0	74.0	70.7	71.3
Claude Opus 4.6	68.7	56.6	73.9	72.2	70.2	70.7

Table 9: Auxiliary visual-quality scores by model and input modality. Scores are reported on a 0–100 scale and macro-averaged across tasks.

Judge model	#Pairs	Detected
GPT-5.4	100	97 / 100
GPT-5-mini	100	95 / 100

Table 10: Judge-model robustness validation on 100 sampled GT/defect-injected HTML pairs.

Aesthetics. A VLM-based scorer evaluates screenshots along high-level perceptual dimensions, including whitespace balance, recurring-element consistency, hierarchy clarity, and overall polish. This complements rule-based layout and color checks with visual judgments that are difficult to encode deterministically.

Modality-specific aggregation. For Text, aesthetics is the primary signal, with layout and color used as auxiliary checks. For Markdown and Sketch, structural similarity to the reference specification receives the largest weight, supplemented by aesthetics. For Image and Video, layout fidelity and color reproduction relative to the reference page are primary, with aesthetics as a secondary signal. All visual scores are macro-averaged across tasks, and full model-by-modality visual scores are reported in Table 9.

C Additional Experimental Details and Results

C.1 Model Deployment Details

Among the locally hosted open-weight models, Kimi K2.5 and Kimi K2.6 are served on Huawei Ascend NPUs, whereas the Qwen-series models are served on NVIDIA GPUs. Both platforms provide comparable inference performance during evaluation.

Mod.	T	Req. Coverage			R	V	Overall
		R_e	R_i	$\Delta\downarrow$			
Text	46.0	56.4	43.0	13.5	48.9	70.4	55.1
MD	50.8	61.1	47.6	13.5	53.7	<u>77.9</u>	<u>60.8</u>
Sketch	48.8	60.1	45.4	14.7	52.0	**81.4**	60.7
Image	<u>53.6</u>	**62.8**	<u>50.8</u>	<u>12.0</u>	<u>56.2</u>	72.2	60.7
Video	**54.8**	<u>61.3</u>	**53.6**	**7.7**	**57.2**	72.6	**61.5**

Table 11: Average modality-level performance on WebRISE across all evaluated models and tasks. T denotes transition validity; R_e and R_i denote explicit and implicit requirement coverage; $\Delta = R_e - R_i$ is the explicit–implicit gap; R denotes overall requirement coverage; V is the auxiliary visual score; and Overall is the mean of T, R, and V. Bold and underlined values indicate the best and second-best results in each column.

C.2 Evaluation Judge Configuration

We use GPT-5-mini for transition-level DOM assertion and visual postcondition scoring, and Gemini-3-Flash-Preview for auxiliary visual-quality scoring. The same judge configuration is applied to all evaluated models, tasks, and modalities.

To verify that the lighter transition-level judge does not reduce defect sensitivity, we compare GPT-5-mini with GPT-5.4 on 100 sampled GT/defect-injected HTML pairs. Each pair contains a GT-validated page that passes the ICG-based evaluation and a corresponding defect-injected variant that introduces a controlled interaction fault. As shown in Table 10, GPT-5-mini remains close to GPT-5.4 on this sampled control set.

C.3 Additional Modality Analysis

Table 11 reports modality-level averages across all evaluated models and tasks. Video achieves the strongest interaction-oriented performance, leading in transition validity (T), implicit requirement coverage (R_i), and overall requirement coverage (R), while reducing the explicit–implicit gap to 7.7 points. This suggests that temporal demonstrations are especially helpful for recovering state changes and implicit product-level behavior. Image obtains the highest explicit requirement coverage ($R_e = 62.8$) and closely follows Video on T and R, indicating that high-fidelity visual grounding helps models recover visible components and initial interface state. By contrast, Sketch obtains the highest auxiliary visual score ($V = 81.4$), but lags behind Image and Video on interaction and requirement metrics. This indicates that visual organization alone is not a reliable proxy for executable

interaction correctness.

C.4 Difficulty and Failure Attribution

Failure-type taxonomy. To analyze where functional failures occur along the interaction implementation chain, we group direct failed transitions into four functional error types. **Availability** captures whether the page provides the required entry point, control, or interaction flow for completing the task. **Execution** captures whether a user action takes effect when the relevant control or input area is present. **State & Logic** captures whether the page correctly updates state, data rules, target content, visual status, and context after an action. **Feedback & Boundary** captures whether the page correctly handles validation, disabled states, loading, errors, confirmations, and empty states.

To understand whether low scores arise from uniformly harder tasks or from qualitatively different failure modes, we analyze the R-based Hard50 and Easy50 splits from both performance and failure-attribution perspectives.

Table 13 compares the R-based Hard50 and Easy50 splits by input modality. The performance gap is large across all modalities, confirming that Hard50 captures genuinely difficult interaction tasks rather than small metric fluctuations. Video remains the strongest modality on both splits, but its margin is much larger on Hard50: compared with Image, Video improves T, R_i, and R by 4.9, 6.1, and 4.8 points on Hard50, but only by 1.6, 1.7, and 1.4 points on Easy50. This suggests that dynamic interaction evidence is especially useful when tasks require non-trivial state transitions and implicit behavior recovery.

Fig. 8 further shows that the two splits expose different failure profiles. State and logic errors dominate both Hard50 and Easy50, indicating that stateful result logic remains the central bottleneck. However, Hard50 contains higher shares of availability failures and feedback/boundary failures, suggesting that difficult tasks often fail before or around the interaction boundary: required affordances may be missing, states may be unreachable, or edge-state feedback may be incomplete. By contrast, Easy50 failures are more concentrated in state and logic errors, meaning that models often expose a basic interaction path but still fail to maintain the correct result logic or state consistency.

C.5 Full Model × Modality Results

Table 12 reports the full model-by-modality results.

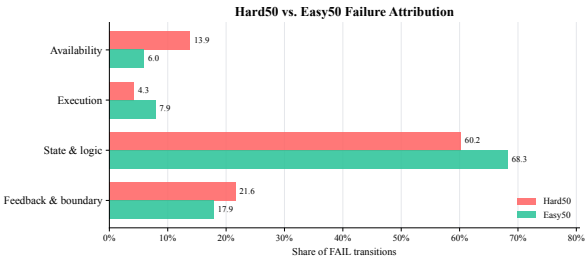

Figure 8: Failure-family attribution on the R-based Hard50 and Easy50 splits. State and logic errors dominate both splits, while Hard50 shows larger shares of availability and feedback/boundary failures.

C.6 Defect Injection Details

We further inspect the 13 defect-injected cases where WebGen marks all test items as YES, but ICG still detects the injected defect. As shown in Table 14, these cases are not dominated by visibly missing controls or rendering failures. Instead, they involve longer-range behavioral constraints, including accumulated history preservation, cross-feature non-interference, navigation-time state retention, action gating, and pre/post state consistency. This explains why checkpoint-style evaluation can miss them: it often verifies whether the local target appears completed, whereas ICG follows transition chains and checks requirement-linked postconditions and state invariants. These ICG-only cases therefore show that explicit state-transition contracts provide complementary coverage for hidden state errors and cross-feature side effects beyond local checkpoint judgments.

C.7 Safety Evaluation Details

We provide rule-level safety diagnostics for GPT-5.5, the strongest model in the main interaction evaluation. These diagnostics are auxiliary to `WebRISE`'s interaction metrics and are intended to reveal common engineering-level weaknesses in generated HTML artifacts.

As shown in Table 16, the weakest rule families are asynchronous interaction robustness, DOM rendering safety, and request security. The low pass rates for R7, R6, and R1 indicate that generated pages often miss repeated-trigger guards, safe DOM rendering practices, and basic protections for sensitive requests. In contrast, navigation security obtains a high pass rate, but covers far fewer applicable checks and should not be interpreted as broad safety reliability.

Table 15 further shows that the most frequent low-pass checks involve missing input constraints,

Model	Text						MD						Sketch						Image						Video						Overall
	S	T	R_e	R_i	R	V	S	T	R_e	R_i	R	V	S	T	R_e	R_i	R	V	S	T	R_e	R_i	R	V	S	T	R_e	R_i	R	V	
Open-Source																															
Qwen3.6-35B-A3B	31.8	26.8	36.6	25.9	30.5	78.2	20.0	15.5	22.3	16.7	19.2	80.8	47.1	41.2	54.5	38.1	45.4	77.0	51.8	46.6	56.7	43.8	49.6	71.7	53.3	49.5	57.7	47.9	52.2	72.8	50.5
Qwen3.5-122B-A10B	42.8	38.0	48.9	35.2	41.2	56.8	47.5	42.5	54.1	39.3	45.9	72.0	43.4	38.0	49.7	36.2	42.3	74.0	45.4	40.2	50.9	38.1	43.8	70.7	47.0	42.8	51.7	43.5	47.1	71.3	51.1
Qwen3.5-27B	41.4	36.3	47.3	34.3	40.0	59.9	46.9	41.7	53.5	38.8	45.5	72.1	44.6	38.6	50.7	36.5	42.7	76.8	47.7	42.6	53.6	41.2	46.7	70.6	47.2	43.1	51.0	43.4	46.9	71.8	51.7
Qwen3.5-397B-A17B	51.2	45.7	57.2	42.8	49.2	64.8	56.2	51.1	62.3	48.2	54.5	75.7	52.5	46.8	60.1	42.8	50.5	78.9	53.2	48.4	57.7	46.3	51.4	72.8	53.3	49.3	56.8	49.4	52.8	72.1	57.6
Qwen3.6-27B	52.7	47.9	58.4	44.8	50.9	75.3	**62.2**	**57.5**	**67.3**	**54.3**	**60.1**	83.0	**55.6**	**50.4**	**60.9**	**47.2**	**53.3**	**87.2**	60.3	55.2	64.8	52.0	57.8	**74.1**	58.5	54.2	61.4	53.4	57.2	**74.1**	62.5
Kimi-K2.5	**53.5**	**48.5**	**59.4**	**46.1**	**51.9**	68.9	61.9	57.0	**67.3**	53.5	59.6	73.8	52.8	47.8	58.3	44.0	50.4	79.9	61.3	56.9	65.2	54.0	59.1	72.6	62.2	58.6	65.0	56.5	60.3	72.9	61.2
Kimi-K2.6	49.4	44.6	54.2	41.8	47.3	**83.1**	56.5	51.7	62.9	48.4	54.9	**87.1**	53.0	47.8	58.8	45.6	51.5	86.3	**63.2**	**58.5**	**66.6**	**55.4**	**60.4**	73.2	**67.1**	**63.7**	**68.4**	**62.6**	**65.4**	73.5	**63.3**
Proprietary																															
Gemini 3 Flash	49.7	44.7	56.2	41.5	48.2	71.9	55.3	50.0	63.2	47.0	54.1	79.3	51.2	46.1	57.7	42.7	49.3	85.4	59.5	54.1	64.7	51.5	57.5	72.4	49.9	45.6	53.5	44.2	48.5	70.8	58.5
Claude Opus 4.6	47.9	43.3	53.1	39.5	45.5	56.6	58.8	54.3	63.3	50.6	56.3	73.9	57.5	52.3	63.6	48.0	55.0	72.2	62.1	57.7	65.9	54.2	59.5	70.2	55.7	52.6	58.4	51.7	54.9	70.7	58.3
Gemini 3.1 Pro	55.6	50.7	61.1	47.5	53.6	69.7	63.6	58.9	69.5	54.9	61.5	79.2	56.8	52.2	62.5	48.8	54.9	84.8	59.1	54.5	63.3	51.9	57.1	72.2	55.8	52.0	58.9	51.5	54.9	71.6	61.9
Qwen3.6-Plus	54.2	49.3	58.6	46.6	51.9	68.2	56.7	51.7	62.6	48.0	54.6	74.5	58.8	53.8	63.8	50.6	56.4	86.3	61.7	57.5	66.0	54.0	59.4	73.8	65.1	61.7	68.3	58.9	63.4	**74.8**	62.5
Claude Opus 4.7	53.4	48.8	57.6	45.8	50.9	68.3	58.6	54.5	63.1	51.2	56.5	76.2	54.3	49.7	59.3	46.9	52.4	77.4	61.3	57.0	64.5	53.9	58.5	70.5	67.9	65.0	**70.0**	62.8	66.1	72.7	61.6
GPT-5.4	64.6	59.7	70.3	54.3	61.4	78.4	65.2	60.5	70.5	55.4	62.2	79.8	62.7	57.8	70.2	52.4	60.3	**86.6**	64.5	60.0	68.7	56.6	62.1	71.5	66.1	63.1	68.4	61.6	64.8	73.7	66.8
GPT-5.5	**65.1**	**60.3**	**71.1**	**55.3**	**62.3**	**85.6**	**69.1**	**64.4**	**73.6**	**59.8**	**66.1**	**83.3**	**65.3**	**60.6**	**71.6**	**56.0**	**62.9**	86.1	**66.4**	**61.8**	**69.8**	**58.0**	**63.4**	**74.1**	**68.4**	**65.6**	69.4	**63.5**	**66.3**	73.9	**69.1**

Table 12: Full model × modality results with state reachability (S), transition validity (T), explicit (R_e) and implicit (R_i) requirement coverage breakdown, and modality-specific visual scores.

Mod.	Hard50			Easy50		
	T	R_i	R	T	R_i	R
Text	24.3	22.1	27.0	68.7	64.6	71.4
MD	25.5	23.7	28.4	73.7	70.4	77.1
Sketch	22.4	21.0	25.7	73.0	69.0	75.9
Image	31.6	31.1	34.7	74.5	71.2	77.3
Video	**36.5**	**37.2**	**39.5**	**76.1**	**72.9**	**78.7**

Table 13: Performance on the R-based Hard50 and Easy50 splits by input modality. Hard50 and Easy50 are selected as the 50 tasks with the lowest and highest model-averaged overall requirement coverage (R), respectively. Video leads on both splits, with a larger advantage on Hard50, especially for implicit requirement coverage (R_i).

unsafe DOM rendering, repeated-click guards, and sensitive-form protections. These results suggest that even strong MLLMs may generate functional and visually plausible webpages while omitting basic front-end safety and robustness safeguards.

C.8 Case Study

This section presents representative qualitative cases for the failure types used in our failure attribution analysis. Each case shows the input modality, a passing artifact, a failing artifact, the executed transition, and the failed evidence. Together, these examples show how `WebRISE` evaluates each transition from the source state to the target state and records where the expected behavior breaks.

Case 1: Execution failure. As shown in Fig. 9, this case tests whether a generated messaging interface can execute a batch operation after filtering and selecting visible conversations. The expected behavior is that the selected conversations disappear after the batch-delete action, while unmatched conversations remain in the restored full list. Although the failing artifact displays the search and selection flow, the selected conversations are still visible after deletion. This indicates an execution failure: the page exposes a plausible operation path, but the underlying delete action is not successfully applied to the selected items.

Case 2: Feedback & Boundary failure. As shown in Fig. 10, this case focuses on process feedback during an infinite-scroll interaction. After the user scrolls to the bottom, the page should indicate that the next page of content is being fetched, for example through a skeleton screen or loading placeholder. The failing artifact reaches the scroll boundary but provides no observable loading state, and the evidence also shows no newly appended posts. This failure shows that the main interaction entry point may exist, while the boundary-state feedback required for a realistic web interaction is still missing.

Case 3: State & Logic failure – inconsistent state update. As shown in Fig. 11, this case evaluates whether a course registration page correctly updates dependent waitlist state. The transition first enrolls Alice and Bob, adds Carol and Dave to the CS201 waitlist, cancels Carol's entry, and then opens the waitlist view. The failing artifact correctly removes Carol, but Dave remains marked as #2 instead of being promoted to #1. The error is an incomplete state update: one part of the state changes, while the dependent queue order is left stale.

Case 4: State & Logic failure – cross-view inconsistency. As shown in Fig. 12, this case tests synchronization between a layer list and the visible canvas. After the topmost layer is hidden from

Error Pattern	N	Why WebGen Misses It	Example
Record loss under repeated operations	4	WebGen verifies the latest successful action, but does not check whether earlier records remain visible.	Multiple prize-wheel spins appear successful, but only the latest prize is kept in the winning records.
Cross-feature side effects	5	WebGen evaluates the target feature locally, without checking whether unrelated UI state or components are changed.	Editing one saved link succeeds, but other saved links have their URLs silently corrupted.
Draft state loss across navigation	2	WebGen checks that pages and forms are reachable, but not whether intermediate user inputs persist across navigation.	A leave type is selected, then resets after the user visits another module and returns.
Missing explicit-action gating	1	WebGen checks that an output can be produced, but not whether it waits for the intended user trigger.	A calculator result updates immediately after editing the input, before the user clicks execute.
Pre/post state inconsistency	1	WebGen accepts confirmation feedback or a results page, without exact before–after value comparison.	A poll vote is confirmed, but the selected option's count does not increase.

Table 14: ICG-only error patterns among cases where WebGen marks all test items as YES. Counts are computed over 13 defect-injected cases detected only by ICG.

Check ID	Interpretation	Checks	Pass(%)
R2_text_input_constraints	Missing required/maxlength/pattern/min/max on inputs	2,956	2.9
R6_dangerous_dom_uncertain	Suspected unsafe DOM rendering patterns	430	0.0
R6_dangerous_dom_rendering	Dangerous DOM API usage (potential XSS)	437	2.5
R7_continuous_trigger_guard	No debounce/disable guard on repeated clicks	273	1.8
R1_sensitive_form_csrf	Sensitive form lacks CSRF token	191	0.0
R1_sensitive_form_post	Sensitive form not using POST method	191	1.0
R7_async_error_recovery	Missing error recovery path for async failures	277	11.6
R5_filter_sort_sync	Filter/sort state out of sync with UI or results	1,284	26.1
R1_button_explicit_type	Button missing explicit type attribute	1,075	35.5
R7_async_loading_state	Missing loading indicator during async operations	277	52.0
R2_search_trim	Unstable whitespace handling in search input	672	64.1
R2_invalid_input_handling	Invalid input handling	1,058	91.8

Table 15: High-frequency safety check details for GPT-5.5, sorted by pass rate. The lowest-pass checks mainly involve input constraints, unsafe DOM rendering, repeated-trigger guards, and sensitive-form protections.

Rule	Meaning	Checks	Pass(%)
R7	Async & interaction robustness	881	20.5
R6	DOM rendering safety	1,169	25.5
R1	Request security	1,457	26.4
R2	Input validation	5,253	38.8
R5	State consistency	2,843	63.8
R3	Upload security	187	66.3
R4	Navigation security	45	97.8

Table 16: Safety rule-level breakdown for GPT-5.5. The weakest rule families are asynchronous interaction robustness, DOM rendering safety, and request security.

the layer panel, the corresponding object should no longer appear on the canvas, and the layer list should reflect the hidden state. The failing artifact provides only uncertain final-state evidence in the layer list and still displays the hidden layer on the canvas. This exposes a cross-view state inconsistency: the control-side state and the rendered canvas state are not synchronized.

Case 5: State & Logic failure – state not preserved. As shown in Fig. 13, this case examines whether a social post editor preserves draft content across an unexpected refresh. The transition requires opening the editor, entering the text "draft recovery test", attaching an image, refreshing the page, and reopening the editor. The failing artifact reopens the editor but shows the placeholder text and no image preview, meaning that neither the text nor the attachment is restored. This demonstrates a persistence failure: the interaction is locally available, but the generated page does not preserve user-created state across the page lifecycle.

Case 6: State & Logic failure – operation state reset. As shown in Fig. 14, this case evaluates whether an image editor preserves earlier editing state when a later operation is applied. The transition first rotates the image by 90 degrees and then performs a crop with a selected aspect ratio. The failing artifact applies the crop, but the final result

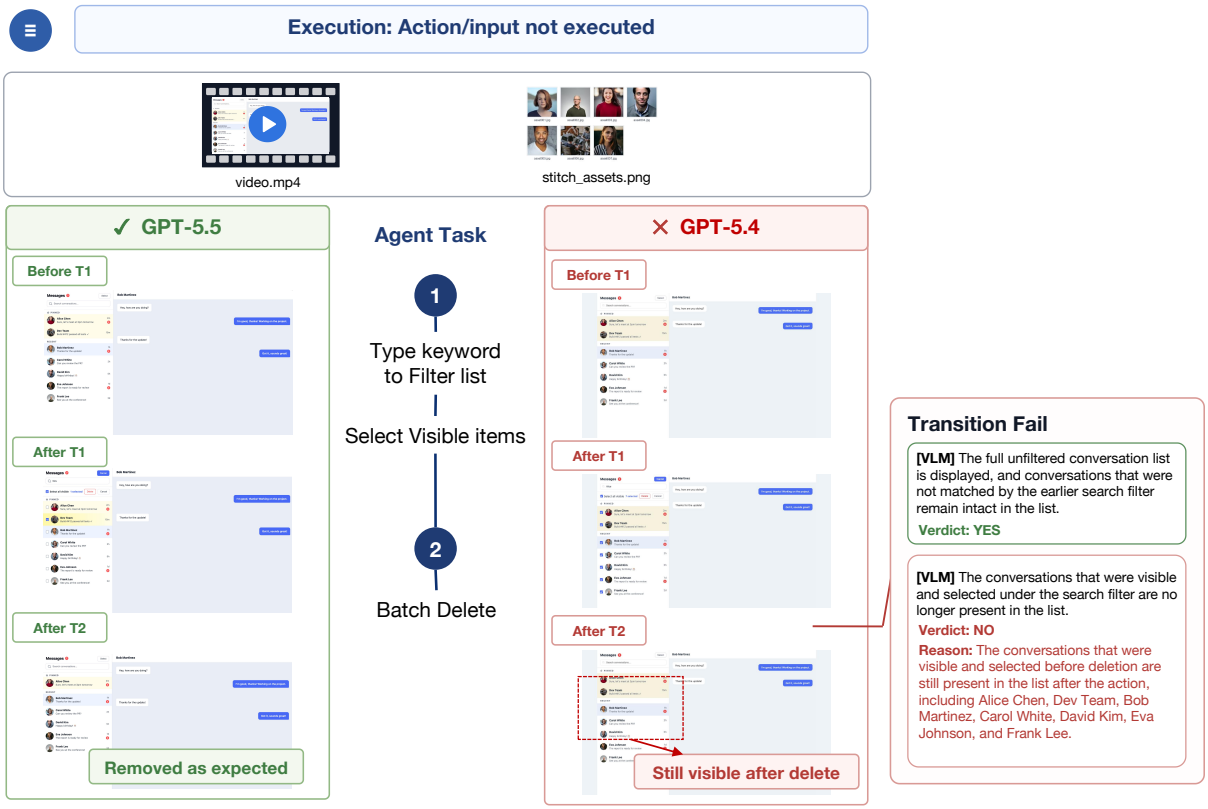

Figure 9: Execution failure in a messaging interface. The transition requires filtering the conversation list, selecting the visible conversations, and batch deleting them. The failing artifact keeps the selected conversations visible after deletion.

no longer preserves the prior 90-degree rotation state. This is a state-preservation error across sequential editing operations: the later crop operation incorrectly resets an earlier transformation state.

D Prompt Templates

This section lists the prompt templates used in WebRISE, including templates for test data contract generation, test item generation, Interaction Contract Graph construction, contract-guided agent execution, DOM assertion scoring, visual postcondition scoring.

E Code and Data Availability

Upon acceptance, we will release the code and data for WebRISE under the MIT license. The release will include task specifications, requirement annotations, Interaction Contract Graphs, evaluation scripts, prompt templates, and aggregated results for reproducing the main experiments. We will exclude information that may identify individual contributors or annotators for privacy reasons.

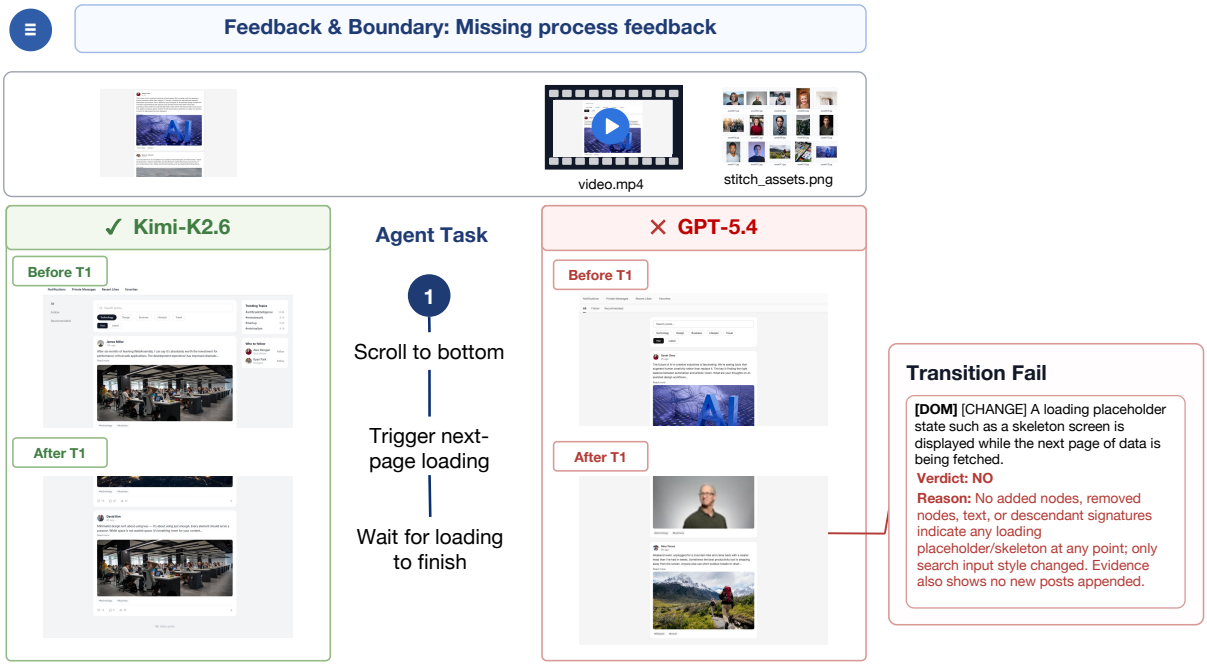

Figure 10: Feedback and boundary failure in a feed-loading interaction. The transition requires scrolling to the bottom, triggering next-page loading, and displaying loading feedback during data fetching. The failing artifact does not show the required loading placeholder.

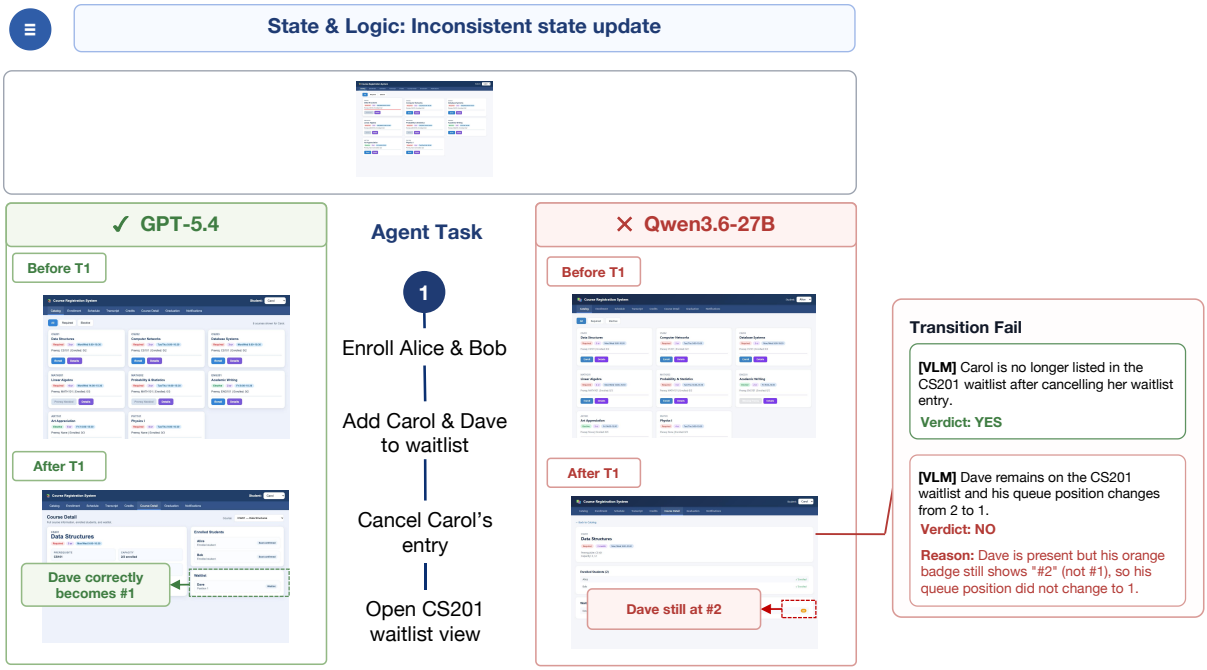

Figure 11: State-and-logic failure in a course waitlist interaction. After Carol's waitlist entry is cancelled, Dave should remain on the CS201 waitlist and move from position #2 to #1. The failing artifact removes Carol but leaves Dave's queue position unchanged.

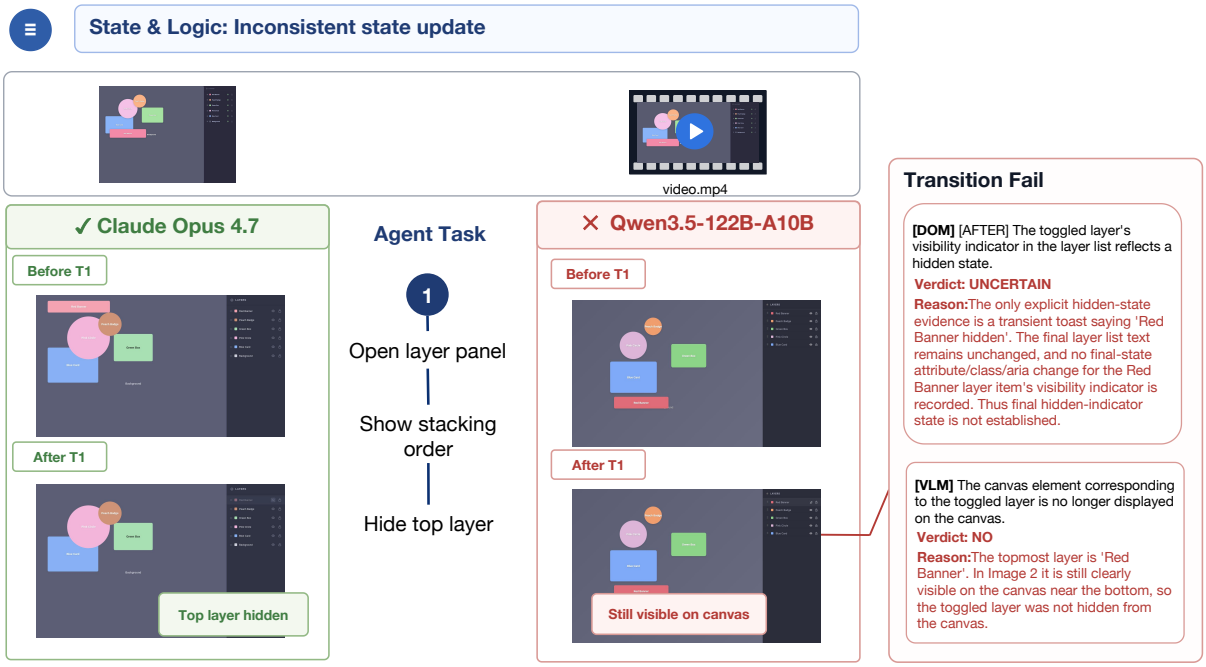

Figure 12: State-and-logic failure in a layer-list interaction. The transition requires opening the layer panel and hiding the topmost layer. The failing artifact gives weak or transient hidden-state evidence but leaves the corresponding canvas element visible.

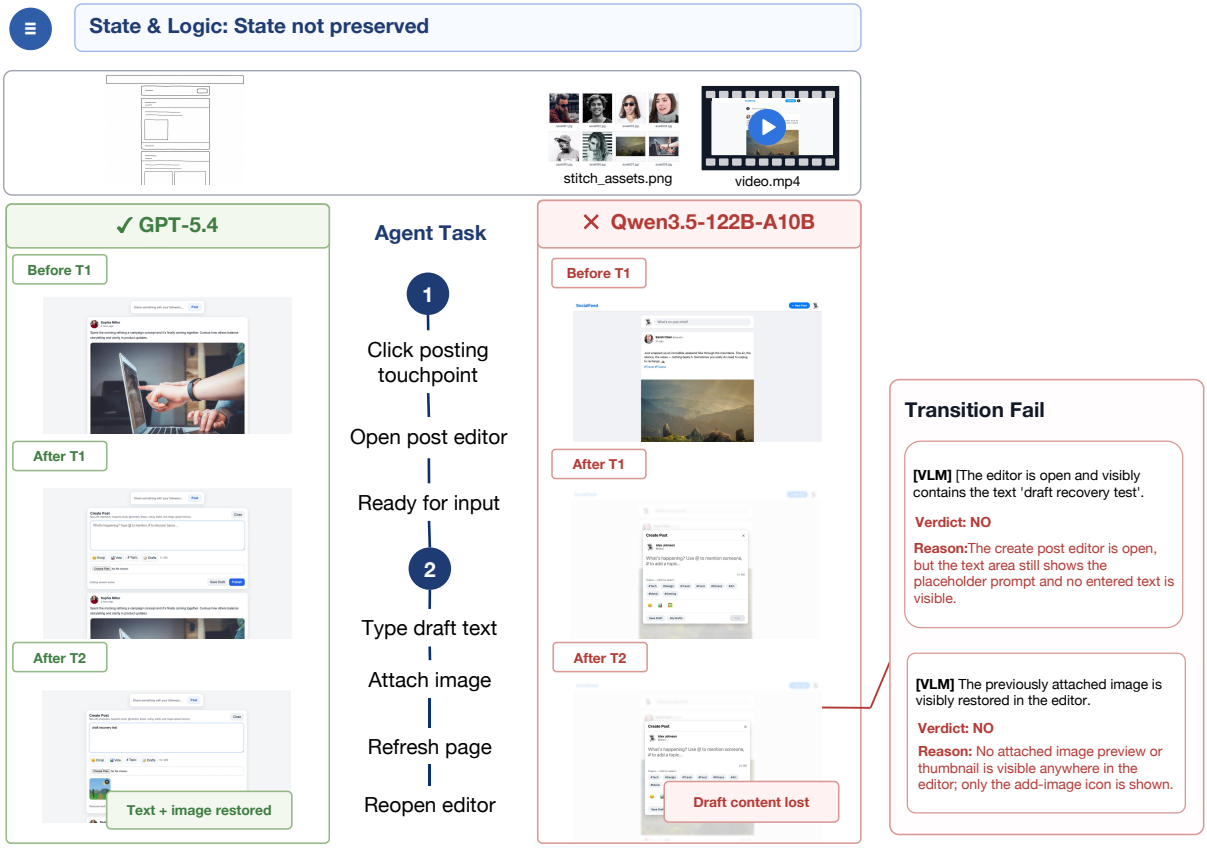

Figure 13: State preservation failure in a draft-recovery workflow. After typing text, attaching an image, refreshing the page, and reopening the editor, the draft should be restored. The failing artifact loses both the entered text and the attached image.

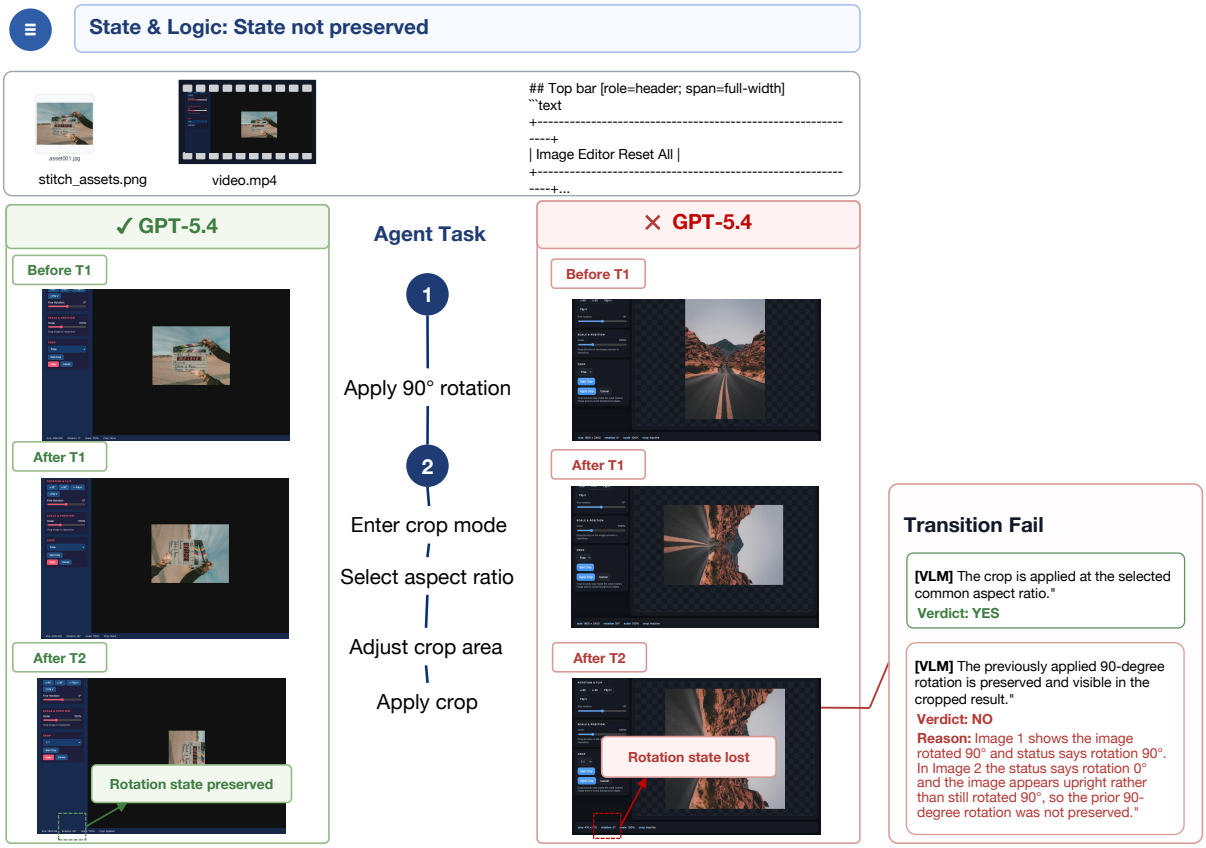

Figure 14: State preservation failure in an image-editing workflow. The transition requires applying a 90-degree rotation, entering crop mode, selecting an aspect ratio, and applying the crop while preserving the prior rotation. The failing artifact applies the crop but resets the rotation state.

Test Data Contract Generation

You are a frontend test specification designer. Given a requirement list, produce a minimal test data contract describing what the page must be functionally ready to do on first load. No reference implementation is provided; derive the contract from the requirements alone.

Rules:

1. Describe functional readiness, not UI structure, visual layout, DOM hierarchy, or exact element counts.
2. For multi-page, multi-view, tabbed, wizard, or navigation-based apps, explicitly name the initial page, view, route, or mode shown on first load.
3. Do not prescribe positions, component hierarchy, styling, specific mock values, asset sources, or exact numbers of items.
4. Include only conditions needed for the first test action to be possible.
5. Do not expose implicit requirements. Remove contract text that reveals behavior beyond the explicit requirements.
6. If a default initial view is not specified but the app requires one, choose a reasonable primary workflow view and state it functionally.

Output: Return exactly one JSON object: `{"test_data_contract": "functional preconditions describing page readiness"}`

Figure 15: Prompt for deriving initial functional readiness from requirements.

Test Item Generation

You are a frontend test specification designer. Produce a test item list covering every testable behavior in the given requirement list. No reference implementation is provided; derive the test items from the requirements alone.

Rules:

1. Generate one test item per distinct testable behavior. Every explicit and implicit requirement ID must appear in at least one item's `req_ids`.
2. Triggers and expected results must be implementation-neutral. A trigger describes user intent, not a click sequence or pure observation; an expected result describes the semantic outcome.
3. Combine tightly coupled behaviors that share the same trigger and artifact, but split named cases with distinct outcomes.
4. Use only primary requirement IDs in `req_ids`, normally at most two per item. If an expected result verifies a requirement, include that requirement ID.
5. Do not invent behaviors. Failure, error, boundary, and follow-up cases must be grounded in named requirements.
6. Implicit requirements may refine explicitly stated behaviors, but must not introduce new scenarios by themselves.
7. Do not create standalone negative-capability items; fold unavailable actions into the expected result of the state-changing item that causes them.
8. Selection-gated controls are modeled as one behavior: the trigger selects the required content and invokes the control, while the expected result covers the gated availability.
9. Guard or prevention behavior may be a separate item only when it has a distinct user trigger and a clearly observable prevention outcome.
10. For toggles or bidirectional behavior, describe switching from the current state to the alternative without assuming a default.
11. Do not split countdown, cooldown, or expiry flows across separate items; keep the complete timed user intent in one item.

Output: Return exactly one JSON object: `{"test_items": [{"item_id": "TI-1", "req_ids": [...], "description": "...", "trigger": "...", "expected_result": "..."}]}`

Figure 16: Prompt for converting requirements into implementation-neutral test items.

ICG Generation

You are a frontend interaction test case designer. Generate a test specification with `states` and `transitions`. Each transition specifies a source state, a target state, a self-contained `agent_task`, mapped test item IDs, `dom_assertions`, and/or visual `postconditions`.

State rules:

1. States are stable, replayable page checkpoints. Use `S0`, `S1`, etc.; `S0` is the initial state.
2. State descriptions are short, implementation-neutral page snapshots. Do not model transient UI such as spinners, toasts, timers, or animations as states.
3. Define a new state whenever visible content, selected controls, open panels, input values, displayed artifacts, or persistent UI state differs materially.
4. Preserve state continuity: unchanged visible aspects from the source state should carry into the target state's description.
5. A self-loop is valid only when the after-state is observably identical to the before-state.

Transition rules:

1. Use sequential IDs `T1`, `T2`, etc. Each transition declares `from`, `to`, `agent_task`, `mapped_test_items`, and at least one non-empty assertion list.
2. `preconditions` are allowed only on `T1`; later transitions must not contain preconditions.
3. Use `dom_assertions` for DOM mutation evidence, temporal evidence, and element-level state. Prefix each DOM assertion with exactly `[CHANGE]` or `[AFTER]`.
4. Use visual `postconditions` for outcomes judged from before/after screenshots. Do not prefix postconditions.
5. The `agent_task` must describe the user's goal from the current `from` state. It must be actionable, self-contained, and not a low-level selector or click sequence.
6. The `agent_task` must not refer to previous transitions; if prior context is needed, describe it as a persistent property of the current source state.
7. Do not create observation-only or self-check transitions. Static checks should be attached to the transition that establishes the checked state.
8. No hitchhiking: every mapped test item must be directly caused by this transition's `agent_task` and verifiable from this transition's final state.
9. Every input test item must be covered by at least one transition.
10. Every mapped test item must have at least one direct `dom_assertion` or visual `postcondition` in the same transition.
11. Independent features should fan out from the same source state rather than being falsely chained; serial chains are used only when the later transition genuinely requires the prior target state.
12. Compound transitions may combine two or three operations only when they share the same artifact and are intended to test state interference or clobbering.
13. Do not invent UI controls, defaults, failure paths, labels, data values, or data assumptions not supported by the Test Data Contract or test items.
14. Empty-state tests must explicitly clear or remove pre-existing content when the Test Data Contract does not guarantee emptiness.

Output: Return exactly one JSON object with top-level fields `states` and `transitions`. Do not include markdown fences or extra prose.

Figure 17: Prompt for generating the state-transition Interaction Contract Graph.

Agent Execution

You are a robot browsing the web to execute a web-testing task. In each iteration, you receive an indexed DOM observation where elements are prefixed with `[N]`, and newly appeared elements may be marked with `*[N]`. Choose exactly one action using indices from the latest observation.

Action grammar:

```
Click [N]
Click [N]; count
Dismiss
DoubleClick [N]
RightClick [N]
LongPress [N]; ms
Hover [N]
Input [N]; text
InputDate [N]; YYYY-MM-DD
Clear [N]
Blur [N]
Select [N]; option label
Check [N] / Uncheck [N]
Press [N]; key
Press [N]; key; count
Scroll [N or WINDOW]; up|down|top|bottom|left|right
Drag [N]; [M]
Drag [N]; offset_x=px,offset_y=px
DragRange [N]; target value
ClickAt; x=px y=px
Upload [N]; file_path
Upload [N]; file_path|file_path
SelectText [N]; text to select
Wait; ms
Refresh
GoBack
Reset
Done
```

Execution constraints: Use only provided test assets for upload, do not bypass a required interaction path with a similar final state, and emit `Done` only after the requested user action has fully executed.

Output: Return exactly one JSON object: `{"thought": "brief reasoning", "action": "ONE Action"}`

Figure 18: Prompt used by the browser agent to execute one transition.

DOM Assertion Scoring

You are a strict UI test evaluator. Determine whether DOM assertions are satisfied based on structured DOM event evidence collected during a web interaction. The evidence contains the action performed, initial and final DOM snapshots, mutation events, changed attributes, added and removed nodes, and interactive-element summaries.

Assertion semantics:

1. `[CHANGE]` means the condition appeared at any point in the full timeline, including initial snapshot, mutation events, intermediate summaries, or final snapshot.
2. `[AFTER]` means the condition must hold in the final stable state. Timeline evidence may help locate the target, but the final state must satisfy the assertion.

Judging rules:

1. Locate elements by semantic role, tag, ID, class, text, attribute changes, and interactive-element summaries.
2. Hidden or `not-visible` text counts as DOM evidence but not as proof that the text is visibly present.
3. For disabled or non-interactive state, prioritize `pointer-events: none`, native `disabled` or `aria-disabled`, then state-indicative class tokens.
4. For selected, active, highlighted, expanded, checked, or pressed states, use ARIA fields first and class tokens as secondary evidence.
5. Added and removed nodes may prove transient feedback such as loading, saving, progress, confirmation, or disappearance.
6. Judge by semantic equivalence rather than exact wording, but be strict on factual contradictions.
7. Prefer `UNCERTAIN` when evidence is incomplete, except when absence from final interactive elements directly supports a non-interactive or absent assertion.
8. For debounce or delayed-update assertions, accept evidence of a single delayed update after input settles rather than requiring keystroke-level events.
9. Do not treat hidden template text as evidence that a visible status or control is active.

Output: Return exactly one JSON object: `{"evaluations": [{"think": "...", "result": "YES|NO|UNCERTAIN"}]}`

Figure 19: Prompt for judging DOM assertions from mutation evidence.

Visual Postcondition Scoring

You are a strict UI test evaluator. Compare two screenshots: Image 1 is before the interaction, and Image 2 is after the interaction. Determine whether each assertion holds on the current page.

Judging rules:

1. Judge by semantic equivalence rather than exact wording; be strict on factual correctness but lenient on terminology.
2. For conditional assertions, determine which branch applies from the screenshots and evaluate only that branch.
3. If content is clipped by the viewport or a scrollable container, evaluate only fully visible items.
4. Accept small numeric changes hidden by rounding or abbreviation when the structural outcome is otherwise correct.
5. For search or filter assertions, an empty result may pass if the filter is visibly active and the page shows a valid empty state.
6. For body or full-text search, do not require every visible result row to display the matching keyword if the active query and changed result set support the outcome.
7. For visually ambiguous natural-image flips, do not answer `UNCERTAIN` solely because the flip is hard to distinguish when other requested edits are clearly visible.
8. Use before/after differences to judge reordering, expansion, collapsed panels, drag placement, list updates, and stale-state removal.

Output: Return exactly one JSON object: `{"evaluations": [{"think": "...", "result": "YES|NO|UNCERTAIN"}]}`

Figure 20: Prompt for judging postconditions from before/after screenshots.