

Do LLM-Generated Web Apps Behave Correctly? A Contract-Grounded Benchmark for Behavioral Conformance

Chansung Park^{1*} Juyong Jiang^{2,3*} Jiasi Shen^{3†} Sunghun Kim^{2†}

¹Electronics and Telecommunications Research Institute

²The Hong Kong University of Science and Technology (Guangzhou)

³The Hong Kong University of Science and Technology

{deep.diver.csp, csjuyongjiang}@gmail.com, {sjs, hunkim}@cse.ust.hk

Abstract

Large language models and coding agents can produce web applications that appear functional, yet surface plausibility is an unreliable proxy for behavioral conformance under interaction. We introduce ConformWeb, a contract-grounded protocol and benchmark that operationalizes public behavioral contracts to construct hidden interaction scenarios, evaluating whether agent-generated applications preserve strict conformance under dynamic execution. By decoupling hidden interaction sequences from the public behavioral scope, the protocol guarantees a fair held-out workflow evaluation without relying on undisclosed reference implementations. Evaluated across 24 validated instances scaling to 756 evaluation scenarios and 6,433 semantic steps, only 252 of the 2,160 total execution runs achieve full conformance, yielding a strict pass rate of 11.7%. While scenario-level conformance reaches 36.7% and step-level progress averages 49.4%, this quantitative gap demonstrates that generative systems frequently achieve substantial partial progress but fail to maintain the critical behavioral invariants required for end-to-end conformance. All code and data are released at <https://github.com/deep-diver/ConformWeb>.

1 Introduction

Large language models and coding agents increasingly generate runnable web applications rather than isolated functions, scripts, or static pages (Lu et al., 2026; Xiao et al., 2025). Although these applications appear functional during brief demonstrations, they merely render pages, display controls, accept form inputs, and initiate familiar workflows as expected. Correctness for an interactive application is not established by a single screen or a local action. It depends on whether behavior remains coherent as actions accumulate across

state, views, routes, permissions, services, and persistence (Trivedi et al., 2024; Hattami et al., 2025). Such coherence requires that submitted data seamlessly propagate to subsequent summary views, role changes strictly dictate actionable permissions, and page reloads faithfully preserve state data that the interface has claimed to save. Generated applications often fail at these boundaries while retaining a plausible surface (Jimenez et al., 2024; Zhu et al., 2025; Peng et al., 2026).

Existing evaluations provide important signals within their respective domains. For instance, code generation benchmarks evaluate functional compliance through isolated input-output mapping (Jimenez et al., 2024; Jain et al., 2025; Zhuo et al., 2025), while visual and front-end benchmarks measure static layout fidelity and structural replication (Si et al., 2025; Lu et al., 2026; Zhu et al., 2025; Xiao et al., 2025). Similarly, web and OS agents are assessed on their ability to complete task-oriented trajectories within pre-configured, stable environments (Zhou et al., 2024; Koh et al., 2024; Drouin et al., 2024; Xie et al., 2024). However, these benchmark settings typically fix at least one side of the interaction boundary, usually providing the callable interface, test harness, target page, or operating environment in advance. The behavioral failures of fully generated applications expose a fundamental shift in this unit of evaluation.

Because generated web applications shift the interaction boundary entirely, the system under test must autonomously produce the necessary controls, routing architectures, and application states by which behavior is validated. Consequently, the core evaluation challenge transitions from verifying static or isolated functional outputs to determining whether the application preserves its declared behavior across dynamic, multi-step operations. This operational vulnerability exposes a fundamental shift in the unit of evaluation, as the generative system can no longer rely on a pre-configured in-

*Equal contributors. Listing order is random.

†Corresponding authors.

frastructure to guide its execution.

Evaluating such generative systems requires a parallel shift in measuring generalization. In traditional machine learning, evaluating models on held-out test sets has long been the foundational standard to prevent overfitting (Kohavi et al., 1995). To assess LLM-based application generation, this testing principle must be extended to verify whether a model can generalize to implied workflows, rather than merely replicating explicit sequences disclosed in the context. Without this rigorous check, generated applications easily become tightly coupled to specific demonstration paths and fracture when encountering alternative compositions of the same requirements. True behavioral evaluation must therefore conceal specific execution paths while verifying conformance within a publicly specified scope.

To address this challenge, we introduce ConformWeb, a contract-grounded protocol and benchmark designed to evaluate behavioral conformance under private interaction scenarios. Each benchmark instance is defined by a public behavioral contract that declares the interactions, observations, and obligations that may be evaluated as workflows unfold. Private evaluation scenarios are admissible only when their actions, evidence channels, and checks are strictly grounded in this public contract. By decoupling the hidden interaction sequences from the public behavioral scope, ConformWeb operationalizes a true held-out workflow testbed. A generated application is thus judged rigorously as a functional implementation of the declared contract, rather than by surface-level visual similarity or by rigid matching against an undisclosed reference implementation.

ConformWeb is instantiated across 24 instances spanning six application families and four complexity tiers, scaling to 756 scenarios over 6,433 semantic steps. To ensure programmatic validity, the benchmark consists exclusively of tasks verified to admit fully conformant implementations under the public contract and evaluation protocol. Across 2,160 comprehensive evaluation runs, strict behavioral conformance remains exceptionally rare, with an overall full-pass rate of only 11.7%. Even the Frontier group reaches only a 24.2% full-pass rate despite achieving 67.2% scenario-level conformance. This sharp divergence underscores our central empirical signal because generated applications successfully build requested controls and navigate substantial workflow prefixes but consis-

tently fracture when long-horizon dependencies demand absolute state or persistence coherence.

Our contributions are summarized as follows.

- We define generated application correctness as coherent behavior across unseen workflow compositions, shifting the evaluation paradigm from local actions or visual similarity.
- We introduce ConformWeb, a contract-grounded framework that decouples public behavioral scope from hidden interaction sequences, ensuring private scenarios remain strictly grounded in the public contract.
- We release 24 instances spanning six families and four complexity tiers, complete with public contracts, multi-stage scenarios, reference implementations, and deterministic browser-executable checks.
- We evaluate 2,160 generations to reveal that full conformance remains rare, diagnosing a critical failure mode where models build plausible surfaces but fracture under stateful, long-horizon dependencies.

2 Related Work

2.1 Web Agent Evaluation

Web interaction benchmarks traditionally evaluate agents that act within pre-existing environments. Established frameworks and recent benchmarks focus on measuring perception, grounding, planning, and task completion across deployed interfaces (Yao et al., 2022; Deng et al., 2023; Zhou et al., 2024; Koh et al., 2024; He et al., 2024; Zheng et al., 2024; Lù et al., 2024; Drouin et al., 2024; Chezelles et al., 2024; Boisvert et al., 2024). In these settings, the website, task environment, and interaction substrate remain fixed throughout the evaluation.

ConformWeb evaluates a fundamentally different object. The generative system must produce the web application itself from a public behavioral specification, and correctness is determined by whether the generated application preserves that behavior under private browser trajectories. This formulation shifts the unit of evaluation from a single action or a graded agent trace to an entire user workflow. Furthermore, unlike automatic trajectory grading and LLM judge based evaluation (Zheng et al., 2023; Liu et al., 2023b; Lù et al., 2025),

ConformWeb treats behavioral conformance as a deterministic testing problem. Private scenarios function as executable tests strictly grounded in the public contract rather than qualitative judgments over an execution trajectory.

This perspective connects ConformWeb to specification based testing, model based testing, and conformance testing, where correctness is defined by adherence to a specification under systematically generated tests (Tretmans, 1996, 2008; Claessen and Hughes, 2000). Similarly, GUI and web testing research has long explored event sequences, interface models, crawling, and interaction driven failures (Memon et al., 2002; Memon, 2007; Mesbah et al., 2012; Hammoud et al., 2017). ConformWeb adapts these principles to the domain of LLM generated web applications. The public contract serves simultaneously as the generation specification, the boundary for admissible private tests, and the source of executable browser checks. While private scenarios can instantiate unique values and compose declared actions in unseen orders, they cannot introduce behavior absent from the public contract. This design guarantees that the hidden evaluation remains fair because the workflow sequences are concealed while the behavioral scope remains fully public.

2.2 Code and Web Application Generation

Code generation benchmarks typically evaluate the functional correctness of isolated functions, scripts, programs, or repository patches (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Lai et al., 2023; Lu et al., 2021; Puri et al., 2021; Casano et al., 2023; Liu et al., 2023a; Jain et al., 2025; Zhuo et al., 2025; Jimenez et al., 2024; Zan et al., 2026). In parallel, interface and website generation benchmarks focus on visual reconstruction, front end layout generation, component behavior, multimodal page understanding, or page level functionality (Beltramelli, 2018; Si et al., 2025; Lin et al., 2025; Awal et al., 2025; Lu et al., 2026; Zhu et al., 2025; Xu et al., 2025). While these evaluations provide vital executable and visual signals, the unit under test is restricted to a single function, component, or disclosed task.

ConformWeb instead treats the generated artifact as a stateful, end to end web application navigated through dynamic browser trajectories. Rather than measuring surface similarity or performance on a single disclosed workflow, correctness is defined by full-layer behavioral conformance when declared

obligations are composed in undisclosed private scenarios.

3 ConformWeb Protocol

ConformWeb operationalizes the public contract principle into an executable evaluation protocol where each benchmark instance comprises a product premise, a public behavioral contract, and both public and private scenarios. While public scenarios provide disclosed representative workflows, private scenarios challenge the application with held-out workflow compositions that remain strictly grounded within the same public contract. Consequently, a generated application is evaluated based on its compliance with declared behaviors rather than its proximity to an undisclosed reference implementation. By isolating surface realization from public and private scenario conformance, the protocol successfully distinguishes between superficial interface omissions, breakdowns on disclosed tasks, and deep failures that emerge only under stateful composition. Figure 1 illustrates this operational framework, and Appendix A details the underlying contract language, compiler, and browser runner.

3.1 Contracts and Private Scenarios

The contract is the object that makes hidden evaluation admissible. It specifies which actions may be taken, which observations may be used as evidence, and which behavioral relations may be checked. We represent a public behavioral contract as

$$C = (\mathcal{A}_C, \mathcal{O}_C, \mathcal{R}_C). \quad (1)$$

The action set $\mathcal{A}_C$ defines the browser operations admitted by the contract, encompassing control actions, input updates, uploads, role changes, navigation, and reloads. The observation set $\mathcal{O}_C$ defines the public evidence channels available to the evaluator, which include application state, browser metadata, service calls, and rendered projections. The relation set $\mathcal{R}_C$ defines behavioral checks over observations, capturing state transitions, validation behavior, rendered projections, persistence, and authorization effects.

A private scenario τ is admissible, written $\tau \preceq C$, only if all of its references can be resolved through the public contract. Each action must instantiate an element of $\mathcal{A}_C$, each observation reference must be drawn from $\mathcal{O}_C$, and each check must instantiate a relation in $\mathcal{R}_C$. The arguments to these actions and checks may use only declared selectors, roles, routes, service fixtures, upload schemas,

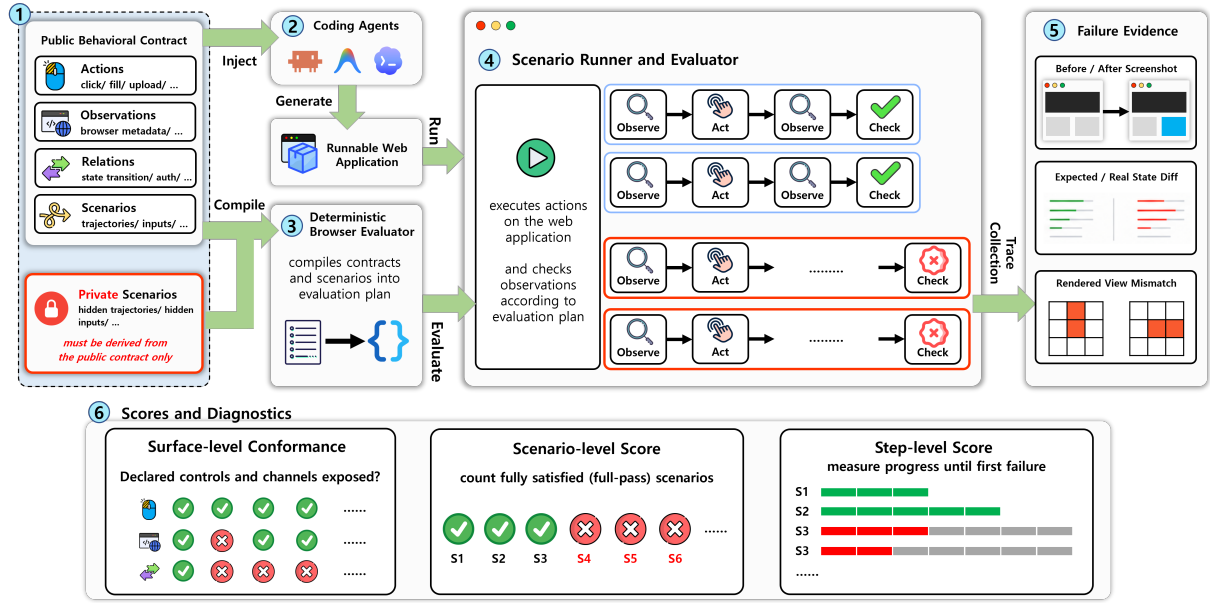


Figure 1: Overview of ConformWeb. The evaluation operates in six integrated stages: (1) Contract Definition, specifying public behaviors that bound the generation of undisclosed private scenarios; (2) Application Generation, where agents ingest the public contract to build a runnable web application; (3) Evaluation Planning, compiling contracts and hidden workflows into deterministic execution plans; (4) Scenario Execution, running multi-stage browser trajectories while checking live observations; (5) Trace Collection, capturing state diffs and rendered-projection mismatches upon contract violations; and (6) Granular Diagnostics, stratifying scores across surface-level exposure, scenario-level completion, and step-level progress.

tables, state paths, and evidence channels. A private scenario may choose concrete values, order declared actions, and compose declared behaviors, but it cannot introduce new product requirements. A scenario that cannot be derived from the contract represents a construction error in the benchmark instance rather than an implementation failure of a generated application.

3.2 Scenario Execution Semantics

For a generated application G and an admissible private scenario τ , we write the sequence as

$$\tau = ((a_i, Q_i))_{i=1}^n, \quad a_i \in \mathcal{A}_C. \quad (2)$$

In this formulation, a_i is a declared browser action and Q_i is the finite set of checks evaluated after that action. Each check $q \in Q_i$ instantiates a behavioral relation from $\mathcal{R}_C$ using paths, values, roles, routes, or evidence channels declared by the contract. Executing τ on G produces a browser trace over the observation surface $\mathcal{O}_C$ expressed as

$$o_0, a_1, o_1, \dots, a_n, o_n. \quad (3)$$

The initial observation o_0 is recorded before the first action, and each subsequent o_i is recorded immediately following action a_i . The application

satisfies τ under contract C , written $G \models_C \tau$, when every individual step passes. For $i = 1, \dots, n$, we define the step-level passage as

$$\text{pass}_i(G, \tau) = \text{exec}_G(a_i) \wedge \text{eval}(Q_i, o_{i-1}, o_i), \quad (4)$$

$$\text{where } \text{eval}(Q_i, o_{i-1}, o_i) = \bigwedge_{q \in Q_i} q(o_{i-1}, o_i). \quad (5)$$

Passing eval requires the joint satisfaction of all behavioral relations checked at step i , taking both o_{i-1} and o_i to evaluate state transitions and persistence effects across the action boundary. Scenario satisfaction is consequently established as

$$G \models_C \tau \iff \forall i \in 1, \dots, n, \text{pass}_i(G, \tau). \quad (6)$$

The term $\text{exec}_G(a_i)$ explicitly integrates actionable execution into the conformance metric. Rather than relying on evaluator heuristics, incorporating $\text{exec}_G(a_i)$ ensures that interactions are actively permitted at the current workflow state and that observations are captured at the exact junctions declared by the contract.

3.3 Scores and Failure Evidence

Private scenarios provide the held-out composition pressure in ConformWeb, while public scenarios

capture conformance to disclosed workflows. In our primary experiments, benchmark scores are computed jointly over the public and private scenario sets. Let P_C and T_C denote the scored public and private scenario sets for contract C , respectively, and let $S_C = P_C \cup T_C$ be the resulting scoring set. ConformWeb aggregates evaluation through two primary metrics. The scenario-level completion score counts fully satisfied scoring scenarios, and the step-level progress score measures execution advancement prior to the first failure:

$$s_{\text{scen}}(G) = \frac{\sum_{\tau \in S_C} w_{\tau} \mathbf{1}[G \models_C \tau]}{\sum_{\tau \in S_C} w_{\tau}}, \quad (7a)$$

$$s_{\text{step}}(G) = \frac{\sum_{\tau \in S_C} w_{\tau} p_{\tau}(G)}{\sum_{\tau \in S_C} w_{\tau}}. \quad (7b)$$

In our framework, w_{τ} represents the scenario weight, set uniformly across all instances in our experiments. For a scenario $\tau = ((a_i, Q_i))_{i=1}^{n_{\tau}}$, let $\ell_{\tau}(G)$ be the length of the longest prefix where every step passes, defining the progress ratio as

$$p_{\tau}(G) = \frac{\ell_{\tau}(G)}{n_{\tau}}. \quad (8)$$

Consequently, $p_{\tau}(G) = 1$ corresponds exactly to full satisfaction ($G \models_C \tau$), whereas lower values directly quantify how far the application progresses before encountering its first failed action or check.

4 Benchmark Suite

4.1 Application Families and Tiers

ConformWeb instantiates the protocol across 24 distinct benchmark instances formed by crossing six application families with four complexity tiers, scaling to 756 scenarios over 6,433 semantic steps. Rather than evaluating isolated tasks, each instance demands that coding agents implement a fully functional application solely from a public specification, which is then challenged under undisclosed private workflows. This design ensures that the evaluation measures genuine conformance to declared behavioral contracts rather than the superficial imitation of a hidden reference application.

As summarized in Table 1, the suite isolates specific failure modes by progressively increasing behavioral pressure across the four tiers. Tier A establishes local consistency, Tiers B and C introduce operational views and policy constraints, and Tier D pushes execution to final release stages where long-horizon state synchronization must remain

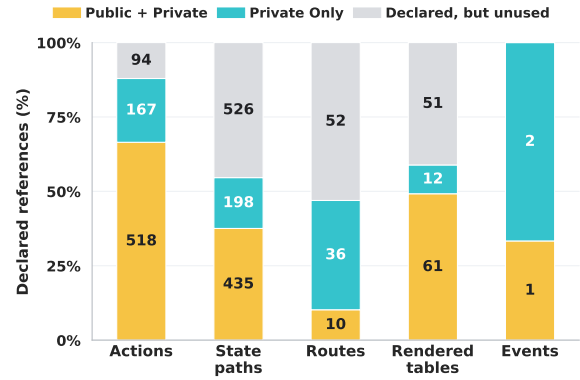


Figure 2: Distribution of declared contract references across evaluation scenarios. Yellow segments indicate references shared between public and private scenarios, cyan indicates private-only references, and gray indicates declared references left entirely unscored.

absolute. This structured depth creates an exceptionally high bar for current LLMs, evidenced by a baseline full-pass rate of merely 11.7%. As detailed in Section 5, even state-of-the-art frontier models consistently fracture under this pressure, revealing clear performance gaps when multi-view consistency is rigorously demanded.

4.2 Contract Reference Coverage

To guarantee that ConformWeb evaluates genuine programmatic generalization rather than memorization, private scenarios must be hidden in their sequential composition but fully exposed in their behavioral scope. To audit this boundary, we track how scenario references to actions, state paths, routes, rendered tables, and event channels resolve back to the public contract. Figure 2 details this structural distribution across the entire suite, confirming that every private test segment remains strictly grounded within the pre-declared public contract.

The empirical distribution reveals a deliberate generalization pressure. As shown by the yellow blocks, a significant portion of private checking reuses references already disclosed in public workflows, forcing models to compose known primitives into new stateful orders. Crucially, the cyan segments denote declared references exercised only in private scenarios, creating held-out pressure without introducing undeclared behavior. Finally, the gray segments verify that the contract is not a thin wrapper or a disguised cheat-sheet for the scoring engine; it leaves ample declared surface unused, penalizing agents that rely on surface-level visual imitation rather than robust contract fulfillment.

Table 1: Complexity tiers in ConformWeb. The tiers progress from short local workflows to longer sequences involving operational views, policy constraints, deterministic services, uploads, persistence, and release synchronization.

Tier	Name	Target scope	Behavioral pressure	Scoring evidence
A	Consumer	Short user workflows	Local state, validation, simple service responses, route transitions, reload behavior	State deltas, validation outcomes, totals, persistence checks
B	Operational	Staff and fulfillment workflows	Operational views, ordered collections, derived counters, status changes, rendered tables	Collection updates, workflow status, counters, rendered rows
C	Regulated	Role and policy workflows	Permission gates, uploads, deterministic services, conflict branches, persistent records	Unauthorized effects, parsed upload state, service outcomes, policy traces
D	Release	Final approval and release workflows	Public ledgers, browser history, re-open behavior, release synchronization, long action chains	Multi-view consistency, history and reload behavior, final ledger checks

4.3 Instance Construction and Validation

Each instance is admitted only after rigorous executable validation. The product premise and public contract are authored first, and both public and private scenarios are subsequently compiled against these declared contract elements. Any scenario referencing undeclared behavior is rejected during construction. For accepted instances, a reference implementation is initialized from the public materials and iteratively validated against private scenarios using limited failure summaries, without access to private scenario files, full traces, screenshots, or expected trajectories. This reference application serves as an executable witness that the contract, scenarios, compiler, and evaluator jointly admit a conformant implementation.

While the reference application establishes executability, blind generation outcomes provide complementary empirical evidence that the held-out private scenarios are attainable in practice beyond the reference construction itself under the same public disclosure boundary without leaking evaluation artifacts. As documented in Panel A of Table 2, 252 out of 2,160 primary single-shot generated applications achieved a 100% full-pass rate across all scoring scenarios, requiring satisfaction of both disclosed public and held-out private workflows. These full-pass runs are not isolated anomalies; they span every complexity tier, distributing as 116/540 in Tier A, 82/540 in Tier B, 44/540 in Tier C, and 10/540 in Tier D. This distribution shows that attainability is not confined to simpler instances, while successful conformance becomes progressively less frequent as sequential workflow pressure increases. Detailed benchmark statistics and representative surfaces are provided in Appen-

dices B and C, while Appendix G documents reference construction and validation.

5 Evaluation

5.1 Experimental Setup

Models and Baseline Selection. We evaluate nine representative LLMs spanning three leading commercial providers directly over the benchmark suite detailed in Section 4. We select the latest model families from OpenAI (OpenAI, 2026), Anthropic (Anthropic, 2026), and Google (Google DeepMind, 2026), categorizing them into three capacity bands based on operational cost and computing scale. The Mini band contains GPT-5.4-nano, Claude Haiku 4.5, and Gemini 3.1 Flash Lite. The Turbo band evaluates GPT-5.4-mini, Claude Sonnet 4.6, and Gemini 3.1 Flash. The Frontier band includes the flagship reasoning capabilities of GPT-5.4, Claude Opus 4.6, and Gemini 3.1 Pro Preview. To mitigate inference variance and ensure statistical robustness, each model generates ten independent samples per instance, producing a unified matrix of exactly 2,160 execution runs.

Generation and Evaluation Pipeline. For every family-tier instance, each model receives identical public inputs consisting of the product premise, contract, and disclosed scenarios, while the fixed prompt template strictly excludes private scenarios, reference implementations, and execution traces. In the primary single-shot setting, generated applications are collected as standard static web artifacts in HTML, CSS, and JavaScript. For multi-turn repair experiments, we additionally provide execution feedback summarized from the first failing public scenario to let the model patch its application.

Table 2: Quantitative evaluation of LLM behavioral conformance and complexity tier stress. Panel A establishes the baseline distribution for the primary single shot evaluation. Panel B itemizes detailed conformance metrics and full-pass rates across capability bands and complexity tiers. All reported values reflect explicit raw execution counts without statistical inference or group-level interpolation.

Group	Model	Full pass	S_{scen}	S_{step}	Complexity Tier			
					Tier A	Tier B	Tier C	Tier D
<i>Panel A: Blind full-pass evidence</i>								
All	Total aggregate	252/2160 (11.7%)	36.7%	49.4%	116/540 (21.5%)	82/540 (15.2%)	44/540 (8.1%)	10/540 (1.9%)
<i>Panel B: model-level conformance and tier stress</i>								
Mini	Group aggregate	6/720 (0.8%)	10.9%	24.8%	1/180 (0.6%)	2/180 (1.1%)	3/180 (1.7%)	0/180 (0.0%)
	GPT-5.4 nano	5/240 (2.1%)	17.5%	32.0%	0/60 (0.0%)	2/60 (3.3%)	3/60 (5.0%)	0/60 (0.0%)
	Claude Haiku 4.5	1/240 (0.4%)	14.3%	31.5%	1/60 (1.7%)	0/60 (0.0%)	0/60 (0.0%)	0/60 (0.0%)
	Gemini 3.1 Flash Lite	0/240 (0.0%)	0.9%	10.9%	0/60 (0.0%)	0/60 (0.0%)	0/60 (0.0%)	0/60 (0.0%)
Turbo	Group aggregate	72/720 (10.0%)	31.9%	43.9%	35/180 (19.4%)	22/180 (12.2%)	12/180 (6.7%)	3/180 (1.7%)
	GPT-5.4 mini	3/240 (1.2%)	4.3%	13.5%	3/60 (5.0%)	0/60 (0.0%)	0/60 (0.0%)	0/60 (0.0%)
	Claude Sonnet 4.6	46/240 (19.2%)	55.0%	67.4%	19/60 (31.7%)	14/60 (23.3%)	10/60 (16.7%)	3/60 (5.0%)
	Gemini 3.1 Flash	23/240 (9.6%)	36.5%	50.9%	13/60 (21.7%)	8/60 (13.3%)	2/60 (3.3%)	0/60 (0.0%)
Frontier	Group aggregate	174/720 (24.2%)	67.2%	79.3%	80/180 (44.4%)	58/180 (32.2%)	29/180 (16.1%)	7/180 (3.9%)
	GPT-5.4	47/240 (19.6%)	68.4%	81.0%	23/60 (38.3%)	17/60 (28.3%)	3/60 (5.0%)	4/60 (6.7%)
	Claude Opus 4.6	69/240 (28.8%)	70.8%	82.3%	32/60 (53.3%)	22/60 (36.7%)	15/60 (25.0%)	0/60 (0.0%)
	Gemini 3.1 Pro Preview	58/240 (24.2%)	62.3%	74.7%	25/60 (41.7%)	19/60 (31.7%)	11/60 (18.3%)	3/60 (5.0%)

The backend-backed setting is evaluated separately as a controlled architectural variant in Section 5.3. Across settings, the evaluation pipeline employs a deterministic Playwright and Chromium runner that initializes each scenario in a fresh browser context, executes the compiled browser plan, records contract-declared state transitions and rendered evidence, and applies rule-based assertions.

Metrics. We report the full pass rate along with strict scenario-level conformance S_{scen} and step-level progress S_{step} . The full pass metric requires every scoring scenario within an instance to succeed. The S_{scen} score credits only fully passed scoring scenarios while S_{step} measures the completed execution prefix before the initial behavioral failure. To analyze performance across varying complexity, we further disaggregate these metrics by complexity tiers A through D, reflecting increasing behavioral requirements.

Further details on the experimental protocols, information boundaries, browser execution settings, and run provenance are provided in Appendix H.

5.2 Main Results

Full Conformance is Rare. Across the 2,160 execution runs, only 252 pass the full scoring set, yielding an 11.7% full pass rate. Partial metrics are substantially higher as scenario level conformance reaches 36.7% and step level progress

reaches 49.4%. This gap is our central empirical signal. Generated applications often instantiate the requested surfaces and complete meaningful workflow prefixes but still fail to preserve all declared behaviors end to end. Detailed breakdowns across all application families are provided in Appendix D. Full pass is strict by design because a user facing workflow fails if a single state transition, permission check, or final ledger update contradicts a prior obligation.

Capability Improves Progress Over Conformance. Table 2 demonstrates that stronger models advance further, yet gains are more pronounced in partial progress than full conformance. The Mini band achieves a 0.8% full pass rate with 6 per 720 runs. The Turbo band reaches 10.0% with 72 per 720 runs. The Frontier band attains 24.2% with 174 per 720 runs. This scaling trend is mirrored in partial metrics where Frontier models reach 67.2% scenario conformance and 79.3% step progress. This rules out a simple surface-generation interpretation. Even the most capable models often preserve local obligations but fail when later operations depend on earlier state, roles, or service persistence.

Higher Tiers Stress Workflow Consistency. The tiered results follow the complexity progression defined in Section 4. Full pass rates decline across tiers, from 21.5% (116/540) in Tier A to

15.2% (82/540) in Tier B, 8.1% (44/540) in Tier C, and 1.9% (10/540) in Tier D. This decline is consistent with the tier design, as later tiers require state, permissions, and browser views to remain synchronized across longer interactions. Table 2 shows that this pressure persists across all capability bands. Higher tiers are not merely longer tasks; they introduce tighter coupling among browser state, role policies, and service responses, effectively separating shallow surface completion from sustained behavioral conformance.

5.3 Diagnosing the Conformance Gap

To test whether the observed conformance gap stems from architectural limits or the absence of repair feedback, we conduct two experiments: back-end separation and multi-round repair with public execution feedback.

Backend Separation. To isolate the effect of client-side state management, we independently generated backend-backed full-stack applications with server-side JSON persistence under the identical benchmark design. As shown in Table 3, introducing server-side persistence does not resolve the conformance bottleneck. The strict full pass remains unchanged at 11.7%, and partial metrics show no meaningful gain. This confirms that server-side architecture alone is insufficient to bridge the behavioral conformance gap.

Table 3: Impact of backend separation.

Setting	Full Pass	S_{scen}	S_{step}
Static client	252/2160 (11.7%)	36.7%	49.4%
Backend backed	252/2160 (11.7%)	36.0%	45.8%
<i>Change</i>	<i>0 runs (0.0 pts)</i>	<i>-0.7 pts</i>	<i>-3.6 pts</i>

Multi-round Repair with Execution Feedback.

We evaluate whether models can self-correct when provided with execution feedback from the first failing public scenario. In each round, the model patches its code based on failure summaries and is re-evaluated under the unchanged protocol (Tables 4 and 5). While public execution feedback steadily improves all metrics, the recovery remains limited. Strict full pass reaches only 23.4% after two rounds, and complex long-horizon workflows in Tier D remain largely unsolved at 7.0%. This demonstrates that while feedback helps, the behavioral conformance gap cannot be attributed solely to the lack of interactive repair.

Table 4: Overall performance across multi-round repair.

Stage	Full Pass	S_{scen}	S_{step}
Initial	252/2160 (11.7%)	36.7%	49.4%
Repair 1	406/2160 (18.8%)	44.0%	53.7%
Repair 2	505/2160 (23.4%)	48.1%	57.6%

Table 5: Tier-wise full-pass progression under repair.

Tier	Initial	Repair 1	Repair 2
Tier A	21.5% (116)	31.3% (169)	37.0% (200)
Tier B	15.2% (82)	25.9% (140)	32.4% (175)
Tier C	8.1% (44)	14.1% (76)	17.0% (92)
Tier D	1.9% (10)	3.9% (21)	7.0% (38)

5.4 Failure Analysis

ConformWeb failures differ in both type and timing. We define progress as the length of the successful prefix sequence that a model completes before the first behavioral violation. For each failed scenario, we analyze the first step after that prefix to obtain a comparable diagnosis point. Figure 3 maps these initial breakdown points across application tiers and workflow stages. We further distinguish surface-access failures, including missing surfaces, invalid observations, and unavailable controls, from post-exposure behavioral violations, including wrong transitions, route or persistence drift, and projection mismatches. Within this framework, long horizon drift characterizes cases where one of these violations surfaces only after a substantial prefix sequence.

Surface-access failures explain only part of the observed gap. Among the 2,160 runs, 1,614 show no observed surface-availability failure, yet their full-pass rate remains only 15.4%. A conservative audit of 253 surface-related first failures found that none could be resolved through surface-only fixes without changing application behavior, as summarized in Table 6. We also continue failed scenarios diagnostically beyond the first failure without changing the benchmark scores. Among failed scenario executions, 51.8% show an isolated failure, 26.0% become blocked, 20.6% exhibit cascading violations, and 1.7% show multiple independent failures. Thus, low step-level progress reflects both localized defects and broader workflow disruption.

Failures also emerge late in operational sequences, often reflecting a decoupling between interaction and evidence: actions execute without corresponding state changes, state changes fail to update rendered views, role changes lack permission effects, or reloads lose persistence. Substantial

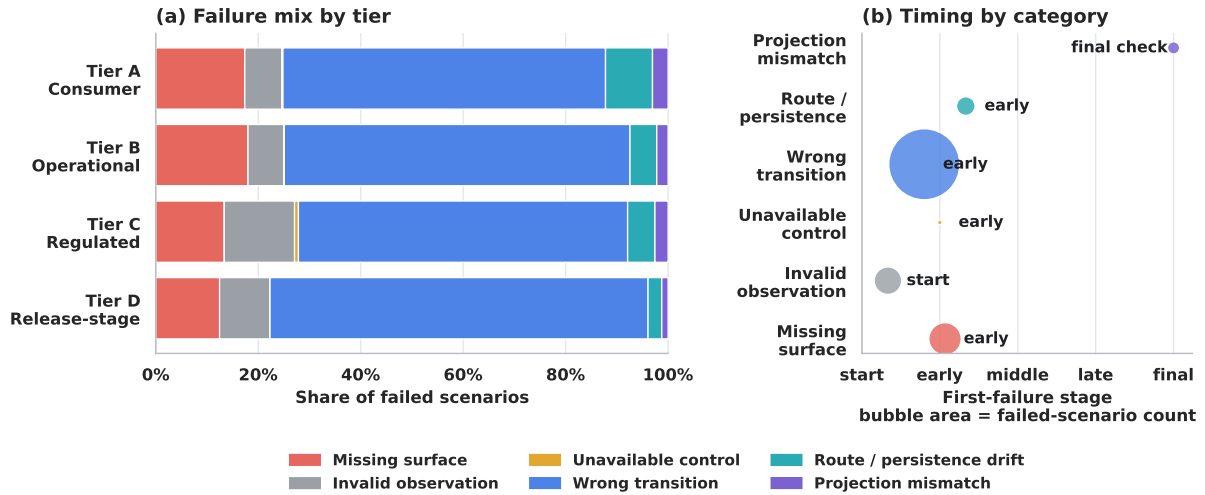


Figure 3: Taxonomy and temporal dynamics of first behavioral failures. (a) Relative distribution of the six failure categories across complexity tier A through D. (b) Temporal positioning of each failure category within normalized workflows, where the bubble area corresponds to the failed scenario count.

Table 6: Surface-compatibility audit of 253 first failures.

Required change	Count (%)
Route or behavior modification	91 (36.0)
Missing underlying state	74 (29.2)
Ambiguous controls	48 (19.0)
Semantically disabled control	40 (15.8)
Total	253 (100.0)

initial progress therefore does not imply end-to-end behavioral coherence. Because the interface can remain plausible after these violations, static inspection is insufficient; detecting them requires dynamic browser execution and contract-grounded assertions. More detailed empirical breakdowns are provided in Appendices E and F.

6 Conclusion

ConformWeb evaluates agent-generated web applications using public and held-out private browser trajectories grounded in public behavioral contracts. Across 2,160 evaluation runs spanning 24 instances, 756 scenarios, and 6,433 semantic steps, only 252 generations achieve full conformance, yielding an 11.7% pass rate. The contrast between high partial progress and the low full-conformance rate highlights a systemic vulnerability where scaling capability fails to guarantee consistency due to a structural decoupling between interaction and evidence. Because these violations evade static inspections by leaving the interface visually intact, ConformWeb establishes a new evaluation paradigm. It shifts the testing unit from isolated agent trajectories to

full-system behavioral conformance under hidden workflow compositions.

Limitations

ConformWeb evaluates applications against public behavioral contracts and held-out tests, so it does not measure open-ended product design quality, accessibility, security, or maintainability. The current suite focuses on stateful workflow applications with a public observation surface, and its contract and scenario semantics are manually designed, so independent semantic validation remains limited. Some diagnostic analyses depend on trace-level artifacts, while the main aggregate results are computed over all validated runs.

Acknowledgements

This work was supported by IITP grant funded by the Korea government(MSIT)[RS-2023-00215959, Development of Access Agnostic wired and wireless integrated optical access technology], the Hong Kong Research Grants Council (Project No. 26216025), and the Guangdong Science and Technology Department (Internal Project No. G_2025_082).

References

- Anthropic. 2026. The Claude Model Card and Evaluation Suite. Technical report, Anthropic. Accessed: 2026-05-26.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten

- Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Rabiul Awal, Mahsa Massoud, Aarash Feizi, Zichao Li, Suyuchen Wang, Christopher Pal, Aishwarya Agrawal, David Vazquez, Siva Reddy, Juan A. Rodriguez, Perouz Taslakian, Spandana Gella, and Sai Rajeswar. 2025. [WebMMU: A benchmark for multi-modal multilingual website understanding and code generation](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25118–25145, Suzhou, China. Association for Computational Linguistics.
- Tony Beltramelli. 2018. [pix2code: Generating code from a graphical user interface screenshot](#). In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '18*, New York, NY, USA. Association for Computing Machinery.
- Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault L De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. 2024. Workarena+: Towards compositional planning and reasoning-based common knowledge work tasks. *Advances in Neural Information Processing Systems*, 37:5996–6051.
- Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2023. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- De Chezelles, Thibault Le Sellier, Sahar Omid Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F Xu, Siva Reddy, Quentin Cappart, et al. 2024. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*.
- Koen Claessen and John Hughes. 2000. Quickcheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 268–279.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.
- Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*.
- Google DeepMind. 2026. Gemini Model Registry and API Specifications. Technical report, Google DeepMind. Accessed: 2026-05-26.
- Dalal Hammoud, Fadi A Zaraket, and Wes Masri. 2017. Guicop: Approach and toolset for specification-based gui testing. *Software Testing, Verification and Reliability*, 27(8):e1642.
- Amine El Hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. 2025. [Webarena verified: Reliable evaluation for web agents](#). In *Workshop on Scaling Environments for Agents*.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890.
- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.
- Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.
- Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.
- Ron Kohavi et al. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada.
- Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel

- Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.
- Zhiyu Lin, Zhengda Zhou, Zhiyuan Zhao, Tianrui Wan, Yilun Ma, Junyu Gao, and Xuelong Li. 2025. Webuibench: a comprehensive benchmark for evaluating multimodal large language models in webui-to-code. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 15780–15797.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY, USA. Curran Associates Inc.
- Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 2511–2522.
- Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.
- Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.
- Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J Pal, and Siva Reddy. 2025. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*.
- Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2026. Webgenbench: Evaluating llms on generating interactive and functional websites from scratch. *Advances in Neural Information Processing Systems*, 38.
- Atif M Memon. 2007. An event-flow model of gui-based applications for testing. *Software testing, verification and reliability*, 17(3):137–157.
- Atif M Memon, Martha E Pollack, and Mary Lou Soffa. 2002. Hierarchical gui test case generation using automated planning. *IEEE transactions on software engineering*, 27(2):144–155.
- Ali Mesbah, Arie Van Deursen, and Stefan Lenselink. 2012. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB)*, 6(1):1–30.
- OpenAI. 2026. OpenAI Model Index and Capability Documentation. <https://platform.openai.com/docs/models>. Accessed: 2026-05-26.
- Zhiyuan Peng, Wei Tao, Xin Yin, Chenhao Ying, Yuan Luo, and Yiwen Guo. 2026. Playcoder: Making llm-generated gui code playable. *arXiv preprint arXiv:2604.19742*.
- Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. 2021. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*.
- 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.
- Gerrit Jan Tretmans. 1996. Test generation with inputs, outputs and repetitive quiescence. *Software: concepts & tools*, 17(3):103–120.
- Jan Tretmans. 2008. Model based testing with labelled transition systems. In *Formal methods and testing: An outcome of the FORTEST network, revised selected papers*, pages 1–38. Springer.
- Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, Bangkok, Thailand. Association for Computational Linguistics.
- 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.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Oworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.
- Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. 2025. Web-bench: A llm code benchmark based on web standards and frameworks. *arXiv preprint arXiv:2505.07473*.

- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.
- Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Li Aoyan, Lu Chen, Xiaojian Zhong, et al. 2026. Multi-swebench: A multilingual benchmark for issue resolving. *Advances in Neural Information Processing Systems*, 38.
- Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.
- 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*.
- Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pages 66602–66656.

A Contract Language and Browser Evaluation

ConformWeb uses a structured contract language so that application generation and evaluation share the same public specification. In the primary blind generation setting, the public contract and disclosed public scenarios are available during generation, while private scenario instances, action orderings, concrete private values, reference implementations, screenshots, traces, and private failure feedback are not disclosed.

Artifact-level auditability requires connecting the formal view in Section 3 to the packaged benchmark files. The artifact represents actions, observations, and behavioral relations in YAML, checks private-scenario admissibility against the public contract, and lowers scenario steps into browser plans. Private scenarios may select values and compose declared behaviors, but the compiler rejects references to undeclared product requirements.

A.1 Contract Files and Disclosure Boundary

Each benchmark instance contains a public `contract.dsl.yaml` file and scenario files when available. The public contract declares the admissible interaction and evidence surfaces. Private scenarios may choose values and compose declared behaviors, but they may not introduce new selectors, routes, state paths, service responses, table schemas, or persistence obligations.

Application generation receives the product premise, public contract, and disclosed public scenarios. The evaluator then tests new compositions of the same declared vocabulary. A failure is therefore interpretable as a conformance failure to the public contract rather than as a penalty for not guessing an undisclosed feature.

Listing 1: Public contract file skeleton.

```
dsl_version: "3.0.0"
app:
  id: <target id>
  name: <display name>
  capability: <short capability label>
  ui_ux_brief: <product premise or interface brief>
runtime:
  target: web
  state_probe:
    expression: <JavaScript expression returning observable
      application state>
  pages: {...}
  session: {...}
  persistence: {...}
  actors: {...}
  services: {...}
  events: {...}
  tables: {...}
state:
  initial: {...}
  schema: {...}
```

```
components:
  <component_id>: {...}
```

A.2 Representative Contract and Scenario Excerpts

The following excerpts are representative fragments from the packaged benchmark rather than synthetic examples. Listing 2 shows how an action declares both allowed and unauthorized effects, while Listing 3 shows how a private scenario composes declared actions and checks.

Listing 2: Role-gated action with allowed and unauthorized effects.

```
components:
  advisor_override_button:
    selector: '[data-testid="advisor-override"]'
    page: audit
    actions:
      click:
        allowed_roles: [advisor]
        preconditions:
          - {op: one_of, path: eligibility_status, values:
            [advisor_required]}
        effects:
          - {op: equals, path: eligibility_status, value:
            eligible}
          - {op: equals, path: approval_status, value:
            advisor_approved}
        unauthorized_effects:
          - {op: changed_by, path: denied_count, by: 1}
          - {op: equals, path: last_action, value:
            unauthorized_advisor}
```

Listing 3: Private scenario composed from declared actions and checks.

```
- id: private_01_ac_quote_and_acceptance_journey
  tier: journey
  steps:
    - {do: reset_button.click}
    - {do: select_ac_button.click}
    - {do: go_booking_button.click}
    - {do: slot_today_button.click}
    - do: customer_name_input.fill
      input: {value: Mina Park}
    - do: address_input.fill
      input: {value: 88 Cedar House}
    - do: get_quote_button.click
      expect:
        - {op: equals, path: quote_amount, value: 18900}
    - {do: approve_quote_button.click}
    - {do: book_visit_button.click}
    - {do: go_visits_button.click}
    - {do: role_pro_button.click}
    - do: pro_accept_button.click
      expect:
        - {op: equals, path: booking_status, value:
          accepted}
```

A.3 Scenario Step Lowering

A component action is not evaluated directly from the YAML file. The compiler inserts a pre-action snapshot and emits the browser action together with post-action checks. Listing 4 shows the lowering of one HomeFix quote step. The snapshot fixes the before state used by delta checks, while the action step records the post-action evidence used by the assertion engine.

Listing 4: Compiled browser plan for one scenario step.

```
- action: snapshot
  save_as: dsl_before_7

- action: click
  component: get_quote_button
  selector: '[data-testid="get-quote"]'
  expect:
    - type: state_equals
      path: quote_amount
      expected: 18900
    - type: state_delta
      path: quote_call_count
      from: dsl_before_7
      by: 1
  contract_refs:
    - contract:get_quote_button.click:effect
```

A.4 Compiler and Evaluator Excerpts

The compiler validates each contract and scenario bundle before constructing the executable suite. The browser evaluator runs each scenario in a fresh browser context, installs deterministic service routes, records declared evidence, and applies rule-based assertions without inspecting source code or using a language model to judge the trace.

Listing 5: Contract compilation entry point.

```
def compile_dsl_to_web_suite(contract_dsl, scenarios_dsl):
    compiled = compile_bundle(contract_dsl, scenarios_dsl)
    return build_web_contract(contract_dsl, \
        build_web_scenarios(scenarios_dsl, compiled))
```

Listing 6: Admissibility check for undeclared references.

```
compiled = compile_bundle(contract_dsl, scenarios_dsl)
# Parse declared state, pages, actors, services, events,
# tables, session, persistence, and components.
# Scenario references are accepted only if declared.
```

Listing 7: Fresh browser context for each scenario.

```
for scenario in self.scenarios.scenarios:
    context = browser.new_context(viewport={"width": 1280,
        "height": 900})
    self.service_calls = []
    self._install_service_routes(context)
    pages = self._open_scenario_pages(context, app_url,
        scenario)
    result = self._run_scenario(pages, scenario, logger)
    scenario_results.append(result)
    context.close()
```

Listing 8: Observation capture across evidence channels.

```
state = page.evaluate(self.contract.state_expression)
state["__browser"] = self._capture_browser_state(page)
state["__services"] = {"calls": list(self.service_calls)}
state["__tables"] = self._capture_table_state(page)
```

A.5 Contract Field Groups

Table 7 summarizes the major public contract fields. The protocol is implementation agnostic. An application may use any internal state representation or architecture provided that it exposes the declared interaction and observation surfaces.

A.6 Behavioral Relation Types

Private scenarios are compiled into browser actions and rule-based assertions. Table 8 lists representative relation types. The relation vocabulary checks behavioral conformance rather than source-code organization or visual polish.

A.7 Scenario Compilation Pattern

Each semantic scenario step is lowered to the executable pattern $\text{OBSERVE}_{pre} \rightarrow \text{ACT} \rightarrow \text{OBSERVE}_{post} \rightarrow \text{CHECK}$. Optional preconditions and waits are inserted only when declared by the contract or scenario.

Table 7: Major contract field groups used in ConformWeb. Application generation receives these fields as part of the public behavioral contract. Private scenarios instantiate and compose only declared behaviors.

Field group	Purpose	Examples of obligations
Components and selectors	Declare the interaction surface available to the evaluator.	Buttons, forms, filters, upload controls, role switches, confirmation controls.
Actions	Define admissible browser operations.	Click, fill, select, upload file, change role, navigate, reload, traverse history.
Observable state paths	Expose auditable application state.	Cart totals, booking status, row counts, validation maps, approval status, ledger entries.
Routes and browser meta-data	Specify navigation and route obligations.	Current page, expected route after action, back/forward behavior, reload persistence.
Roles and permissions	Specify which actors can perform or observe actions.	Manager-only approval, registrar override, technician dispatch, unauthorized no-op or denial effects.
Service fixtures	Make external dependencies deterministic.	Quote lookup, inventory service, eligibility checks, dispatch readiness, campaign budget readiness.
Uploads and parsing	Specify file-derived behavior.	CSV rows, asset lists, manifest updates, replacement semantics, derived counters.
Rendered projections	Link observable state to rendered evidence.	Table rows, ordered lists, cells, badges, totals, operational ledgers.
Persistence and history	Specify behavior across browser events.	Reload state, route preservation, history traversal, persistent approvals or cleared state.

Table 8: Representative relation types compiled into deterministic browser assertions. These relations are derived from public contract fields and do not introduce hidden product requirements.

Relation type	What it checks	Example failure
State equality / membership	Observable state has an expected value or belongs to an expected set.	A status remains draft after a required approval action.
State delta / append	Observable state changes by a declared amount or appends a declared object.	A cart count changes but the corresponding total does not.
Validation relation	Validation maps and errors match the declared input conditions.	An invalid field is accepted without a declared error.
Route relation	Browser route or page identity matches the expected route.	A workflow panel appears but the route remains stale.
Persistence relation	State and route obligations hold after reload or history traversal.	A reload preserves the screen but loses public workflow state.
Rendered projection relation	Rendered evidence agrees with observable state.	A table row is stale, missing, incorrectly ordered, or inconsistent with state.
Unauthorized no-op / denial	Role-gated actions produce declared no-op or denial effects.	A manager-only action mutates state for an unauthorized actor.
Service fixture relation	Declared deterministic service interactions are reflected in state and UI.	A quote or readiness response is ignored or partially applied.

Table 9: Representative compilation of a semantic scenario step into deterministic browser evaluation. The compiler lowers declared actions and relations without adding new product requirements.

Stage	Evaluator operation	Evidence recorded
Precondition	Confirm that declared route, role, state, or UI precondition holds.	Public state snapshot, browser route, rendered evidence, screenshot.
Action	Execute a declared browser action through the declared selector or operation.	Action metadata, actor/role, selector, upload or service fixture metadata.
Wait	Wait for declared asynchronous completion or observation point.	Timing status, service call log, post-action stabilization evidence.
Post-observation	Record application state after the action.	Public state, browser metadata, rendered UI evidence, service interactions.
Check	Evaluate rule-based relations over before/after observations.	Passed checks or first-failure relation, expected value, observed value.

B Benchmark Suite Statistics

ConformWeb contains six application families and four complexity tiers per family, yielding 24 benchmark instances. This appendix characterizes the frozen benchmark bundle at three levels: aggregate tier structure, instance-level contract and scenario inventory, and representative public-private scenario composition.

B.1 Contract Bundle Inventory

The frozen benchmark bundle contains the YAML artifacts used to define the benchmark instances. Table 10 summarizes the contract and scenario inventory by tier, while Table 11 reports the corresponding statistics for all 24 benchmark instances.

The tier inventory shows the intended scaling pattern. Median observable state paths rise from 28 in Tier A to 68 in Tier D, median component counts rise from 23 to 41, and median rendered table counts rise from 2.5 to 8.5. Private scenario counts also increase across later tiers. These statistics do not define difficulty by size alone, but they show that later tiers expose more places where state, route, role, and rendered evidence can diverge.

B.2 Scenario Lengths

Figure 4 shows how private scenario lengths vary across complexity tiers and application families. Longer scenarios enable multiple behavioral obligations to be composed and therefore expose failures that may occur only after substantial workflow progress.

B.3 Scenario Composition Across Families

Table 12 illustrates how disclosed workflows and private scoring compositions differ across application families and tiers. Private scenarios reuse

declared behaviors while composing them across roles, routes, services, persistence, and rendered views. The table summarizes the interaction structure without exposing concrete hidden values.

The flow cards also illustrate why ConformWeb is not a screenshot reconstruction benchmark. Many private compositions cross role, route, service, and persistence boundaries after the initial page has already rendered. The behavior being scored can therefore emerge only after substantial interaction.

Table 10: Benchmark inventory by complexity tier. Public and private columns report the scoring scenarios used in evaluation. Median statistics summarize structural properties of the corresponding benchmark instances.

Tier	Name	Public	Private	Med. paths	Med. comps.	Med. pages	Med. roles	Upload acts.	Med. tables
A	Consumer	52	67	28	23	3	2	1	2.5
B	Operational	64	80	42.5	29	4	4	4	5
C	Regulated	85	145	56	35	5	5	10	6.5
D	Release-stage	87	176	68	41	5.5	5.5	17	8.5
Overall	–	288	468	50	32.5	4.5	3.5	32	6

Table 11: Contract inventory for the 24 benchmark instances.

Family	Tier	Paths	Comps.	Pages	Roles	Svcs.	Uploads	Tables
Travel	A	33	23	3	2	1	1	2
Travel	B	41	31	4	2	2	1	3
Travel	C	57	36	6	3	4	2	6
Travel	D	62	37	6	3	4	2	7
Grocery	A	23	30	3	2	1	0	2
Grocery	B	39	46	4	2	2	0	2
Grocery	C	51	58	5	3	4	0	2
Grocery	D	56	59	5	3	4	0	3
Clinical	A	29	25	3	2	1	0	3
Clinical	B	47	34	4	4	2	1	6
Clinical	C	92	50	5	5	5	4	12
Clinical	D	97	51	5	5	5	4	13
Campus	A	27	23	3	2	1	0	3
Campus	B	44	27	4	4	0	1	7
Campus	C	64	29	5	5	1	2	10
Campus	D	84	38	6	6	3	4	11
Media	A	27	23	3	2	1	0	3
Media	B	38	22	4	4	0	1	6
Media	C	47	28	5	5	0	1	7
Media	D	74	40	5	6	2	5	9
HomeFix	A	50	18	3	2	1	0	2
HomeFix	B	50	24	4	4	1	0	4
HomeFix	C	55	34	5	5	1	1	6
HomeFix	D	54	42	6	6	2	2	8

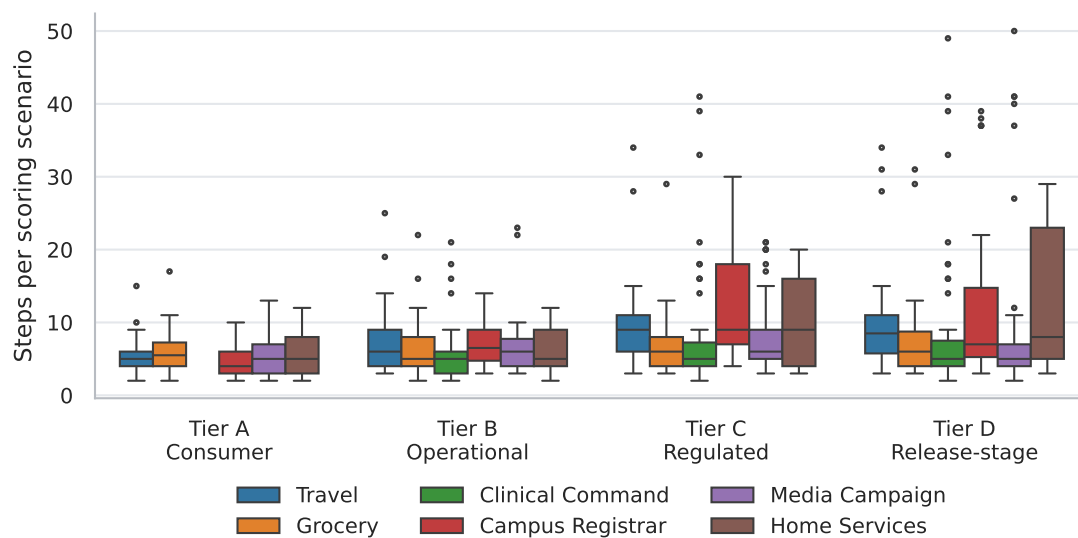


Figure 4: Distribution of scoring-scenario lengths across complexity tiers and product families. Longer scenarios provide more opportunities for late failures and long-horizon drift.

Table 12: Scenario-flow cards by product family and tier group. The table summarizes representative disclosed workflows and private scoring compositions without listing concrete hidden values.

Family	Tier	Typical public scenario flow	Typical private scoring composition
Travel	A/B	select destination, quote package, update itinerary, checkout state	add-ons, service quotes, reload checks, booking ledger consistency
Travel	C/D	supplier/conflict branches, concierge operations, reopen/release actions	role and release-stage synchronization across itinerary, quote, and ledger views
Grocery	A/B	catalog filter, cart mutation, coupon or checkout validation	substitutions, fulfillment tables, staff transitions, support ticket state
Grocery	C/D	cold-chain review, restricted items, manager approval, dispatch readiness	service-backed compliance, late dispatch/release checks, multi-view order ledger
Clinical	A/B	capacity filtering, patient selection, handoff packet, escalation state	role-gated clearance, audit-trail updates, reload and command-board consistency
Clinical	C/D	insurance/auth roles, incident events, seal or reseal gates	long command journeys combining roles, ledgers, rendered tables, and persistence
Campus	A/B	catalog filtering, schedule planning, transfer upload, advisor actions	role-gated overrides, waitlist state, enrollment and audit table projections
Campus	C/D	policy-sensitive approvals, override branches, release-stage enrollment	synchronized registrar ledgers, browser history, persistence, and final release checks
Media	A/B	asset checklist, readiness filters, budget and review tables	approval branches, service readiness, launch checklist state and projection checks
Media	C/D	budget/readiness services, campaign approval, launch or rollback	release or rollback synchronization across campaign ledgers and operational views
HomeFix	A/B	quote, booking slot, technician or roster selection, parts readiness	dispatch status, QA checks, invoice or visit tables, persistence and route checks
HomeFix	C/D	insurance, reset/reseal, warranty-command release, quality ledgers	long warranty journeys combining homeowner, pro, dispatch, QA, and ledger state

C Representative Application Surfaces

Figures 5–10 show representative surfaces for the six application families. These screenshots provide interface context for the public contracts. They are not visual targets, and generated applications are not scored by screenshot similarity.

The six families differ in vocabulary, layout, and workflow organization, but the evaluator interacts with each application through the same contract-based abstraction of actions, observations, and behavioral relations.

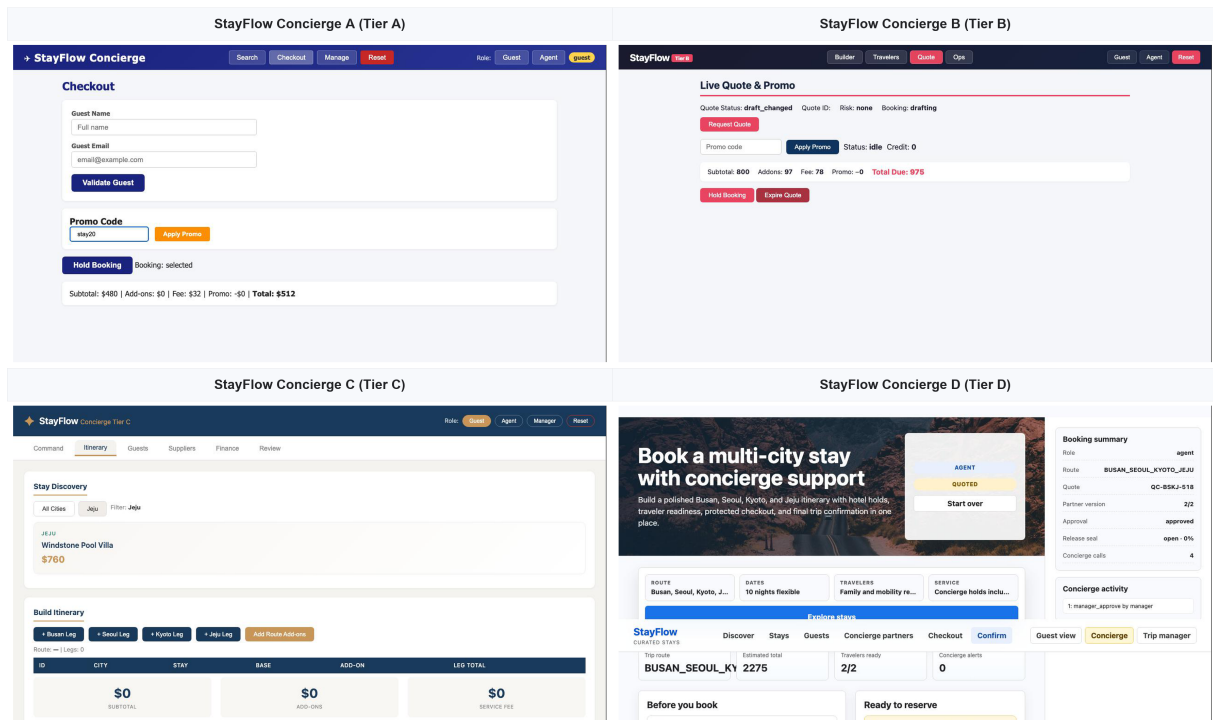


Figure 5: Representative target surfaces for Travel. The screenshots illustrate travel concierge workflows across booking, itinerary, checkout, and add-on coordination. Scoring is based on contract-derived browser interactions, not visual similarity.

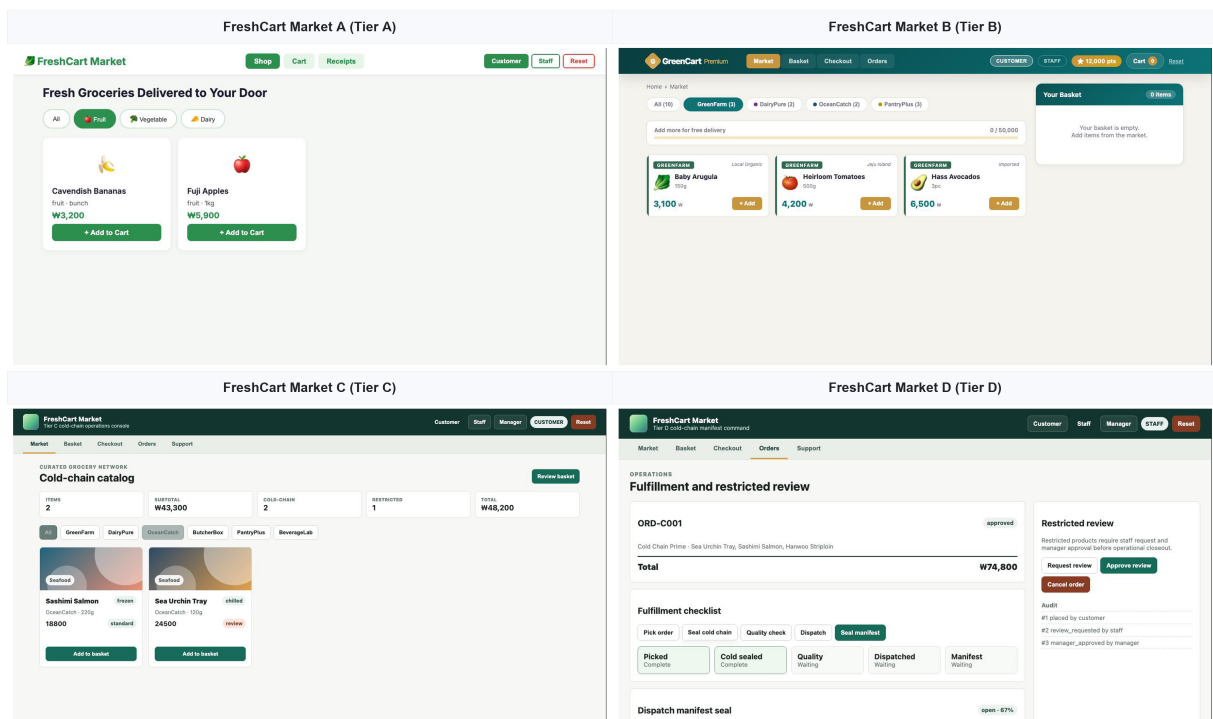


Figure 6: Representative target surfaces for Grocery. The screenshots illustrate grocery fulfillment workflows across carts, coupons, substitutions, and operational readiness. Scoring is based on contract-derived browser interactions, not visual similarity.

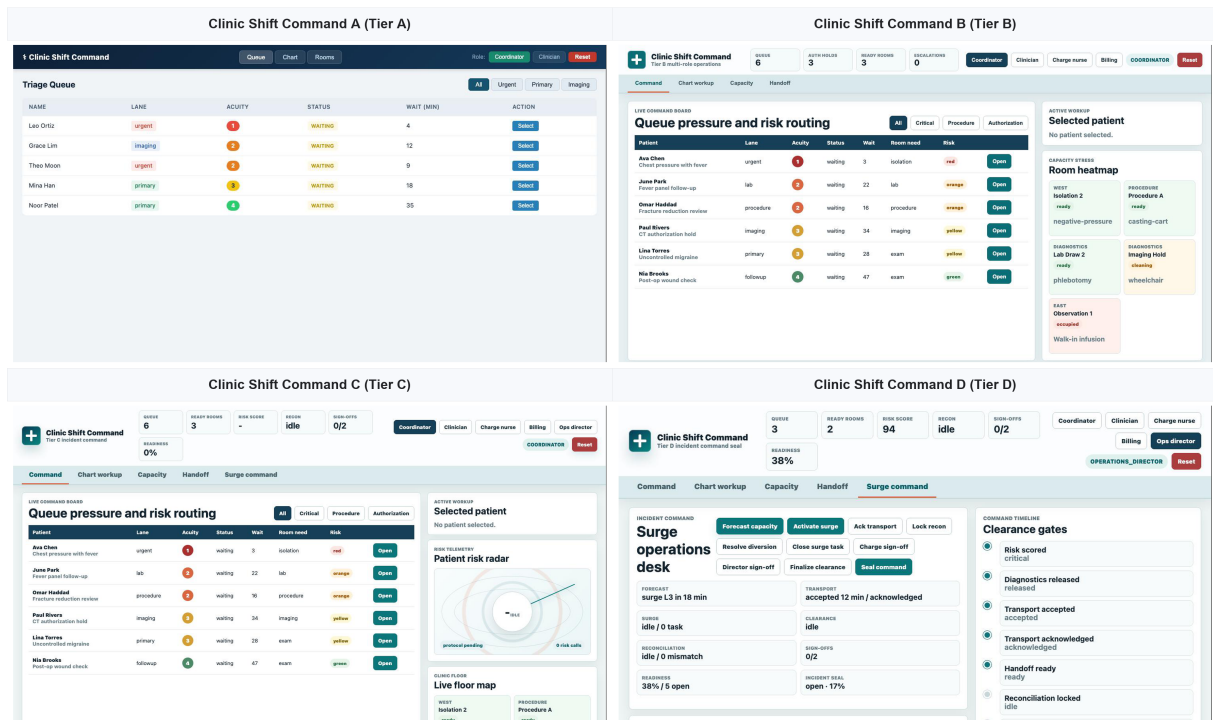


Figure 7: Representative target surfaces for Clinical. The screenshots illustrate clinical command workflows across capacity, authorization, escalation, handoff, and audit state. Scoring is based on contract-derived browser interactions, not visual similarity.

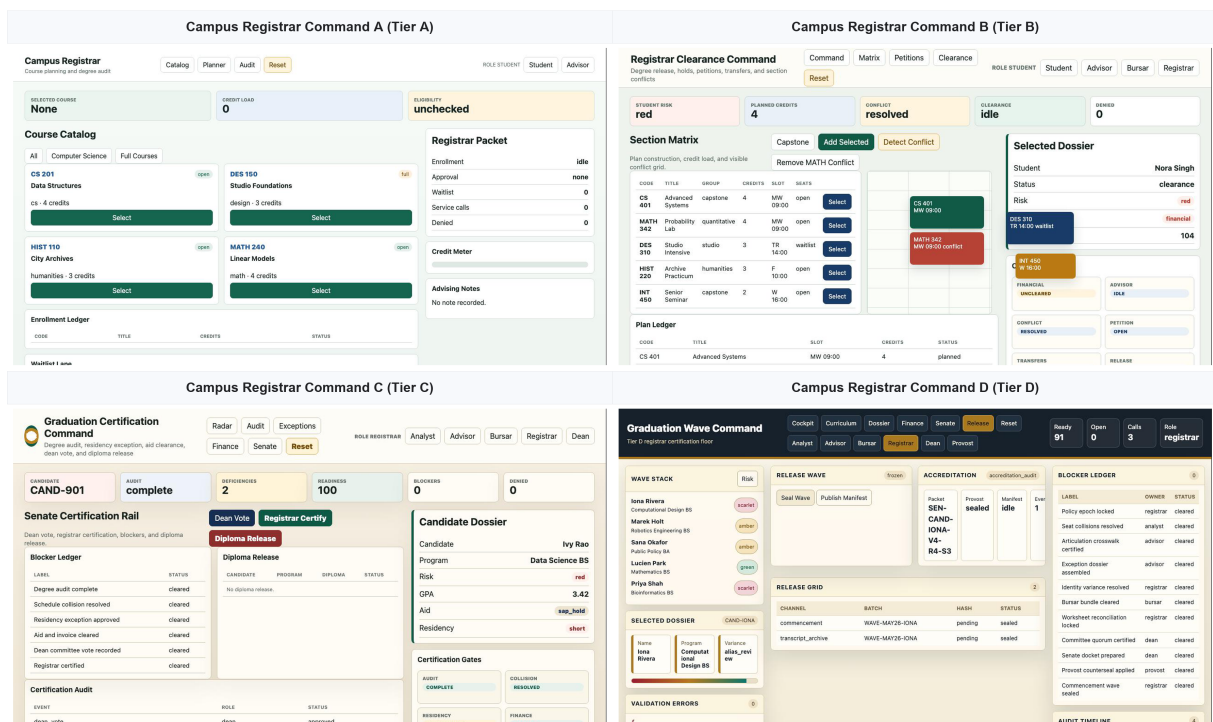


Figure 8: Representative target surfaces for Campus. The screenshots illustrate registrar workflows across enrollment, waitlists, holds, overrides, and approvals. Scoring is based on contract-derived browser interactions, not visual similarity.

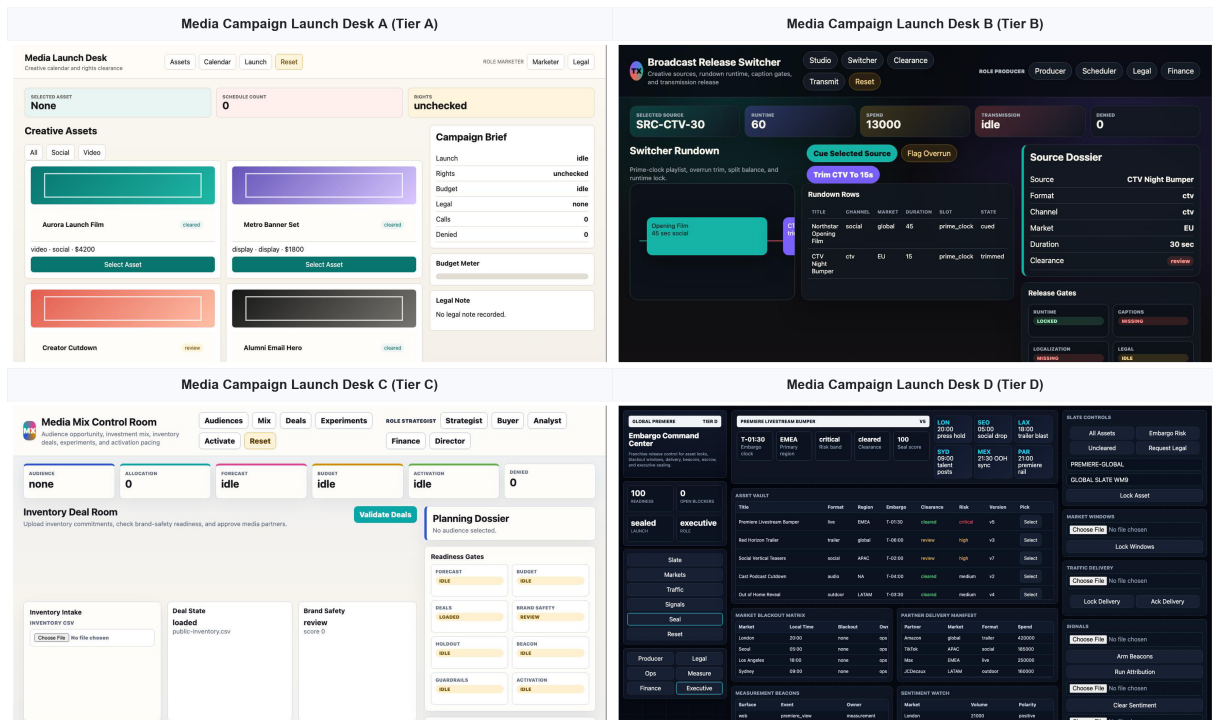


Figure 9: Representative target surfaces for Media. The screenshots illustrate campaign-launch workflows across assets, readiness checks, budgets, approvals, and release operations. Scoring is based on contract-derived browser interactions, not visual similarity.

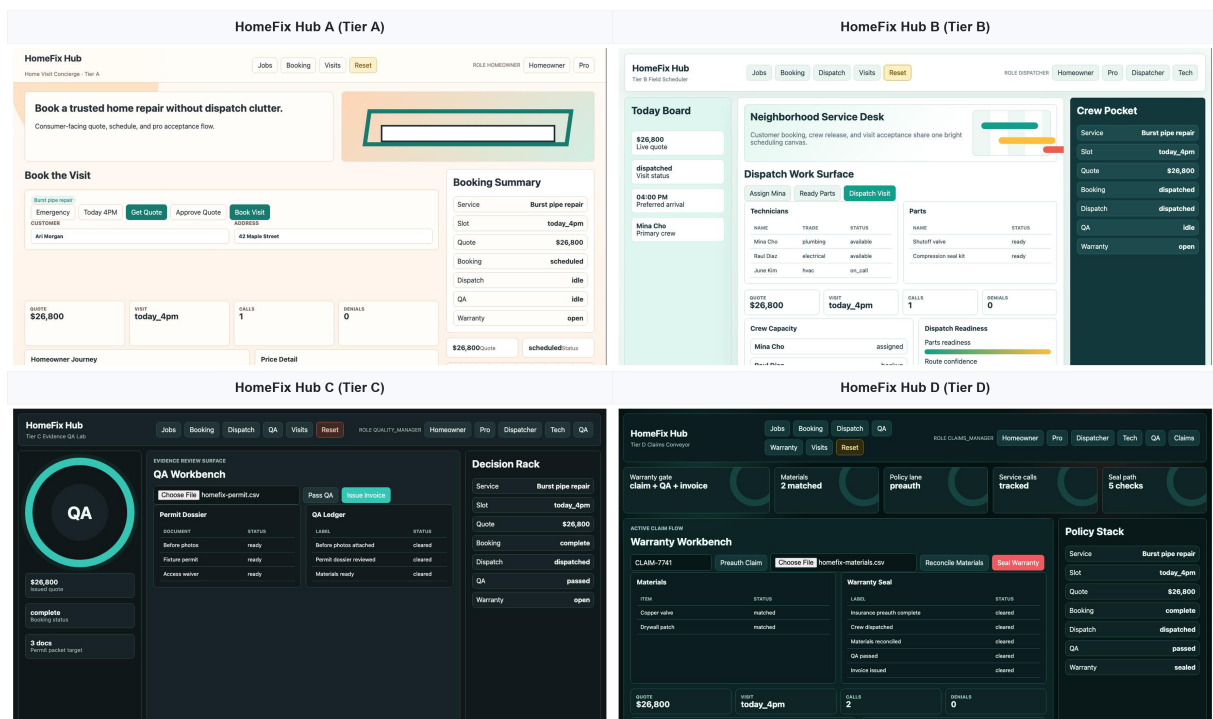


Figure 10: Representative target surfaces for HomeFix. The screenshots illustrate home-service repair workflows across booking, dispatch, parts readiness, quality assurance, insurance, and warranty completion. Scoring is based on contract-derived browser interactions, not visual similarity.

D Result Breakdowns

The main paper reports aggregate trends by model capability and complexity tier, while the results below expose variation across application families, benchmark instances, and individual generations.

D.1 Family-level Aggregate Results

Table 13 shows that the conformance gap is not confined to one application family. Even in the stronger families, step-level progress substantially exceeds strict full-pass rate.

Table 13: Family-level aggregate results. Full-pass counts use generated runs as the denominator; S_{scen} and S_{step} are mean percentages.

Family	Runs	Full	S_{scen}	S_{step}
Travel	360	6 (1.7%)	28.5	46.0
Grocery	360	20 (5.6%)	24.7	40.2
Clinical	360	43 (11.9%)	42.9	52.9
Campus	360	70 (19.4%)	42.4	52.5
Media	360	86 (24.0%)	45.3	54.2
HomeFix	360	19 (5.3%)	33.8	48.5

D.2 Family-by-tier and Instance-level Results

Figure 11 reports scenario-level conformance for each product family and complexity tier. Figure 12 further separates results by model group and benchmark instance. Together, these views show that aggregate trends are distributed across heterogeneous targets rather than being driven by a single family or instance.

D.3 Generation Variance

Figure 13 shows variation across independent generations and individual models. Repeated generations from the same public specification can yield substantially different application-level conformance.

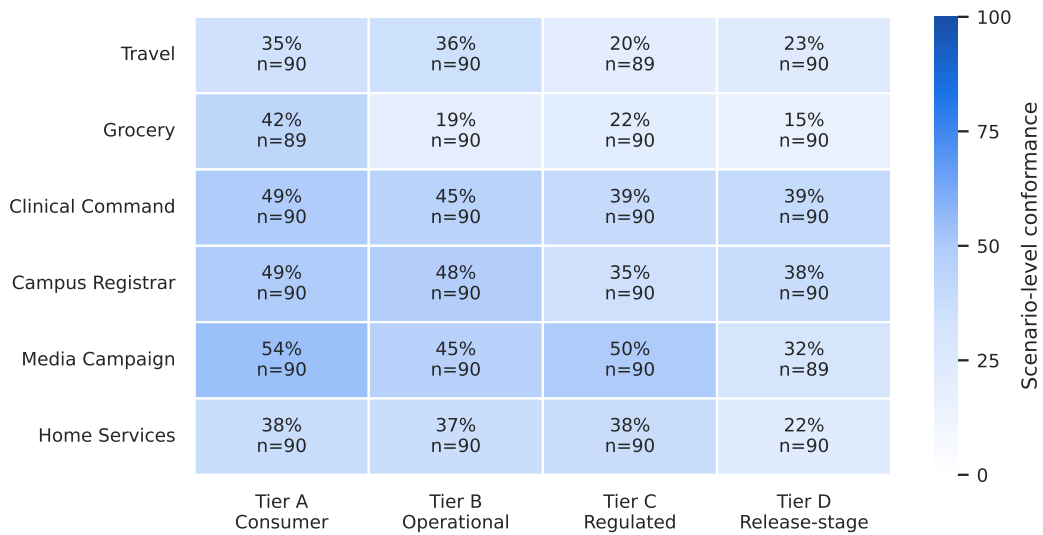


Figure 11: Family-by-tier conformance heatmap. Each cell reports scenario-level conformance for a product family and complexity variant.

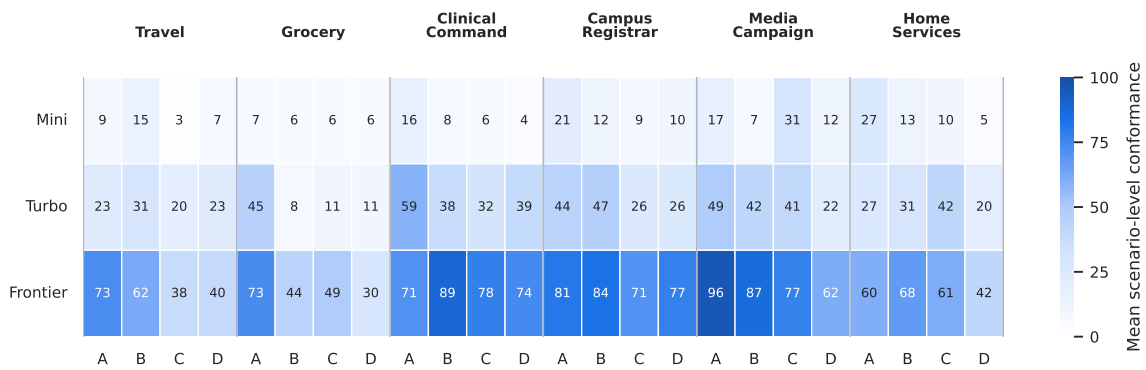


Figure 12: Model-group by benchmark-instance conformance heatmap. This view checks whether aggregate trends are driven by a small number of instances or are visible across the benchmark suite.

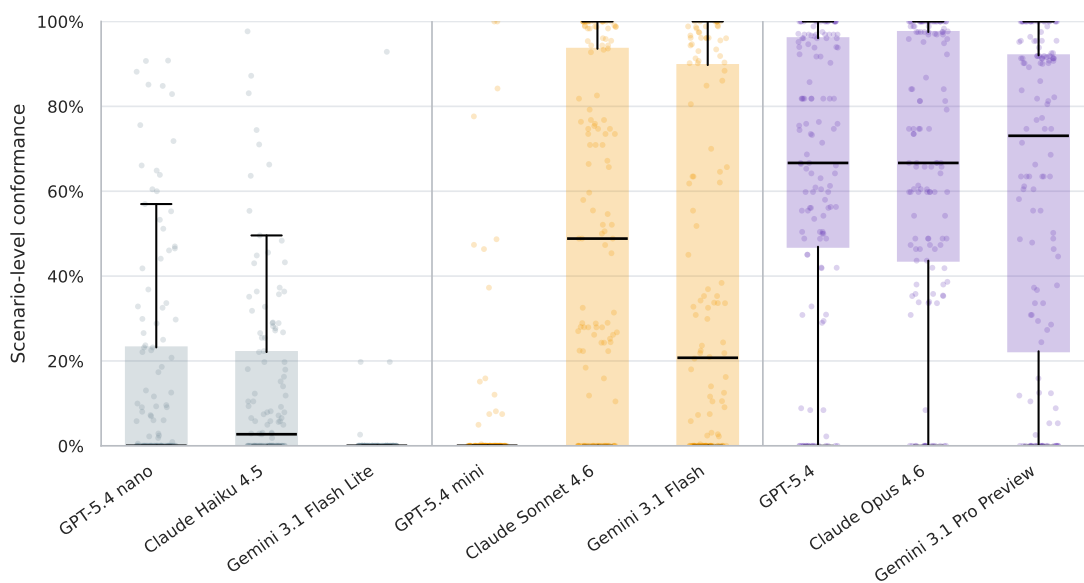


Figure 13: Generation variance across models and capability groups. Independent generations are evaluated separately under a fixed public contract.

E Failure Analysis Details

The main paper reports first-failure categories at a high level. This appendix adds relation-level breakdowns and timing views that identify what fails and where the first violation appears within the workflow.

E.1 Wrong-transition Breakdown

Wrong state transitions are a broad first-failure category in which the action is executable but the resulting public state violates an expected relation. Figure 14 splits this category into more specific subtypes.

E.2 Relation-level Failure Breakdown

Figure 15 reports which contract-derived relation types most often identify the first failing step, connecting the failure taxonomy directly to deterministic assertions.

E.3 First-failure Position

Figure 16 shows the distribution of normalized first-failure positions. Immediate failures often indicate missing surfaces or invalid observations, whereas later failures expose state, projection, route, persistence, or release synchronization drift.

E.4 Long-horizon Survival

Figure 17 complements the first-failure histogram with a survival view of normalized scenario progress. Passed scenarios survive through the end of the normalized scenario, while stronger model groups tend to preserve the contract over longer prefixes before drifting.

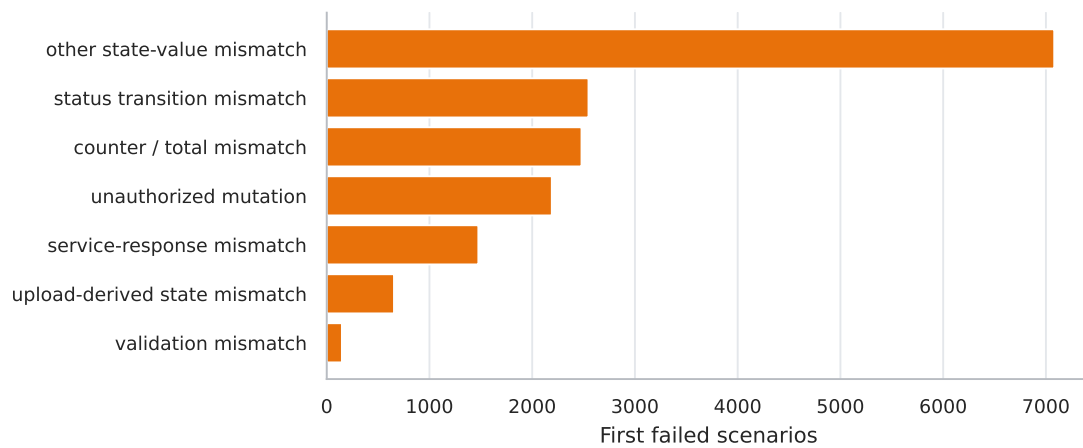


Figure 14: Breakdown of wrong-state-transition failures. The broad category covers actions that execute but produce incorrect public state, such as validation, count, status, service-response, or ledger inconsistencies.

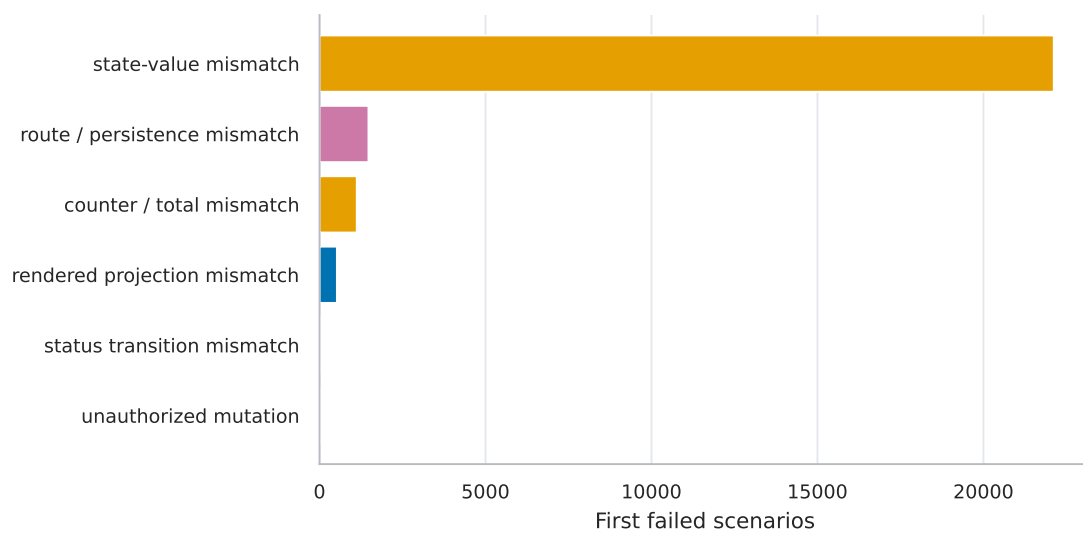


Figure 15: Relation-level breakdown of first failures. The chart reports which rule-based assertions most often identify the first contract violation.

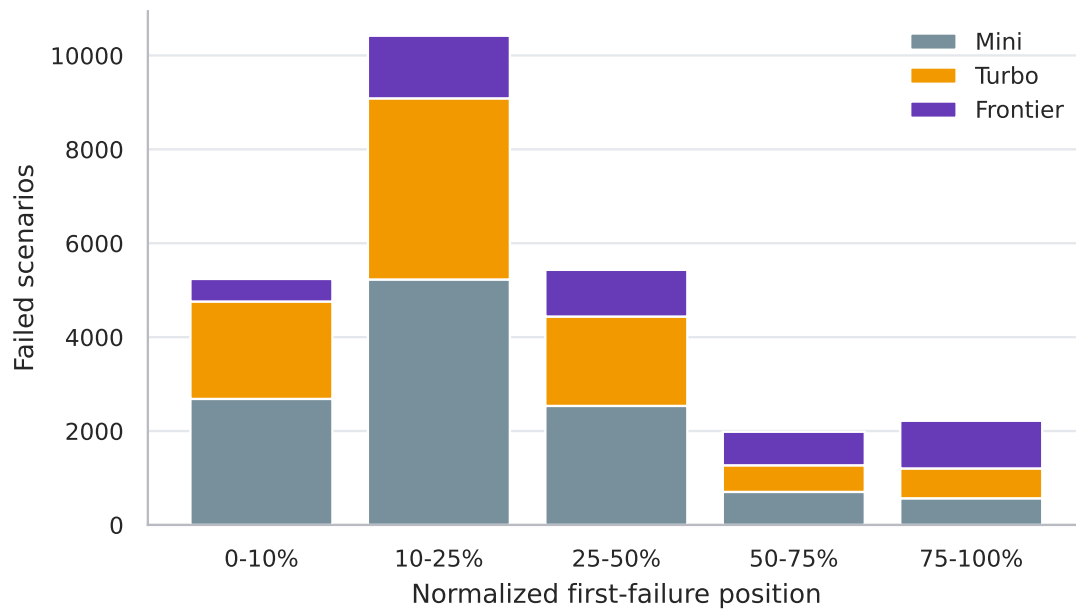
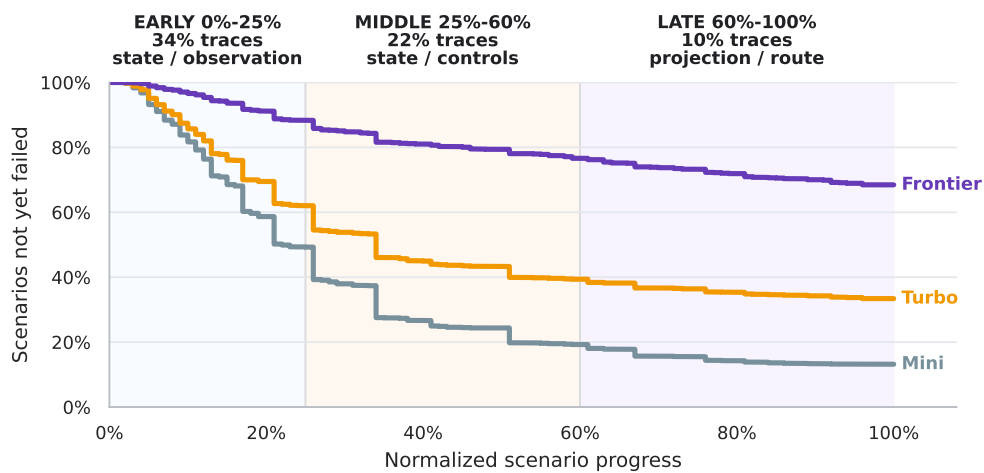


Figure 16: Distribution of normalized first-failure positions. Later first failures indicate implementations that make substantial progress before violating a contract-derived relation.



Shaded bands group where first failures occur along the normalized trace; passed scenarios survive through 100%.

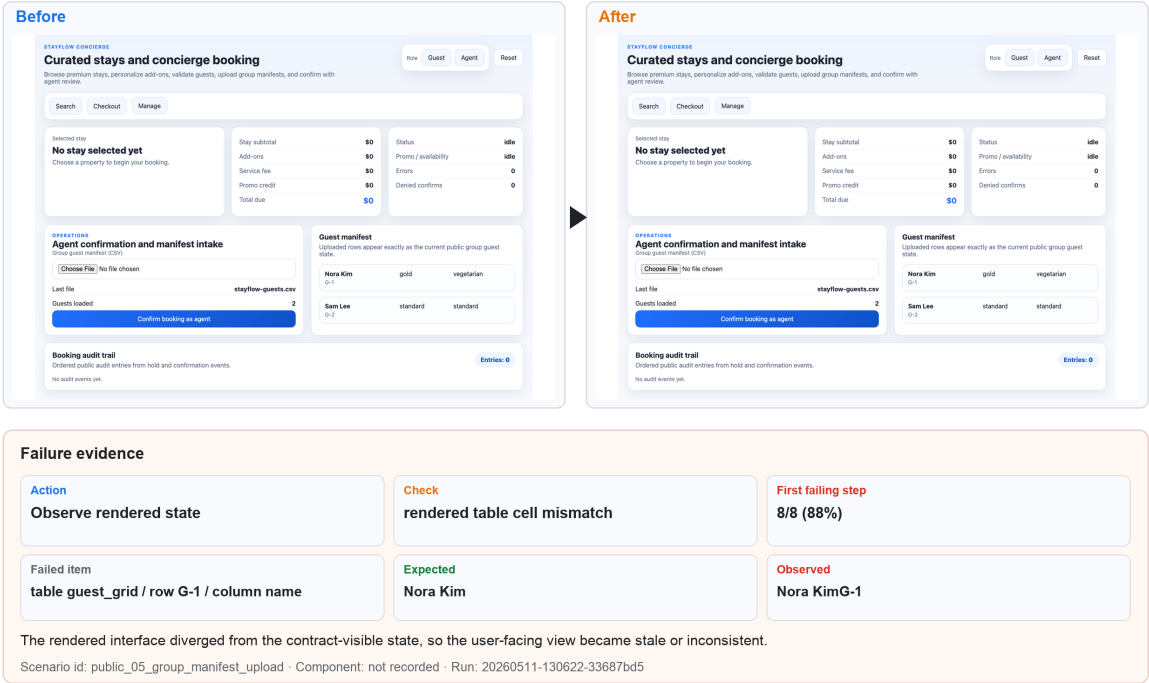
Figure 17: Long-horizon survival of generated applications. Stronger models often survive longer into held-out workflows but still fail before completing strict end-to-end behavioral conformance.

F Qualitative Failure Evidence

Figures 18–20 provide three representative first-failure panels spanning the main post-exposure failure patterns and multiple complexity tiers. Each panel shows the browser state around the first failing check together with the action, violated relation, and expected and observed values when available. These screenshots are audit artifacts collected by the deterministic evaluator, not aesthetic judgments.

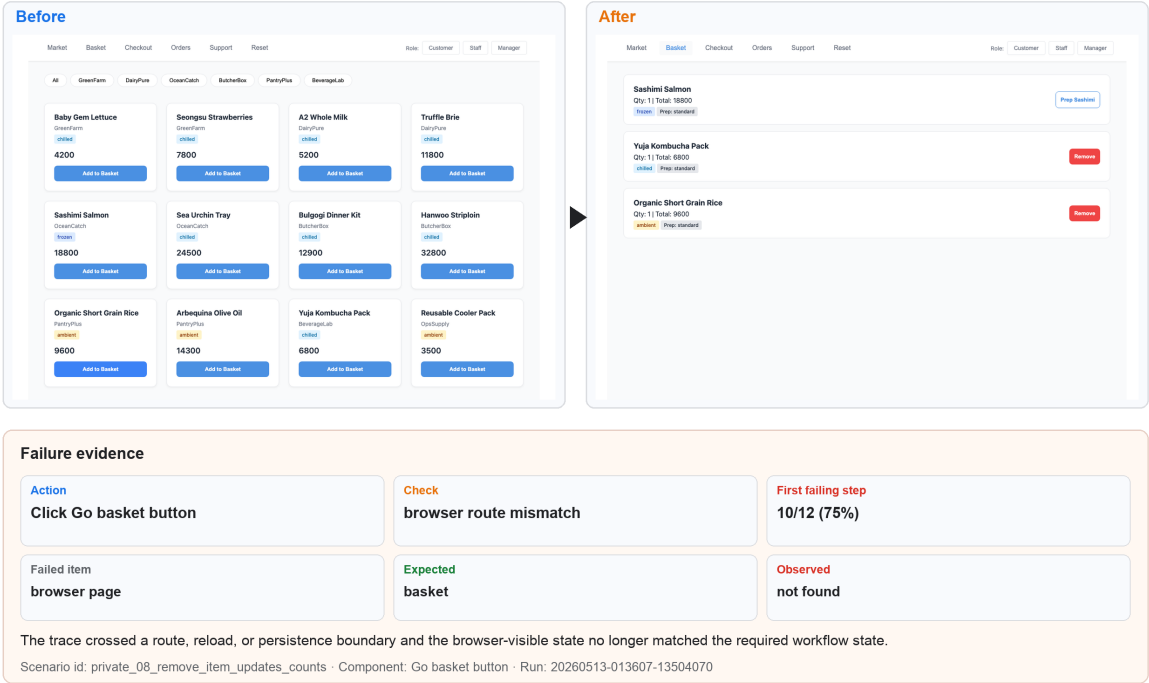
StayFlow Concierge A (Tier A): Projection mismatch

Scenario: Group manifest upload (Public) · Model: GPT-5.4 (Frontier)



FreshCart Market C (Tier C): Route / persistence drift

Scenario: Remove item updates counts (Private) · Model: Gemini 3.1 Pro Preview (Frontier)



HomeFix Hub D (Tier D): Wrong transition

Scenario: Dispatch count does not increment without parts (Private) · Model: Gemini 3.1 Pro Preview (Frontier)

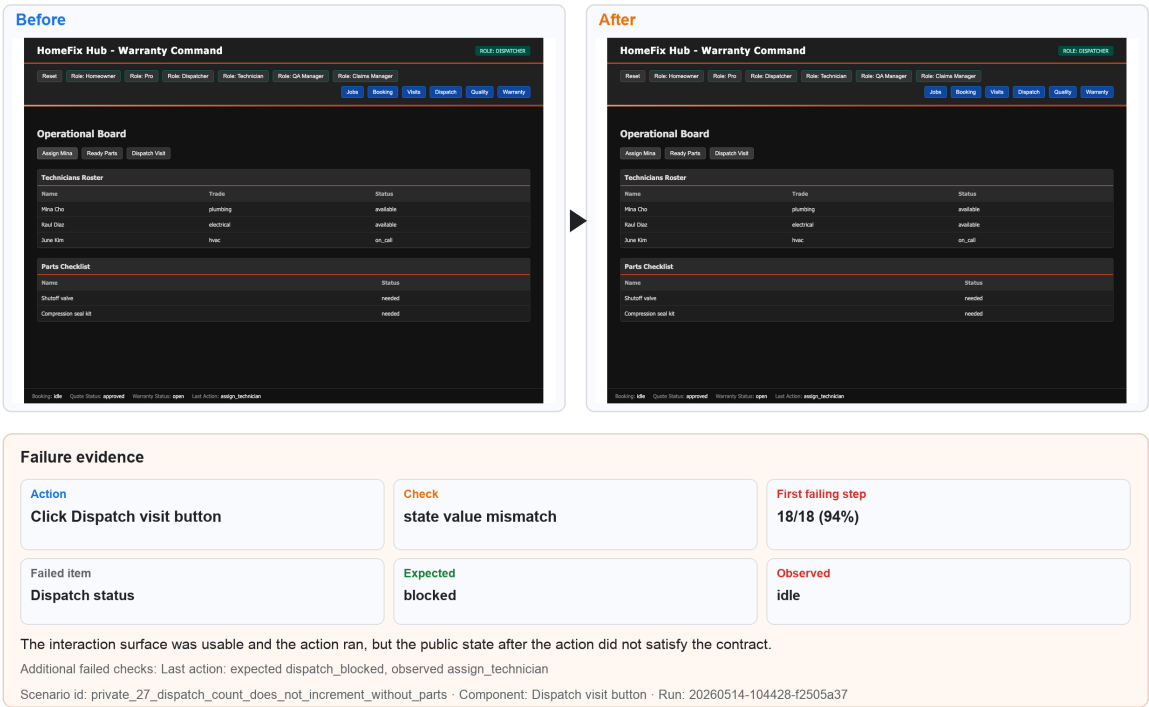


Figure 20: Representative first-failure evidence for HomeFix / Tier D: *wrong state transition*. A warranty-command action executes, but dispatch counts, parts state, or operational ledgers fail to update as required. Screenshots are audit artifacts collected by the deterministic browser evaluator, not aesthetic judgments.

G Reference Generation and Validation Loop Details

Every included benchmark instance has a reference implementation that passes the scoring scenarios before generated applications are evaluated. The reference implementation is an executable witness rather than a source-code template, and it is never shown to experimental applications. Its purpose is to verify that the public contract, private scenarios, compiler, and browser evaluator describe behavior that can actually be implemented.

For each application family and complexity tier, construction begins from the public behavioral contract, public scenarios when present, private scenarios, and a product premise. Candidate reference implementations are built from the public contract and product premise. When a candidate fails, private scenario files, full traces, screenshots, and expected trajectories are not disclosed to the builder. Limited failure summaries are used only to determine whether the problem lies in the contract, scenario specification, compiler, evaluator, or reference implementation.

This is therefore a benchmark construction and validation loop rather than a model baseline. Once an instance is frozen, experimental applications are sampled separately under the fixed disclosure boundary and receive only the public contract, product premise, and disclosed public scenarios.

Reference construction was deliberately more expensive than blind generation. Reference applications were developed with Codex using GPT-5.5 as the underlying frontier model, and one family-tier instance typically required about five to eight hours of agent-assisted iteration before passing the frozen private scenarios. This establishes attainability rather than the expected difficulty of the blind generation task.

Listing 9: Reference validation loop used during benchmark construction.

```
for instance in benchmark_instances:
    while True:
        candidate = build_reference(public_contract,
                                    product_premise)
        report = run_private_scenarios(candidate)

        if report.passed_all:
            freeze_instance(instance, candidate)
            break

        summary = limited_failure_summary(report)

        if indicates_contract_or_evaluator_defect(summary):
            revise_contract_compiler_or_evaluator(instance,
            summary)
            recompile_private_scenarios(instance)
        else:
            revise_reference_from_public_spec(candidate,
            summary)
```

H Experimental Details and Reproducibility

This appendix specifies the information boundary, generation and repair protocols, browser execution environment, and run provenance used in the experiments. The goal is to make explicit which information is available to the generating model in each setting and to distinguish changes to the generation substrate or feedback protocol from changes to the behavioral evaluation itself.

H.1 Information Boundary Across Experimental Settings

Table 14 summarizes the information exposed to the model in the three experimental settings. In all settings, the behavioral scope is defined by the same public contract and disclosed public scenarios. Private scenarios remain unavailable to the model and are introduced only during evaluation. All reported scores are computed over the union of the public and private scoring scenarios.

Table 14: Information boundary across experimental settings. The backend experiment changes the implementation substrate while preserving the behavioral specification, whereas repair adds failure information derived exclusively from disclosed public workflows.

Setting	Model-visible information	Not exposed to the model
Single-shot	Product premise, public behavioral contract, disclosed public scenarios, and the fixed DSL interpretation instructions	Private scenarios, reference implementation, evaluator traces, screenshots, failure feedback, and expected private trajectories
Backend-backed	Same behavioral inputs as single-shot, plus instructions requiring a local Express backend with persistent storage	Same private evaluation artifacts as single-shot
Public-feedback repair	Current application source, public behavioral contract, disclosed public scenarios, run identity metadata, and one structured first-failure summary obtained from public-scenario execution	Private scenarios, private execution traces, reference implementation, screenshots, and hidden expected trajectories

H.2 Single-shot Generation Protocol

All primary generations use a fixed instruction template shared across models. For each family-tier instance, the instantiated prompt contains the product premise, the complete public behavioral contract, and the disclosed public scenario examples. The template explicitly instructs the model to implement the behavior declared by the contract rather than infer an undisclosed test trajectory.

In addition to the instance-specific contract, the prompt specifies the operational interpretation of the DSL constructs that may appear in that contract. These instructions cover component selectors and actions, public state transitions, collection operations, validation, multipage navigation, asynchronous effects, role-gated actions, reload persistence, browser lifecycle events, multi-actor execution, deterministic service fixtures, rendered tables, file uploads, normalization, conflicts, and derived projections. The interpretation instructions therefore make the semantics of the public specification explicit; they do not reveal which declared behaviors will be composed in private scenarios.

The model is required to return a runnable static application consisting of exactly `index.html`, `styles.css`, and `app.js` through a structured JSON response. The generated application must expose the public state through the state-probe expression declared by the contract and retain the declared selectors as directly actionable browser elements. No private scenario, reference implementation, evaluator trace, screenshot, or failure feedback is inserted into the generation prompt.

The benchmark scripts do not explicitly override temperature, top- p , or a provider-side sampling seed. Model-specific output-token and request-

timeout budgets are controlled by the experiment runners and retained in available generation metadata. The complete single-shot prompt template is included verbatim in the released experiment artifacts.

H.3 Backend-backed Generation Protocol

The backend-backed experiment isolates the effect of the generated application substrate. The model receives the same public contract, disclosed scenarios, and behavioral instruction block used in the primary single-shot setting. The behavioral requirements are inserted verbatim from the static-client generation template; only the required application architecture is changed.

Instead of the three-file static application, the model must generate an Express application consisting of `package.json`, `server.js`, and frontend assets under `public` folder. Durable workflow state is stored by the generated backend, and user-facing state mutations are required to pass through browser fetch or XHR requests to that backend. The frontend may mirror state for rendering and for the declared state probe, but browser reload must rehydrate the state from backend storage.

Session isolation follows the browser-context boundary used by the primary evaluation. A fresh context without an application session cookie initializes a fresh copy of the contract-declared initial state, whereas a reload in the same context preserves the corresponding persisted session. Before scoring, the harness invokes a dedicated reset endpoint to clear backend storage and in-memory session state. This endpoint is used only to initialize the evaluation environment and is not part of any scored workflow.

The backend application is then evaluated by

the same contract compiler, public-plus-private scenario scope, browser actions, and rule-based assertions as the primary setting. Thus, the controlled intervention changes the persistence and execution substrate without changing the behavioral contract or evaluation semantics. The complete backend generation template is included in the released artifacts.

H.4 Public-feedback Repair Protocol

The repair experiment introduces execution feedback while preserving the private evaluation boundary. At each repair round, the model receives the current application source files, the same public behavioral contract and disclosed public scenarios used during generation, and one structured summary of the first observed failure from execution of public scenarios. The model is instructed to repair the application by generalizing from the public contract and this failure summary rather than by reproducing a particular test trajectory.

The additional repair context contains only run identity information: application family, complexity tier, instance identifier, replicate identifier, repair round, requested model identifier, local model slot, and capability band. It contains no scenario result, assertion, expected or observed value, trace, screenshot, or other evaluator-derived information.

The evaluator feedback supplied to the model has the following schema:

Listing 10: Structured public-scenario failure feedback supplied during repair.

```
{
  "scenario_id": "...",
  "scenario_tier": "...",
  "first_failed_step": ...,
  "failed_action": "...",
  "failed_component": "...",
  "failed_assertion": "...",
  "expected_value": ...,
  "observed_value": ...,
  "message": "...",
  "failure_category": "...",
  "scenario_set": "public"
}
```

Only the first failure exposed by public-scenario execution is summarized. Private scenarios are never executed to construct repair feedback. If an application already satisfies the relevant public workflow or no public failure is available to summarize, no private information is substituted for the missing feedback.

After a patch is produced, the application is evaluated again using the unchanged scoring protocol over both public and private scenarios. Repair therefore provides diagnostic information about

disclosed workflows while preserving the private workflow compositions used to measure behavioral generalization.

As a leakage check, we audited all retained patch metadata from the final public-feedback repair experiment. Every retained repair record identifies its feedback source as the public scenario set, and no private scenario identifier appears in the supplied feedback metadata. The exact repair prompt template and the corresponding prompt metadata are included in the released artifacts.

H.5 Deterministic Browser Execution

All application variants are evaluated with the same Playwright-based Chromium runner. Each scenario begins in a fresh browser context with a 1280×900 viewport. A fresh context prevents browser state from leaking between scenarios while still allowing multiple pages or actors within a single scenario when required by the public contract. Browser actions use a five-second default action timeout unless the runner explicitly specifies otherwise.

Before executing a semantic action, the evaluator records the relevant contract-declared observation state. It then performs the compiled browser operation and captures the corresponding post-action observations. These observations include the public application state, browser metadata, rendered table projections, and service-call history. Contract relations are evaluated directly over these observations using rule-based assertions.

External dependencies declared by the contract are made deterministic through runtime service fixtures. For every scenario context, the evaluator intercepts the declared service endpoints and selects fixture responses according to the request method, query parameters, and JSON body. Service calls themselves are also recorded as observable evidence. Consequently, application correctness is evaluated through browser execution and contract-grounded observations rather than source-code inspection, screenshot similarity, or an LLM-based judge.

H.6 Run Configuration and Provenance

The primary experiment follows a fixed factorial design over six application families, four complexity tiers, nine models, and ten independent generations per model–instance pair:

$$6 \times 4 \times 9 \times 10 = 2,160$$

planned primary runs. The final primary aggregate contains all 2,160 planned cells. Each generated application is evaluated against the scoring scenarios associated with its benchmark instance; public and private scenarios both contribute to the reported metrics.

Generation artifacts retain the requested model identifier and, where available, hashes of the fully instantiated generation prompt and public contract together with the producing source-code revision. Run-level records retain the benchmark instance, family, tier, generation identifier, generation or execution status, and evaluator output. These records are used to derive the model-, tier-, family-, scenario-, and step-level aggregates reported in the paper.

The released artifact contains the public contract and scenario files, fixed prompt templates, contract compiler, browser evaluator, experiment runners, and aggregation code used for the reported results. Together, these artifacts specify both the model-visible generation inputs and the deterministic evaluation procedure needed to reconstruct the experimental protocol.

I Use of AI Assistants

AI-assisted tools such as ChatGPT were used to improve the clarity, grammar, and readability of the manuscript. AI agents used during benchmark construction are described separately in Appendix [G](#).