

RStackBench: A Five-Layer Fault-Injection Method for Measuring Browser-Agent Reliability Beyond Apparent Success

Exzil Calanza

Independent Researcher, Iloilo City, Philippines

Corresponding author: Exzil Calanza (e-mail: exzilcalanza@gmail.com).

ABSTRACT Browser-agent benchmarks can credit an interaction even when the requested postcondition has not been durably achieved. This paper introduces RStackBench, a controlled Chromium fault-injection method that separates Apparent Success from Verified Task Success (VTS) across five execution layers: Access, Perception, Action, State, and Verification. The final harness spans three user-interface archetypes - configuration cards, a multi-field profile form, and a modalconfirmation workflow - and 288 deterministic task conditions. Three paired, hand-designed validation scripts yield 864 Chromium traces. Across all templates, a selector-only control achieves 18.4% VTS while displaying apparent success on 63.9% of runs, producing 131 false completions; semantic recovery achieves 66.7% VTS but produces 96 false completions; a benchmark-aware verification control reaches 100% VTS with no false completions at higher action and latency cost. A separate 240-trace scoring-horizon ablation shows that State faults resolve through passive settling (0% VTS at 50 ms and 100% by 750 ms), whereas Verification faults remain at 0% through 1000 ms without additional action. Finally, a 30-pair GPT-5.6 Sol planning pilot improves from 53.3% to 70.0% VTS under reliability-oriented prompting, but the paired difference is not statistically significant (exact McNemar $p = 0.0625$). RStackBench is therefore presented as a diagnostic measurement method, not a general browser-agent leaderboard: visible completion cues can overstate durable task completion, and postcondition-centered scoring exposes failure classes that interaction recovery alone misses.

INDEX TERMS browser agents, web agents, reliability, fault injection, benchmark, computer use, postcondition verification, LLM agents, human-computer interaction

I. INTRODUCTION

Large language model (LLM) and vision-language model agents increasingly act through web browsers. Modern benchmarks have made major progress toward realistic evaluation. WebArena created reproducible functional websites and long-horizon tasks [1]; WorkArena and the BrowserGym ecosystem expanded evaluation toward enterprise workflows and standardized action/observation interfaces [2], [3]; recent large-scale benchmarks such as WebRetriever and CAP broaden website coverage and diagnose navigation, interaction, and perception limitations [4], [5]. Across these studies, robust task completion remains difficult.

Reliability, however, is not identical to navigation or even to a visible success indication. Consider a configuration task in which an agent selects a requested value and clicks a Save button. The page can immediately display "Saved" while the backend update remains pending, fails, or requires a second

confirmation. An evaluator that credits the success banner records completion even though the durable state remains unchanged. This gap is operationally important because agents are often expected to perform state-changing work: submitting forms, changing settings, creating records, sending messages, booking resources, or updating business systems.

Prior work has directly recognized robustness gaps. WAREX augments existing web-agent benchmarks through a transparent network proxy that injects network, server, JavaScript, adversarial, and dynamic-content failures [6]. This is complementary to the problem studied here. RStackBench instead asks whether failures can be localized across an application execution pipeline and whether evaluation can distinguish apparent success from a verified postcondition. Its fault model includes target ambiguity, action delivery loss, asynchronous state persistence, and misleading completion feedback in addition to delayed access.

The central research question is:

RQ1: Does separating apparent interface success from a verified persisted postcondition reveal reliability failures that are hidden by ordinary interaction-success signals?

We also ask:

RQ2: Which of the five controlled fault layers are addressed by waiting and semantic action recovery, and which require explicit postcondition verification?

RQ3: What action and latency cost is introduced by verification-aware recovery in the controlled benchmark?

RQ4: Does the five-layer discrimination replicate across distinct user-interface interaction archetypes?

RQ5 (exploratory): Does a reliability-oriented prompt change the plans produced by a current general-purpose model when hidden execution faults may occur?

The paper makes four bounded contributions:

1. **A five-layer execution reliability model.** We define Access, Perception, Action, State, and Verification as separable fault-injection layers for browser task execution.
2. **A postcondition-centered metric.** Verified Task Success (VTS) is determined from controlled persisted state, while Apparent Success and False Completion remain separate observables.
3. **A reproducible fault-injection harness.** RStackBench runs on localhost in an isolated Chromium profile, produces deterministic task definitions and run records, and supports graded fault severity without mutating third-party sites.
4. **A controlled evaluation with replication and sensitivity analysis.** We report 864 paired Chromium validation-script traces across three interface archetypes, 240 additional scoring-horizon traces, and a separately labeled 30-pair GPT-5.6 Sol planning pilot. The hand-designed scripts validate benchmark discrimination; they are not presented as learned-agent state of the art.

The principal claim is deliberately narrower than "verification solves browser agents." The experiments demonstrate that, under known postconditions in controlled applications, postcondition-centered scoring exposes fault classes that are invisible to policies that stop at a success UI. The discrimination replicates across three interface archetypes, but generalization to uncontrolled websites, broad task semantics, and autonomous agents remains future work.

II. RELATED WORK

A. A. Functional and Realistic Web-Agent Benchmarks

WebArena introduced a realistic and reproducible environment spanning e-commerce, forums, collaborative development, and content management, together with tasks evaluated for functional correctness [1]. Its reported GPT-4-based baseline substantially trailed human performance, establishing the difficulty of realistic browser task completion. WorkArena shifted attention toward common enterprise knowledge-work tasks using ServiceNow and introduced

BrowserGym as a rich environment for multimodal browser-agent actions and observations [2]. The BrowserGym ecosystem subsequently unified multiple web-agent benchmarks and experiment tooling, emphasizing the need for comparable, reproducible evaluation [3].

Recent work has increased scale and diagnostic resolution. WebRetriever reports 1,550 tasks over 800 websites and explicitly argues that navigation success alone is not a sufficient predictor of real-world application effectiveness [4]. CAP constructs 420 cross-site tasks spanning 108 websites and 24 domains, with emphasis on complex UI operations and visual perception [5]. These benchmarks motivate fine-grained diagnosis but do not eliminate the need to distinguish interface feedback from durable postconditions.

B. B. Reliability and Fault Injection

WAREX is the most directly related reliability framework identified in this study [6]. It operates as a transparent proxy between the agent and benchmark, injecting network delays, HTTP errors, JavaScript failures, popups, and other dynamic variations while preserving compatibility with existing benchmarks. It demonstrates significant task-success degradation under injected failures.

RStackBench is designed to be complementary rather than competitive in scope. WAREX primarily perturbs transport, delivery, server-response, JavaScript, and dynamic-content conditions through a transparent proxy. RStackBench operates one layer higher in the application/task stack: DOM target identity, action delivery at the application control, persistence timing, and postcondition verification. Its *State* and *Verification* faults specifically cover cases in which browser interaction and visible feedback can appear normal while the requested persisted postcondition is absent. The controlled server allows the evaluator to compare UI feedback against application ground truth directly. A combined design could therefore compose WAREX transport perturbations with RStackBench application-state scoring rather than treating the methods as substitutes.

C. C. Evaluation Signals and Completion

End-to-end task success is valuable because it measures whether an agent achieved the requested result. In practice, however, evaluators use heterogeneous signals, including URL matching, DOM or storage values, textual answers, screenshots, rule-based state checks, and LLM judges. WebRetriever's use of richer interaction context and CAP's fine-grained operation decomposition reflect broader recognition that a single navigation or visual signal is insufficient.

RStackBench contributes a measurement distinction rather than a replacement judge: **Apparent Success** describes user-visible completion feedback, while **Verified Task Success** describes the controlled durable postcondition. This distinction enables measurement of False Completion, where the former is present and the latter is absent.

III. RELIABILITY MODEL

D. A. Task and State Model

A benchmark task is represented as a requested transformation from initial persistent state s_0 to a target postcondition g . The browser agent observes rendered state o_t , issues actions a_t , and eventually terminates or is stopped by a scoring boundary. For task i , Verified Task Success is

$$VTS_i = 1[S_i(t_{score}) = g_i],$$

where $S_i(t_{score})$ is the server-side persisted state at the benchmark's scoring boundary and g_i is the requested target value.

Apparent Success is

$$AS_i = 1[U_i(t) \text{ contains a benchmark-defined success indication for some } t \leq t_{score}],$$

where U_i denotes the rendered user-visible interface.

False Completion is

$$FC_i = AS_i \text{ AND NOT } VTS_i.$$

The distinction is important: AS_i measures what the interface told the agent/user, while VTS_i measures what the controlled application actually persisted.

E. B. Five Reliability Layers

1) Access. The intended interactive surface is not yet available when the agent begins. RStackBench implements delayed control availability (70, 180, or 420 ms by severity). The relevant capability is bounded waiting or recovery from transient absence.

2) Perception. Multiple visually plausible controls share labels or structures, making target identity ambiguous. RStackBench inserts one to three decoy project cards before the requested card. Each decoy contains a Mode control and a

Save change button. The relevant capability is semantic target binding rather than first-match selection.

3) Action. The intended action is correctly targeted but not delivered. RStackBench swallows the first one to three target Save clicks. The relevant capability is observing the lack of progress and retrying within a bound.

4) State (temporal settling). The page emits optimistic success while persistence is delayed. RStackBench displays success immediately but commits the requested state after 120, 320, or 700 ms. No additional user action is required; passive time is sufficient. This layer measures premature termination relative to a declared scoring horizon rather than a permanent backend failure.

5) Verification (active postcondition gap). The page emits a success indication although the requested postcondition has not been committed and requires additional confirmation actions (two or three total target Save clicks by severity). Passive waiting alone cannot satisfy the goal. The relevant capability is checking the requested postcondition and repairing a mismatch rather than trusting the success banner. The distinction is therefore causal rather than merely visual: State faults resolve through settling time, whereas Verification faults require an additional action. Section V.D measures this distinction directly across multiple scoring horizons.

Fig. 1 summarizes the five-layer execution sequence and the capability tested at each boundary.

These layers are not claimed to be a complete ontology of browser failure. Security attacks, authentication challenges, cross-site dependencies, nondeterministic third-party content, rate limits, and model reasoning errors can be layered on separately. The five-layer model is a deliberately testable decomposition of execution reliability.

RStackBench five-layer execution reliability model



FIGURE 1. Fig. 1. RStackBench five-layer execution reliability model.

IV. RSTACKBENCH DESIGN

F. A. Controlled Web Applications and Interface Archetypes

RStackBench uses a Node.js localhost HTTP server and isolated headless Chromium profiles controlled through the Chrome DevTools Protocol. The server owns authoritative state for each run, while rendered pages expose user-visible state and completion cues. Fault injection is specified by deterministic task configuration rather than uncontrolled external failure, enabling exact replay and paired comparison. The final study uses three interaction archetypes:

1. **Configuration-card workflow.** A target project card contains a Mode selector, a Save change button, a visible status element, and a summary of the current mode. Perception faults insert visually plausible decoy cards.
2. **Multi-field profile form.** A profile editor combines text/select input with a submission control and persisted profile summary, allowing the same reliability layers to be tested on form filling rather than card-local selection.
3. **Modalconfirmation workflow.** A destructive record action opens a confirmation modal before deletion, testing target selection and persistence through a different control structure and interaction sequence.

No third-party website, account, or external data is mutated. The benchmark therefore measures execution behavior without operational risk to external services.

G. B. Task Generation

The original configuration-card experiment contains 144 task conditions: 24 clean baseline tasks and 24 tasks for each of the five fault layers. The two additional interface archetypes contribute 72 conditions each: 12 baseline tasks and 12 tasks for each fault layer. The final cross-template validation set therefore contains **288 unique task conditions** distributed across three interface archetypes.

Fault-layer tasks cycle through three severity levels. Target identifiers, initial values, and requested postconditions are generated deterministically from seeded choices. Exact task definitions are serialized before execution. Each condition is executed by all three hand-designed validation scripts, yielding 864 cross-template Chromium traces.

H. C. Reference Policies

Three hand-designed reference scripts are executed on every task as harness validation controls. They are designed to isolate the capabilities targeted by each injected layer and are not empirical baselines for autonomous agents.

Selector-only control. This open-loop script attempts to select the target value when immediately available, clicks the first visible Save change button, and performs no bounded recovery. It represents a brittle interaction strategy rather than a modern state-of-the-art agent.

Semantic-recovery control. This script waits for the requested project card, targets that card specifically, and retries

when visible success does not occur. It is intended to isolate the benefit of target binding and action recovery.

Verification-aware control. This script adds a second, server-backed user-visible postcondition check. If the summary does not reflect the requested state, it performs bounded repair and re-verification. This policy is benchmark-aware by design. Its purpose is to validate whether the benchmark's State and Verification layers discriminate postcondition checking, not to establish an upper bound for general web agents.

I. D. Experimental Protocol

All three harness-validation scripts execute the same 288 task conditions across the three interface archetypes, producing **864 Chromium execution traces**. For each run RStackBench records fault layer and severity, Apparent Success, Verified Task Success, False Completion, action count, retry count, elapsed task latency, and bounded diagnostics.

The primary control experiments use a fixed short scoring boundary after the script stops interacting. Because this can confound temporal settling with permanent failure, a separate passive-settling sensitivity analysis re-executes all 24 original State tasks and all 24 original Verification tasks at five horizons (50, 150, 350, 750, and 1000 ms), producing **240 additional Chromium traces**.

A separately labeled model-planning pilot samples 30 unique original-template tasks (five from each of baseline plus the five fault layers) and obtains one ordinary and one reliability-oriented plan for each task, yielding 60 model plans and 60 interpreted Chromium executions.

J. E. Statistical Analysis

For each validation script we report VTS rate, Apparent Success rate, False Completion count, mean actions, and mean latency. VTS proportions are accompanied by Wilson 95% confidence intervals in the machine-readable summary. Because all validation scripts are evaluated on identical task conditions, pairwise success differences are tested with the exact two-sided McNemar test using discordant pairs.

The tests characterize deterministic differences among these particular harness-validation scripts on this controlled benchmark; the resulting p-values are secondary implementation checks rather than estimates of population-level agent performance. They do not imply population-wide effect sizes for arbitrary web tasks.

V. CONTROLLED EXPERIMENT RESULTS

K. A. Overall Cross-Template Results

TABLE I

Cross-Template Validation-Script Results

Validation script	Runs	Verified Task Success	Apparent Success	False Completions	Mean Actions	Mean Latency (ms)
Selector-only	288	18.4%	63.9%	131	1.73	250
Semantic-recovery	288	66.7%	100.0%	96	2.50	289
Verification-aware	288	100.0%	100.0%	0	2.83	640

Fig. 2 visualizes the aggregate gap between interface feedback and persisted completion. Across three interface archetypes, the selector-only control shows the largest gap between visible completion and persisted completion: apparent success occurs on 184 of 288 runs while only 53 reach the requested persisted postcondition. Semantic recovery reaches 192 of 288 VTS outcomes, but all 96 State/Verification conditions remain false completions. The benchmark-aware verification control

reaches all 288 targets and serves as a positive harness control rather than a production-agent baseline.

The reliability cost remains measurable. Relative to semantic recovery, verification-aware execution increases mean actions from 2.50 to 2.83 and mean host-side latency from 289 to 640 ms. These values are implementation- and host-specific and are reported descriptively.

Apparent success versus verified persisted completion (288 conditions)

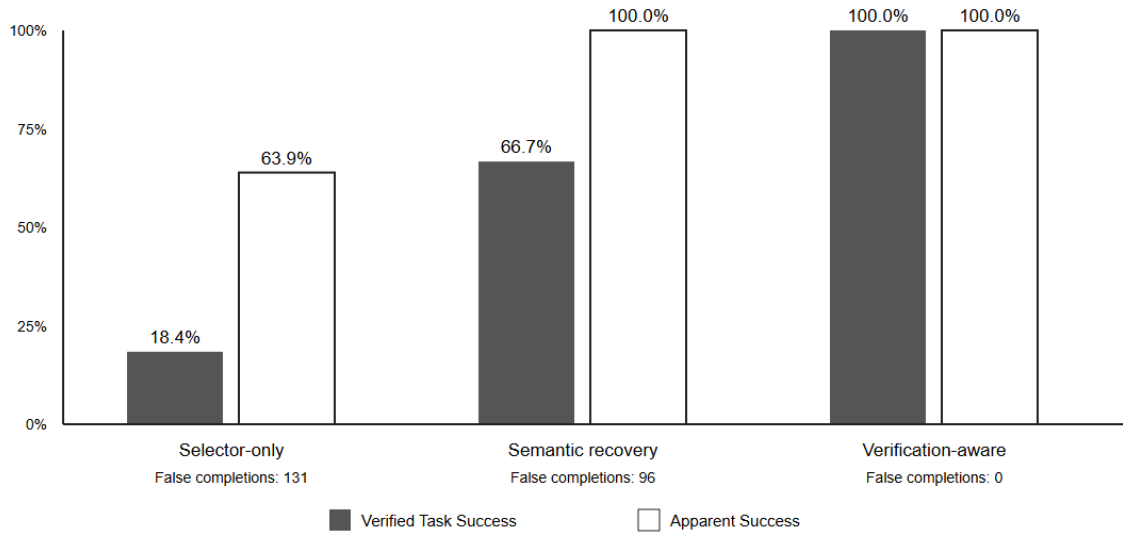


FIGURE 2. Fig. 2. Aggregate Apparent Success and Verified Task Success across 288 task conditions. False Completion is Apparent Success without the requested persisted postcondition.

L. B. Layer-Specific Results

TABLE II

Verified Task Success by Reliability Layer

Validation script	Baseline	Access	Perception	Action	State	Verification
Selector-only	100.0%	8.3%	0.0%	0.0%	2.1%	0.0%
Semantic-recovery	100.0%	100.0%	100.0%	100.0%	0.0%	0.0%
Verification-aware	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Fig. 3 shows the corresponding layer-by-layer VTS matrix. Each layer contains 48 conditions across the three templates. Semantic recovery fully resolves the injected Access, Perception, and Action conditions but not State or

Verification. The single selector-only State success (1/48) reflects a timing coincidence at the fixed scoring boundary and reinforces why State results require an explicit horizon rather than a timeless success label.

Verified Task Success by reliability layer across three UI archetypes

	Baseline	Access	Perception	Action	State	Verification
Selector-only	100.0%	8.3%	0.0%	0.0%	2.1%	0.0%
Semantic recovery	100.0%	100.0%	100.0%	100.0%	0.0%	0.0%
Verification-aware	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

FIGURE 3. Fig. 3. Layer-specific Verified Task Success across the three interface archetypes.

The same qualitative discrimination reproduces across the three interface archetypes:

M. C. Cross-Template Replication and Paired Controls

TABLE III

Cross-Template Replication

Interface archetype	Selector-only VTS	Semantic-recovery VTS	Verification-aware VTS
Configuration cards (144 conditions)	18.1%	66.7%	100.0%
Multi-field profile form (72 conditions)	19.4%	66.7%	100.0%
Modal confirmation (72 conditions)	18.1%	66.7%	100.0%

Fig. 4 compares the replicated pattern across the three interface archetypes. This replication does not establish open-web external validity, but it reduces the risk that the five-layer discrimination is an artifact of one dropdown/card DOM structure. It covers selection, form submission, and modal-confirmation workflows under the same persisted-ground-truth measurement contract.

Exact paired McNemar checks over the 288 conditions give 140 discordant wins versus one reverse discordance for

semantic recovery over selector-only, 235 wins versus zero reverse discordances for verification-aware over selector-only, and 96 wins versus zero reverse discordances for verification-aware over semantic recovery. The corresponding exact two-sided p-values are extremely small, but because these are deterministic, hand-designed controls over a constructed benchmark, the tests are treated only as consistency checks and not as population-level evidence about autonomous agents.

Fault discrimination replicates across three interface archetypes

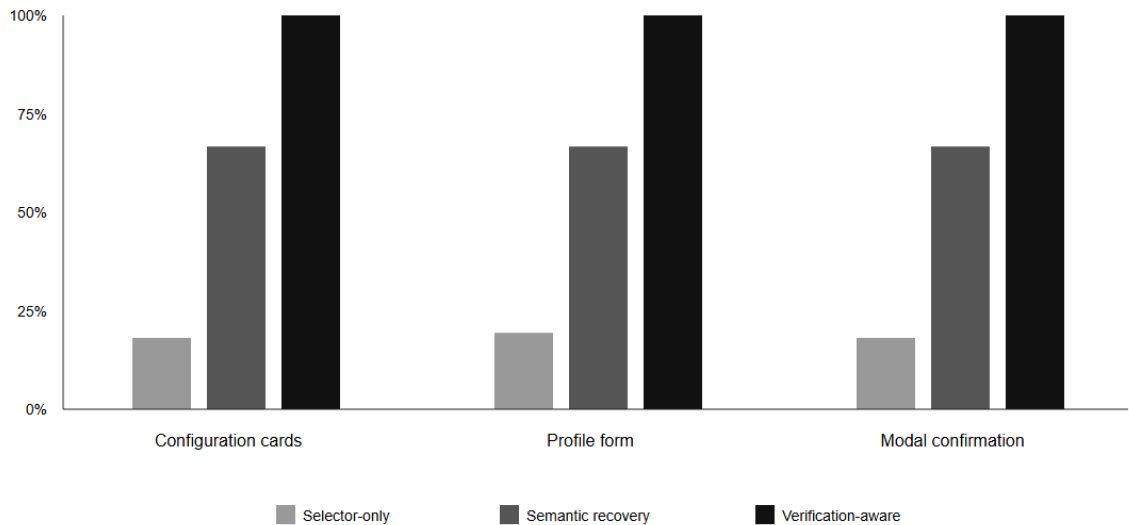


FIGURE 4. Fig. 4. Cross-template replication of the validation-script discrimination pattern.

N. D. Scoring-Horizon Sensitivity: State Versus Verification

To test whether the State layer merely encodes an arbitrary timeout, we re-ran all 24 State tasks and all 24 Verification

tasks with a single target action followed by passive waiting before server-side scoring. Five prespecified horizons were tested: 50, 150, 350, 750, and 1000 ms. This produces 240 additional Chromium traces.

TABLE IV

Scoring-Horizon Sensitivity		
Passive scoring horizon	State VTS	Verification VTS
50 ms	0/24 (0.0%)	0/24 (0.0%)
150 ms	7/24 (29.2%)	0/24 (0.0%)
350 ms	15/24 (62.5%)	0/24 (0.0%)
750 ms	24/24 (100.0%)	0/24 (0.0%)
1000 ms	24/24 (100.0%)	0/24 (0.0%)

Fig. 5 shows the full horizon response. The State curve follows the injected 120/320/700 ms persistence delays and reaches complete VTS by 750 ms without any additional action. Verification remains at 0% through 1000 ms because its hidden contract requires additional confirmation actions. This ablation resolves an important construct-validity ambiguity: a State-layer failure at a short scoring horizon is a premature-

completion event, whereas a Verification-layer failure is an unresolved postcondition that time alone does not repair. Consequently, State results must always be reported together with the scoring horizon; Verification results can be interpreted as requiring additional interaction under the present benchmark contract.

Passive settling resolves State faults but not Verification faults

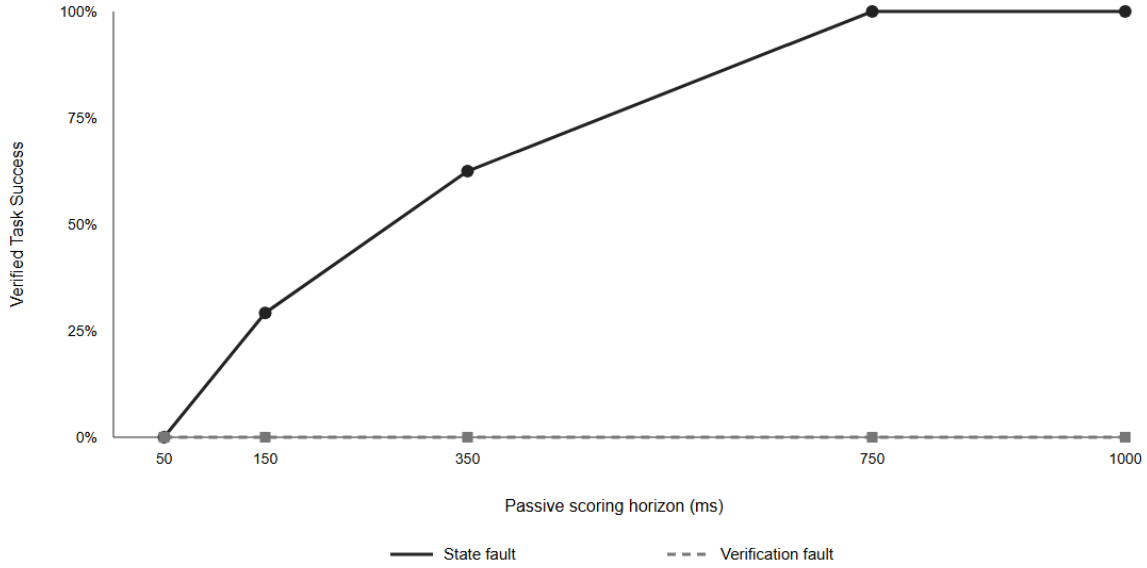


FIGURE 5. Fig. 5. Passive scoring-horizon sensitivity. State faults resolve through time alone; Verification faults do not.

VI. EXPLORATORY GPT-5.6 SOL PLANNING PILOT

GPT-5.6 Sol is a publicly documented OpenAI model and is available through Codex and the OpenAI API [7]. To test whether reliability framing affects plans without exposing the hidden fault label, we conducted a paired planning pilot using the locally authenticated Codex CLI reporting model `gpt-5.6-sol`.

Thirty unique original-template task observations were sampled: five clean baseline tasks and five tasks from each of the five fault layers. Each observation was presented under two prompt conditions:

1. **Ordinary execution prompt:** plan a concise normal execution and do not add defensive steps unless the visible page makes them necessary.

2. **Reliability-oriented prompt:** assume transient loading, ambiguous controls, swallowed actions, delayed persistence, or misleading success UI may occur; verify the durable user-visible outcome and include bounded recovery.

The model received only the natural-language task and visible time-zero observation, not the hidden fault label or private implementation. Plans were constrained to a JSON action schema and executed by a deterministic Chromium interpreter. Harness-development attempts that failed before producing a valid plan were excluded from outcomes and preserved separately in logs.

TABLE V

Exploratory GPT-5.6 Sol Planning Pilot

Prompt condition	Paired tasks	VTS	Apparent Success	False Completions	Mean Executed Steps	Mean CLI Planning Time
Ordinary	30	16/30 (53.3%)	25/30	9	3.30	11.31 s
Reliability-oriented	30	21/30 (70.0%)	26/30	5	8.57	22.89 s

Wilson 95% intervals are 36.1%-69.8% for the ordinary condition and 52.1%-83.3% for the reliability-oriented condition. In the paired comparison, five tasks improve under the reliability prompt and none improve in the reverse direction. The exact two-sided McNemar p-value is 0.0625.

The result is therefore **not statistically significant at the conventional 0.05 threshold** and is reported as exploratory evidence only.

The layer pattern is informative but underpowered. Both prompts reach 5/5 on Baseline, Access, and Perception.

Ordinary prompting reaches 0/5 on Action, 1/5 on State, and 0/5 on Verification; reliability prompting reaches 1/5 on Action, 5/5 on State, and 0/5 on Verification. Reliability prompting increases mean executed steps by 160% and approximately doubles planning time. CLI logs report mean total token usage of 9,706 tokens per ordinary plan and 10,185 per reliability-oriented plan (about 4.9% higher), although these totals are tool-reported usage rather than a controlled API-cost measurement.

The pilot supports only a bounded observation: explicit reliability instructions changed plan structure and reduced false completions in this small sample, but did not produce statistically conclusive VTS improvement and did not solve the benchmark's active Verification faults. It is not a provider ranking or evidence of general browser-agent performance.

VII. DISCUSSION

O. A. Apparent Success Is Not a Sufficient Completion Criterion

The key methodological observation is not the 100% score of the benchmark-aware verification control. That policy is explicitly designed around known benchmark postconditions and therefore has a strong structural advantage. The more general result is the gap between Apparent Success and VTS. Both selector-only and semantic-recovery can display high apparent success while failing to achieve the requested persistent state.

This matters for benchmark construction. If a task's goal is state-changing, an evaluator should prefer a postcondition tied to the requested state when such a signal is available. A success toast, navigation event, click receipt, HTTP request initiation, or agent self-report can be useful evidence, but none is automatically equivalent to completion.

P. B. Recovery and Verification Address Different Failures

The five-layer decomposition separates two engineering problems that are often conflated.

Recovery handles transient inability to perform the intended action: wait for access, disambiguate the target, and retry a lost action. This is sufficient for the injected Access, Perception, and Action layers.

Verification asks whether the requested outcome exists after apparently successful interaction. The State and Verification layers demonstrate cases where the action path can look normal while the postcondition is absent at the relevant time. In such cases, more aggressive clicking without an outcome check is not a principled reliability strategy.

Q. C. Reliability Has Cost

Verification-aware execution increases both elapsed time and action count. In LLM-driven systems it can also increase model calls, input/output tokens, and context usage; the present controlled reference scripts do not measure monetary API cost. Production agents therefore require a policy for

when and how strongly to verify. High-impact mutations may justify expensive postcondition checks; low-risk read-only navigation may not. The benchmark can be extended to study adaptive verification policies that estimate the value of verification relative to latency, action, or token cost.

R. D. Relationship to WAREX and Large-Scale Benchmarks

RStackBench should be combined with, not substituted for, broader benchmarks. CAP and WebRetriever provide scale and real-site diversity [4], [5]. BrowserGym provides standardized agent experimentation [3]. WAREX adds delivery-layer and adversarial perturbations to existing benchmark environments [6]. RStackBench contributes a controlled execution-layer decomposition and a persisted-ground-truth completion metric.

A promising future experiment is to integrate the two approaches: apply network/server/JavaScript fault injection through WAREX while simultaneously scoring RStackBench-style postconditions. Such a design could determine whether apparent-success errors increase under environmental instability and whether adaptive verification remains effective.

VIII. THREATS TO VALIDITY

S. A. Construct Validity

VTS is strong when the evaluator controls the application and knows the authoritative target state. It is not universally available on the open web. Some tasks have subjective or informational goals for which a server-side equality predicate is inappropriate. The contribution is therefore the distinction between interface evidence and goal evidence, not a claim that every browser task can use the exact VTS implementation.

The five layers are also not exhaustive. Authentication, authorization, captchas, cross-site coordination, prompt injection, policy compliance, rate limiting, browser crashes, and reasoning failures are outside the present fault set.

T. B. Internal Validity

The validation scripts are hand-designed and benchmark-aware. In particular, the verification-aware policy knows how to inspect the benchmark's server-backed summary and retry. Its 100% result validates the controlled mechanism but cannot establish a real-world ceiling.

Faults are injected independently and deterministically. Real incidents may combine layers, such as delayed access followed by stale perception and a partial write. Compound fault experiments are future work.

Timing thresholds are part of the benchmark contract. Different thresholds would change State-layer outcomes. All task definitions and delays are serialized to allow independent sensitivity analysis.

U. C. External Validity

The study now spans three interface archetypes (configuration cards, a multi-field form, and a modal-confirmation workflow), but these remain synthetic localhost applications rather than a representative sample of the Web. The experiments run on one Windows host and one installed Chromium build. Results should not be extrapolated to mobile agents, native desktop agents, or arbitrary production websites. The exploratory LLM pilot uses a single public model family [7], 30 paired tasks, one constrained planner schema, and one deterministic plan interpreter. It is not a model comparison and should not be used to rank providers.

V. D. Statistical Conclusion Validity

Exact McNemar tests are appropriate for paired binary outcomes in the controlled comparison, but the task conditions are generated by one benchmark family. Very small p-values confirm consistent paired differences on this constructed set; they do not measure ecological importance. Reported latency means are descriptive and host-specific.

IX. REPRODUCIBILITY AND ARTIFACT AVAILABILITY

The research package contains the deterministic original and multi-template task generators, Chromium benchmark runners, exact serialized task definitions, all 864 cross-template validation-script traces, the 240-trace horizon-sensitivity dataset, aggregate summaries, independent reanalysis scripts, the constrained LLM planning schema, all 60 final paired model plans and execution records, figures, and SHA-256 manifests. The package also preserves failed harness-development attempts separately so they cannot be confused with evaluated outcomes.

Independent reanalysis verifies the key invariants and regenerates aggregate rates from raw JSONL rather than trusting the manuscript tables. Artifact hashes are frozen into a reproducibility manifest. IEEE Access emphasizes sufficient methodological detail and supports reproducibility artifacts [8]-[10].

Before journal submission, the local package should be archived in a public, versioned repository or research archive with a stable identifier. The manuscript should cite that frozen public version rather than the working-directory path. Public release is deliberately not performed automatically as part of this study package.

X. ETHICAL, PUBLICATION, AND AI-USE DISCLOSURE

The benchmark operates exclusively against localhost test services and isolated temporary Chromium profiles. It does not perform experimental mutations on third-party websites, accounts, or user data.

A public engineering commentary titled "Why Browser Agents Fail: A Real-World Reliability Stack" was published on exzilcalanza.info before this research manuscript. That commentary motivated the five-layer framing but did not contain the RStackBench method, the 288-condition three-template benchmark, the 864-trace control dataset, the 240-

trace horizon ablation, or the 30-pair model pilot reported here. The relationship should be disclosed in a cover letter to avoid ambiguity about prior public material. IEEE Access permits preprints and prior public versions under its stated conditions, but the author remains responsible for similarity and originality compliance [9].

In accordance with the current IEEE Access AI-generated-content policy [9], OpenAI GPT-5.6 Sol [7] was used for experimental scripting and debugging (Sections III-V and the reproducibility artifact), structured plan generation for the explicitly labeled pilot (Section VI), statistical aggregation/reanalysis code (Sections V-VI), and drafting/editing language throughout the manuscript. Google Gemini, labeled "3.6 Thinking" in the provider interface during this work, was used after the initial draft as an adversarial critique tool; its recommendations were independently checked against raw artifacts before incorporation. All executable experiments were run on the author's controlled system, machine-generated outputs were checked against stored artifacts, primary references were independently verified, and the author remains solely responsible for the research design, interpretation, manuscript, and submission.

XI. CONCLUSION

RStackBench demonstrates a specific measurement failure in browser-agent evaluation: an interface can indicate success while the requested postcondition is absent. Across **864 paired validation-script Chromium traces spanning 288 conditions and three interface archetypes**, apparent-success rates substantially overstate persisted completion for controls that do not verify outcomes. A separate **240-trace scoring-horizon ablation** distinguishes State faults that resolve through passive settling from Verification faults that require additional action. A **30-pair GPT-5.6 Sol pilot** shows a directionally higher VTS rate under reliability prompting (53.3% to 70.0%) but does not cross the conventional significance threshold ($p = 0.0625$), and therefore remains exploratory.

The paper does not claim that explicit verification makes general browser agents reliable or that hand-designed controls predict production-agent rankings. It argues for a narrower evaluation principle: when a browser task changes state, benchmark credit should be tied as closely as possible to the requested postcondition, while interface feedback is treated as evidence rather than proof. The five-layer fault model, VTS/False-Completion distinction, cross-template replication, and frozen artifacts provide a reproducible method for studying that principle alongside larger-scale and network-level robustness benchmarks.

REFERENCES

[1] S. Zhou et al., "WebArena: A Realistic Web Environment for Building Autonomous Agents," arXiv:2307.13854, 2023.

- [2] A. Drouin et al., "WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks?" arXiv:2403.07718, 2024.
- [3] T. Le Sellier De Chezelles et al., "The BrowserGym Ecosystem for Web Agent Research," arXiv:2412.05467, 2024.
- [4] W. Dong et al., "WebRetriever: A Large-Scale Comprehensive Benchmark for Efficient Web Agent Evaluation," arXiv:2607.06118, 2026.
- [5] Z. Xu et al., "CAP: A Scalable Benchmark for Evaluating Cross-Site Browser Agents with Complex Actions and Perception," arXiv:2608.08392, 2026.
- [6] S. Kara, F. Faisal, and S. Nath, "WAREX: Web Agent Reliability Evaluation on Existing Benchmarks," arXiv:2510.03285, 2025.
- [7] OpenAI, "GPT-5.6 Sol Model," OpenAI API Documentation, accessed Aug. 14, 2026.
- [8] IEEE Access, "Reproducibility Pilot Program," accessed Aug. 14, 2026.
- [9] IEEE Access, "Preparing Your Article," accessed Aug. 14, 2026.
- [10] IEEE Access, "Submission Guidelines for Authors," accessed Aug. 14, 2026.

AUTHOR BIOGRAPHY

EXZIL CALANZA is an independent researcher and software builder based in Iloilo City, Philippines. His academic background is in information systems at Iloilo Science and Technology University. His technical background includes technical support, web development, workflow automation, Python/Django, PHP/WordPress, SQL, REST APIs, and AI-assisted reliability tooling. His current interests include browser-agent reliability, agent evaluation, workflow verification, and human-computer interaction.