

When the DOM Lies: Separating UI Settlement, Transport Completion, and Semantic Success in Autonomous Web Agents

Exzil Calanza

Independent research preprint. This work is not peer reviewed.

Source commit: [ecd71496f10ec95841123e956c5433e81dc310dd](https://github.com/exzilcalanza/when-the-dom-lies/commit/e71496f10ec95841123e956c5433e81dc310dd)

The reported counts characterize only the frozen experiments and production sample described here; they are not universal reliability estimates.

Abstract

Autonomous browser agents frequently need to decide whether an action has actually completed. A visible Document Object Model (DOM) change is tempting as a completion signal, but modern web applications can mutate optimistically before a request finishes, can receive an HTTP-successful response whose application semantics represent failure, and can continue committing application state after transport completion. This study separates three evidence layers: UI/DOM settlement, correlated transport completion, and action-specific semantic success. We first measured a preregistered deterministic fixture in Chromium and independently replicated it in Firefox. In each 320-run controlled matrix, optimistic DOM settlement accepted every constructed failure, while correlated transport removed transport-layer failures but still accepted constructed semantic HTTP-200 failures. A fixture-specific committed-state oracle separated all registered controlled cases. We then tested a frozen 50-trial production sample on an owner-controlled public site using independently frozen direct-HTTP truth labels. In that sample, correlated transport eliminated false negatives and rejected all registered HTTP-404 cases, but a generic stabilized-rendered-state rule still accepted 9 of 10 semantic HTTP-200 negative trials. The production result limits the controlled finding: transport completion is stronger evidence than optimistic DOM settlement, but neither transport success nor generic page stabilization establishes semantic action success. Reliable autonomous verification therefore requires explicit action-to-request correlation plus an action-specific semantic success contract. The reported counts characterize the registered samples only and are not universal reliability estimates.

1. Introduction

A browser agent must often answer a deceptively simple question: Is the action finished? The answer is difficult because web applications expose several clocks that need not agree. The DOM can change immediately after a click. Response headers can arrive later. The response body can finish later still. Application code can then validate the payload, update state, render a final view, or reject the operation while retaining HTTP status 200.

Treating those events as interchangeable creates a verification error. A visible UI mutation proves that a UI mutation occurred; it does not prove that the target request completed. A completed HTTP request proves transport completion; it does not prove that the application accepted the operation. A stable page proves only that the observed rendering has stabilized under the chosen stability predicate. It does not, by itself, define business or task success.

This distinction matters for autonomous agents because a false-positive completion decision can cause the next action to execute against an invalid state. The failure may be subtle: optimistic success text may appear before a request fails, an unrelated background request may finish inside the same action window, or a semantically negative search response may return HTTP 200 and settle into a perfectly stable page.

The study asks a narrower question than “which browser wait is best?” It asks what each observable signal can legitimately establish, and where a verification architecture must introduce action-specific semantics.

2. Evidence model

We distinguish four evidence layers.

UI/DOM observation. A target element, title, or text state becomes visible. This is direct evidence of rendered state at an observation time, but it may be optimistic or intermediate.

Action-to-request correlation. Network evidence is tied to the intended action and target rather than to an arbitrary request occurring nearby in time. Correlation is necessary because modern pages generate analytics, images, prefetches, background API calls, service-worker traffic, and other concurrent activity.

Terminal transport state. The correlated target request reaches a terminal success or failure event. Chrome DevTools Protocol exposes request/response and loading-completion/failure events; WebDriver BiDi similarly distinguishes response start from response completion. These event families make it possible to distinguish “headers exist” from “the response completed,” but terminal transport success remains a transport claim.

Semantic success contract. The application-specific condition that makes the user’s intended action true. Examples include a newly created object appearing with a stable identifier, a save operation returning a version that is subsequently readable, a search returning at least one result satisfying the query contract, or a workflow entering a named terminal state. This layer cannot be inferred universally from HTTP 2xx or generic DOM stability.

The central operational rule is therefore:

Completion evidence should be no stronger than the strongest independently established layer. DOM settlement establishes DOM settlement; correlated transport establishes correlated transport; semantic success requires a semantic contract.

3. Related standards and empirical background

The Chrome DevTools Protocol Network domain provides events including `Network.requestWillBeSent`, `Network.responseReceived`, `Network.loadingFinished`, and `Network.loadingFailed`. The distinction between response availability and loading completion permits instrumentation of the gap between headers and terminal transport state [1].

The W3C WebDriver BiDi specification provides a standards-track network model with events including `network.beforeRequestSent`, `network.responseStarted`, `network.responseCompleted`, and `network.fetchError`. In particular, response start and response completion are distinct events, supporting a protocol-neutral formulation of the transport-settlement problem even though implementations must still be measured rather than assumed identical [2].

Empirical work on flaky JavaScript and UI tests provides broader motivation. Hashemi, Tahir, and Rasheed report asynchronous waits, race conditions, deadlocks, and network stability among important flakiness causes in JavaScript systems [3]. Romano et al. analyze flaky UI-test samples across web and Android projects and motivate detection and prevention approaches specific to UI-test behavior [4]. Those studies do not establish the settlement policy tested here; they support the premise that asynchronous UI and network behavior is a meaningful reliability problem.

4. Research questions

RQ1. How often does optimistic DOM settlement disagree with registered outcome truth in controlled asynchronous scenarios?

RQ2. What failures are removed by requiring terminal transport evidence for the correlated target request?

RQ3. Does a committed-state rule that works in a controlled fixture generalize to a small production sample when the semantic outcome is independently labeled?

RQ4. What additional latency accompanies stronger settlement evidence in the registered controlled experiments and production sample?

5. Method

5.1 Controlled Chromium matrix

A deterministic local fixture exposed separately timestamped action dispatch, optimistic DOM mutation, response lifecycle, application commit, and observer settlement. The preregistered expanded experiment used Chromium 145.0.7632.6, ten scenario families, two target-delay levels (50 ms and 250 ms), sixteen repetitions per scenario-delay cell, and 320 registered rows. Scenarios included success, HTTP failure, semantic HTTP-200 failure, truncated/aborted response behavior, redirects, streaming, cache/service-worker paths, and concurrent target/background traffic.

The registered policies were:

1. optimistic DOM settlement; 2. correlated target network completion; 3. correlated target transport plus optimistic state; 4. correlated target transport plus fixture-defined committed application state; and 5. an uncorrelated successful-request negative control.

An excluded smoke run exposed a redirect-correlation observability defect. The correlation fallback was corrected before the registered matrix, and a second excluded smoke passed all instrumentation gates. The full matrix then ran from zero.

5.2 Controlled Firefox replication

External-validity Arm A froze its protocol before Firefox outcome rows were observed. The confirmatory matrix used Playwright Firefox 146.0.1 with engine-neutral Playwright request, response, request-finished, and request-failed observations; no Chromium DevTools Protocol session was created. The same ten scenario families, two delay levels, sixteen repetitions per cell, and 320 registered rows were used.

The first excluded smoke attempt produced no outcome rows because the required Playwright Firefox binary was absent. After installation, a fresh excluded smoke passed all ten hard instrumentation checks. The confirmatory run then proceeded without post-outcome protocol modification.

5.3 Production Arm B

Production Arm B used a frozen, non-destructive sample from the owner-controlled public site exzilcalanza.info. The manifest contained ten cases with five repetitions each: five successful public navigations, three deterministic HTTP-404 failures, and two semantic HTTP-200 negative search cases, for fifty final trials.

Outcome labels were generated independently through direct-HTTP evidence and SHA-256-frozen before browser-policy telemetry was collected. The labels could not inspect the candidate policy's predictions.

The production measurement implementation required two preserved amendments. Version 1 revealed a transient Loading ...?_aiwp_epoch=... shell at domcontentloaded, making the original committed-state observation too early. Amendment v1.1, frozen before rerun, added title-shell settlement. Version 1.1 then revealed that final title settlement could precede asynchronous body commitment. Amendment v1.2, again frozen before rerun, added a bounded generic body-stability predicate: three equal normalized text samples 200 ms apart, with a five-second bound. The production manifest and independent truth labels were unchanged. Failed v1 and v1.1 measurement runs remain retained as evidence rather than being silently removed.

6. Results

6.1 Chromium controlled matrix

Policy	TP	FP	TN	FN
Optimistic DOM	192	128	0	0
Correlated target network	192	64	64	0
Correlated target + optimistic state	192	64	64	0
Correlated target + committed fixture state	192	0	128	0
Uncorrelated successful request	192	64	64	0

Mean correlated target-network terminal latency was 122.998 ms. Mean transport-plus-final-state settlement latency was 166.926 ms.

In the constructed fixture, optimistic DOM settlement accepted all controlled failures. Correlated terminal transport removed the transport-failure half but still accepted semantic HTTP-200 failures. The fixture-specific committed-state oracle separated all registered rows. The uncorrelated negative control accepted 64 controlled failures, demonstrating that a successful request in the action window is not adequate causal evidence.

6.2 Firefox controlled replication

Policy	TP	FP	TN	FN
Optimistic DOM	192	128	0	0
Correlated target network	192	64	64	0
Correlated target + optimistic state	192	64	64	0
Correlated target + committed fixture state	192	0	128	0
Uncorrelated successful request	192	64	64	0

The controlled-failure false-positive rate for the committed-state rule was 0/128, with a Wilson 95% interval of approximately 0.000–0.029 for this registered controlled sample. The corresponding 0/192 controlled-success false-negative count has an upper Wilson bound of approximately 0.0196. These are sample intervals, not production-prevalence estimates.

The median correlated network-terminal latency was 60.307 ms and the median dual-final latency was 107.500 ms. The committed-state rule was correct on 128 paired rows where optimistic DOM was wrong and on 64 paired rows where correlated network-only was wrong, with no reverse disagreements in the constructed fixture. Those discordances show controlled separation; they do not prove production universality.

6.3 Production Arm B

Policy	TP	FP	TN	FN
Optimistic DOM at domcontentloaded	14	8	17	11
Correlated top-level network completion	25	10	15	0
Correlated transport + stabilized rendered state	25	9	16	0

Median domcontentloaded latency was 1,071.519 ms. Median additional title-settlement wait was 415.049 ms, and median body-stability wait was 524.570 ms.

The production sample changes the interpretation materially. Correlated network completion retained all registered successful navigations and rejected the registered HTTP-404 failures, but it accepted all ten semantic HTTP-200 negative trials. Adding generic rendered-state stabilization rejected only one of those ten semantic negatives. Thus 9/10 semantic HTTP-200 negative trials remained false positives under the generic committed-state heuristic.

7. Discussion

7.1 DOM settlement is an observation, not a transaction receipt

The controlled matrices intentionally placed optimistic UI mutation before terminal outcome. Under that construction, optimistic DOM settlement accepted every failure. The production sample was less extreme but still produced both false positives and false negatives at domcontentloaded. The generalizable point is not the measured rate. It is the evidentiary boundary: a DOM observation cannot by itself prove a request or application transaction completed.

7.2 Transport completion repairs transport ambiguity, not semantic ambiguity

Correlating the intended action with its target request is a substantial improvement over using arbitrary network activity. It distinguishes the causal target from background traffic and exposes explicit transport failure. The production sample shows the resulting boundary clearly: all registered HTTP-404 cases were rejected after correlation, while semantic HTTP-200 failures remained accepted.

HTTP 200 therefore must not be equated with semantic success when an application can encode failure, emptiness, rejection, or unmet task criteria in a successful transport response.

7.3 The controlled “perfect” committed-state result was oracle-specific

In both controlled engines, the committed-state oracle encoded the fixture’s intended application outcome. Its 320/320 classification therefore demonstrates that a correctly specified semantic oracle can separate the registered fixture cases; it does not show that generic DOM stabilization is a universal semantic oracle.

Production Arm B is the critical falsification pressure. A stable title and stable body were still insufficient for nine of ten semantic HTTP-200 negative search trials. The page had genuinely settled; it had simply settled into a state that did not satisfy the intended action.

The architectural implication is stronger than “wait longer.” Waiting longer can reduce premature observation, but time and stability cannot define task semantics. The verifier must know what success means for the action.

7.4 Action-specific success contracts

A robust autonomous-browser verifier should define success at action registration time. A contract can combine:

- the causal action identifier;
- the expected target request or request family;
- terminal transport success/failure semantics;
- the application artifact that constitutes success;
- a stable identifier or version where available;
- explicit negative states;
- timeout and ambiguity handling; and
- a postcondition that can be independently re-read when practical.

For a navigation, an acceptable contract may include canonical destination and expected content identity. For a search, “HTTP 200 and stable page” is weak; a stronger contract might require a result-set predicate tied to the query or an explicit empty-result outcome when emptiness itself is the intended result. For a create/update operation, the strongest contract may be an identifier returned by the operation plus an independent read-back showing the expected committed state.

The contract should fail closed when correlation is ambiguous. Treating an unrelated successful request as proof is specifically contradicted by the controlled negative-control result.

8. Threats to validity and limitations

First, the controlled scenario distributions were constructed. Their false-positive and false-negative proportions are not estimates of real-world prevalence. The controlled experiments are instrumentation and mechanism tests.

Second, the production sample has only fifty trials on one owner-controlled WordPress site. It cannot establish application-wide, browser-wide, or industry-wide reliability rates.

Third, production Arm B used Firefox. Chromium and Firefox controlled results agreed under the registered fixture, but browser equivalence for arbitrary applications is not established.

Fourth, the production semantic-negative cases were search cases. Other application semantics—mutations, payments, uploads, authentication transitions, long-running jobs, or multi-step workflows—may expose different failure modes.

Fifth, generic body-text stability is only one rendering-stability predicate. A different generic predicate may perform differently, but Arm B demonstrates the conceptual limitation: rendering stability and semantic correctness are different variables.

Sixth, long-lived transports such as WebSocket, Server-Sent Events, and long polling require separate settlement contracts. A request-finished primitive is not a universal completion signal for those protocols.

Seventh, the production implementation required two pre-rerun amendments after instrumentation failures were exposed. Those runs are retained, and the manifest and independent labels were unchanged, but the final v1.2 measurement procedure was not the initial procedure. This is reported explicitly to avoid post-hoc invisibility.

Finally, no claim is made that the measured latency costs are representative. They reflect the registered fixture timings and the selected production page-settlement predicates.

9. Operational verification architecture

A practical verifier can be organized as a monotonic evidence state machine:

1. Dispatch: record action identity, target, and expected semantic contract before execution.
2. Observe UI: record optimistic/intermediate UI transitions but do not close the action solely from them unless the action contract explicitly says UI presence is sufficient.
3. Correlate: bind network observations to the action and target; reject ambiguous unrelated activity as completion evidence.
4. Resolve transport: record terminal success, failure, cancellation, redirect chain, cache/service-worker classification, and timeout behavior.
5. Evaluate semantics: apply the action-specific postcondition independently of generic page stability.
6. Re-read when feasible: for durable state changes, verify the committed artifact through a separate read path.
7. Close or fail closed: mark success only when the registered contract is satisfied; otherwise retain explicit failure, timeout, or ambiguity.

This architecture does not require every browser action to use every layer. It requires the declared success claim to match the evidence actually obtained.

10. Conclusion

The experiments support a bounded conclusion. Optimistic DOM settlement can occur before the underlying action is resolved. Correlated terminal transport is stronger because it separates target-request completion from unrelated activity and transport failure. But transport success is not application success, and generic rendered-state stabilization is not semantic success.

The controlled Chromium and Firefox fixtures showed that a fixture-specific committed-state oracle could separate every registered case. The independently labeled production sample then exposed the limit of that result: a generic stable rendered state still accepted 9 of 10 semantic HTTP-200 negatives. This is not a reason to abandon network evidence. It is a reason to stop asking network or DOM evidence to prove more than they can.

For autonomous web agents, the safer design is explicit action-to-request correlation followed by an action-specific semantic success contract. The DOM tells what rendered. The network tells what transported. The contract tells whether the intended action succeeded.

References

- [1] Chrome DevTools Protocol, "Network domain," official protocol documentation. <https://chromedevtools.github.io/devtools-protocol/tot/Network/>
- [2] W3C, "WebDriver BiDi," W3C specification. <https://www.w3.org/TR/webdriver-bidi/>
- [3] M. Hashemi, A. Tahir, and N. Rasheed, "An Empirical Study of Flaky Tests in JavaScript," 2022. <https://arxiv.org/abs/2207.01047>
- [4] A. Romano et al., "An Empirical Analysis of UI-based Flaky Tests," 2021. <https://arxiv.org/abs/2103.02669>

Reproducibility and evidence receipts

This draft is derived from the frozen candidate-159068 evidence set. Principal retained artifacts include:

- candidate_159068_expanded_preregistration_v2.md
- candidate_159068_expanded_registered/expanded_summary.json
- candidate_159068_expanded_registered/expanded_runs.csv
- candidate_159068_external_validity_preregistration_v3.md
- candidate_159068_firefox_registered_v3/firefox_summary.json
- candidate_159068_firefox_registered_v3/firefox_runs.csv
- candidate_159068_arm_b_manifest_v1.json
- candidate_159068_arm_b_truth_labels_v1.json
- candidate_159068_arm_b_amendment_v1_1.md
- candidate_159068_arm_b_amendment_v1_2.md
- candidate_159068_arm_b_registered_v1_2/arm_b_summary.json
- candidate_159068_arm_b_registered_v1_2/arm_b_runs.csv

Publication is intentionally not performed by this draft step. The standing publication contract requires an external review path that is incompatible with the current ChatGPT-only responsibility constraint.