Reading Coherence Failure in a Live Web Agent

A capability-graded study of Agent-E under retained-window pressure

Embedded Risk Analytics · A Fathom Case Study · August 2026

We ran Agent-E, the open-source web-navigation agent published by Emergence AI, on a controlled and interaction-dense ordering task, across two model tiers and a range of retained-window budgets, and we read the agent's committed state with Fathom. Fathom is an information-theoretic instrument that reconstructs what an agent has actually committed from the agent's own action stream, and it applies three independent coherence reads to that reconstructed state. The reads are oracle-free. They consult only the agent's own actions and tool returns, and they never consult ground truth.

One result stands out. At a fair retained window a frontier model completed the task correctly and all three reads stayed silent. When the window was tightened to the point where a subtask could no longer complete its confirmation step, the same frontier model silently duplicated the order while every conventional success signal stayed green, and a committed-state read recovered each duplicate from the action stream alone.

Fathom reads committed state, and reads it from the action stream.

Fathom reconstructs committed state by folding an agent's actions and tool returns into the set of facts the agent has actually written. The fold obeys one rule that a transcript does not: a tool return marked as a failure is treated as a no-op, so an attempted action that did not take effect leaves nothing behind in the state. From that reconstructed state the instrument runs a contradiction detector, which flags any committing action that depends on a fact the agent has already removed or overwritten. The snapshot the detector reads is sized to the live facts, so its cost tracks the state the agent holds and stays flat as the action history grows. Three reads consume this one state, each looking for a different way an agent can act against what it has committed.

Agent-E structures a web task as a planner and a browser executor.

As we read the run logs, Agent-E pairs a planner that decomposes a task into single-action subtasks with a browser-navigation executor that carries each subtask out against a live page. Each subtask opens by re-grounding on a fresh view of the page. This per-subtask re-grounding is a real defense. It keeps element references current and closes the classic failure in which an agent reuses a selector that has scrolled out of view. The executor operates within a retained window that bounds how many turns of a subtask it can hold at once, and that window is the pressure we vary in this study.

One task, three reads of the same committed state.

The task runs on a controlled order builder at a single address, where adding or removing a line re-renders the table and shifts the element identifiers underneath the agent. The agent is asked to add three catalog items, set two quantities, apply a coupon, and place the order. Fathom reads the resulting action stream three ways.

The stale-element read flags an action taken on a page element that the agent has not observed on the page it is currently on, which is the web form of acting on state the agent can no longer see. Where the log carries no observation of the current page, the read abstains rather than guess.

The premature-commitment read flags an order-mutating action taken after the order has already been placed, which is a plan-level contradiction that every individual click can hide.

The duplicate-state read flags the addition of a line item that the agent has already observed present in the order, which is a duplicate of a row the agent can see.

The reads are graded by model capability.

The three reads are quiet on a coherent run and speak on a broken one, and which read speaks depends on the model and the window. In a separate run on the same task family, a smaller model committed the order prematurely and then continued to operate it, and the premature-commitment read flagged the resulting cascade of post-commitment actions. With a frontier model at a fair window the same task ran cleanly. Under window pressure the frontier model failed in a different way, and a different read caught it. Table 1 sets the two frontier conditions side by side.

Condition	Outcome	Stale-element read	Premature-commitment read	Duplicate-state read
Frontier model, fair window	Order correct: 3 lines, 6 units, $44.10	Silent	Silent	Silent
Frontier model, starved window	Order duplicated: 8 lines, 15 units, $106.20	Silent	Silent	Flagged: 5 duplicate adds

Table 1. The same three reads over the same task at two retained-window budgets for a frontier model. Each cell reports whether the read flagged a coherence break. Figures are from single controlled runs.

The failure the first two reads do not see.

The starved run duplicated the order by a specific route, and the route is what makes it hard to catch. When the window is tight, the add-item subtask is cut off before its confirmation returns, and the planner receives an empty result. Having no confirmation that the add landed, the planner re-issues the add as a fresh subtask. That new subtask re-grounds on its own view of the page, which already shows the item present, and it adds the item a second time. The stale-element read stays quiet, because every click lands on an element the agent has just observed. The premature-commitment read stays quiet, because the duplication happens before the order is placed. The duplicate-state read is the one that speaks. It counts each item's rows in the agent's own observations and flags an add of an item whose observed count is already positive. On the starved run it recovered all five duplicates, two of one item and three of another, from the action logs alone, with no access to the model and no access to ground truth.

The duplicate-state read was developed and confirmed against this live failure. It keys on the mechanism that produced the duplication, the addition of an item the agent has already observed present, so it fires on the real event and holds silent when the agent re-grounds and adds each item once. That property was checked on the coherent run described next.

Silence on the coherent run is what makes a flag mean something.

The fair-window run is the control. The same three reads, run over a correct execution, all stayed silent. The instrument abstains where the log gives it nothing to verify, and it holds quiet where the agent holds its state. A read that fired on a clean run would carry no information. The value of the flag on the starved run rests on the silence on the coherent one.

The failure returned a clean success.

On the starved run the agent reported the order placed, the page displayed a confirmation line, and no step raised an error. Every conventional signal read as success. The order was wrong by nine units and by more than sixty dollars. A crash counter would have scored the run as a pass, and so would a task-completion rate that reads the confirmation line. A read of committed state is what tells an order that was placed apart from an order that was placed correctly.

What a committed-state read offers an orchestration layer.

An orchestration layer that coordinates web and API agents holds coherence over a shared and growing state, and that shared state is where these failures concentrate. Fathom reads that state from the action stream, offline, with no access to model internals, and reports where coherence holds and where it breaks. The read runs over sample action logs from a single long-horizon agent, which makes it a low-friction way to see the instrument on real traces before any deeper integration. The same read that flags a break also informs when a repair is worth its cost, by telling a task that is hard on its own apart from an agent that has lost track of its own state. The method and its evidence are public at embeddedriskanalytics.com.

Method and conditions.

The task, the site, and the agent are held fixed. The model tier and the retained-window budget are varied. The reads are deterministic and oracle-free: they consume the agent's own actions and tool returns, apply success-gating so that a failed action is a no-op, and reconstruct committed state by a fold whose size tracks live facts. Agent-E is the open-source, MIT-licensed web-navigation agent published by Emergence AI, and the runs used its released planner and executor without modification. The figures in this note come from single controlled runs and are reported as such.