EMBEDDED RISK ANALYTICS · A FATHOM WHITEPAPER

The Real Cost of Agent Memory

An instrumented, cross-cloud benchmark of managed memory on Amazon Bedrock and Microsoft Azure

JULY 2026 · EMBEDDEDRISKANALYTICS.COM

Every team building a long-horizon agent makes a decision about how the agent remembers its own work, and most make it without measuring what the decision costs. The cost is real, it grows with the length of the run, and it lands in a place teams rarely look. We built a benchmark to measure it, and we ran the same benchmark on the two clouds where enterprises field agents today, Amazon Bedrock and Microsoft Azure.

Two results stand out. In the default configuration, the agent loses commitments it made to itself earlier in the run and then contradicts them. The premium managed-memory path holds those commitments well, and it re-grounds the agent on each one through a model-generated summary whose token cost climbs as the session accumulates state. Recovering a single committed fact after a hundred accumulated commitments cost as much as 155 times a deterministic lookup on one cloud and 85 times on the other, and neither path leaves a record an auditor can cite. This briefing reports how we measured that and what it means for anyone budgeting, building, or governing agents at length.

The problem with how agents remember

A long-horizon agent makes commitments to itself as it works. Over a long build a coding agent renames a field, changes a function signature, settles a routing decision, and a later step has to honor each one. When the run outgrows the platform's memory horizon, the agent works from a partial view of its own history and writes against a decision it can no longer see. The task in front of it was not hard. The agent handled it incorrectly because it lost an accurate account of what it had already done.

Every platform gives a team two ways to manage that history. There is a default path that serves back a recent window of the session, and there is a premium managed-memory path that extracts and stores the session and retrieves it by relevance. Teams choose between them early, usually before they have any measurement of what either one costs in practice, because the cost does not appear on the line item most teams watch.

The reason it stays hidden is because a model API is stateless, so everything the agent needs to remember is re-sent and re-billed on every call, and an agent makes many model calls over a single task. Memory in this setting is not an artifact you store once and pay for once. It is a stream you pay to re-present, on every step, for the length of the

run. The longer and more stateful the agent, the more that re-presentation costs, and long-horizon agents are precisely the ones that run long and accumulate state.

Public figures for this cost are vendor anecdotes and one-off blog numbers. There is a lack of independent measurements, instrumented the same way on more than one cloud, of what managed agent memory actually costs to keep an agent coherent. We provide such a measurement below.

The methodology of our cost measurement

The benchmark holds the task fixed and changes only how the agent remembers, so any difference in the result is the cost of remembering and nothing else. The agent maintains a small working service across a session and commits one change per turn. Because the committed state is code, the benchmark settles the true result by running it, which removes the usual problem of asking one model to grade another. The same instrument runs on both clouds, so a reading taken on Bedrock is directly comparable to the same reading taken on Azure.

> **WHAT THE BENCHMARK CONTROLS**
>
> Every measurement runs alongside a control that hands the agent its current state directly, which isolates ordinary task difficulty from the cost of self-tracking. A second control, in which there is no self-made commitment to lose, has to stay clean for a reading to count, which separates this effect from ordinary long-context drift. No model is called until a planted, known-answer case has shown that the instrument catches real failures and does not invent them. The claims each measurement supports are written down before the runs, and corrections are kept in the open.

The default configuration loses the agent's own commitments

On Bedrock, the default memory surface serves back a recent window of session events, on the order of a hundred. Once an early commitment falls behind that window the agent no longer sees it, and at the step that depends on it the agent writes against a stale picture of its own work. We did not tune a setting to force this. It follows from the platform's own default meeting the length of a realistic build. Measured on a strict rename task, the failure was present on both clouds and both model families we tested, while the control that hands the agent its state sat at zero throughout.

Table 1. Rate at which the agent breaks its own committed state on a strict rename task in the default setting, with no window tuned to force the failure. The control hands the agent its current state and isolates ordinary task difficulty. The deterministic layer re-surfaces the agent's own record.

Cloud and model	Default configuration	Given the state (control)	Deterministic layer
AWS Bedrock · Claude Sonnet 4.6	**0.50**	0.00	**0.00**
Azure Foundry · gpt-5-mini	**0.35**	0.00	**0.00**

Many teams ship on defaults and never turn on anything further, so for those deployments, the failure is the operating reality rather than an edge case. Azure's default thread retrieval behaves differently and does not drop the commitment through a window cap, and its exposure appears instead on its opt-in managed-memory preview,

which we turn to next alongside the premium Bedrock path.

Premium memory holds the state, and re-grounds it at a growing cost

The premium managed-memory path on each cloud is strong. We spent real effort trying to make it lose a commitment. We pushed it with dense sessions at scale, a schema key renamed several times in a chain, a dispatch table grown from ten to a hundred distinct entries, names that carry no inferable pattern, and both a frontier model and a mid-tier one. It preserved the specific commitment in every case, on both clouds. The story here is not that the platform forgets the agent's work.

The story is how it remembers, and what that process costs. The premium path reconstructs committed state from a model-generated summary and retrieves it by relevance, so the answer it hands back is a reconstruction produced on demand, and the size of that reconstruction grows as the agent accumulates commitments. The benchmark measures this directly. To re-ground the agent on one committed fact after a hundred accumulated commitments cost **3,727 tokens on Bedrock** and **2,034 on Azure**, against **24 tokens** for a deterministic lookup of the same fact, and the gap widens with every further commitment the agent makes.

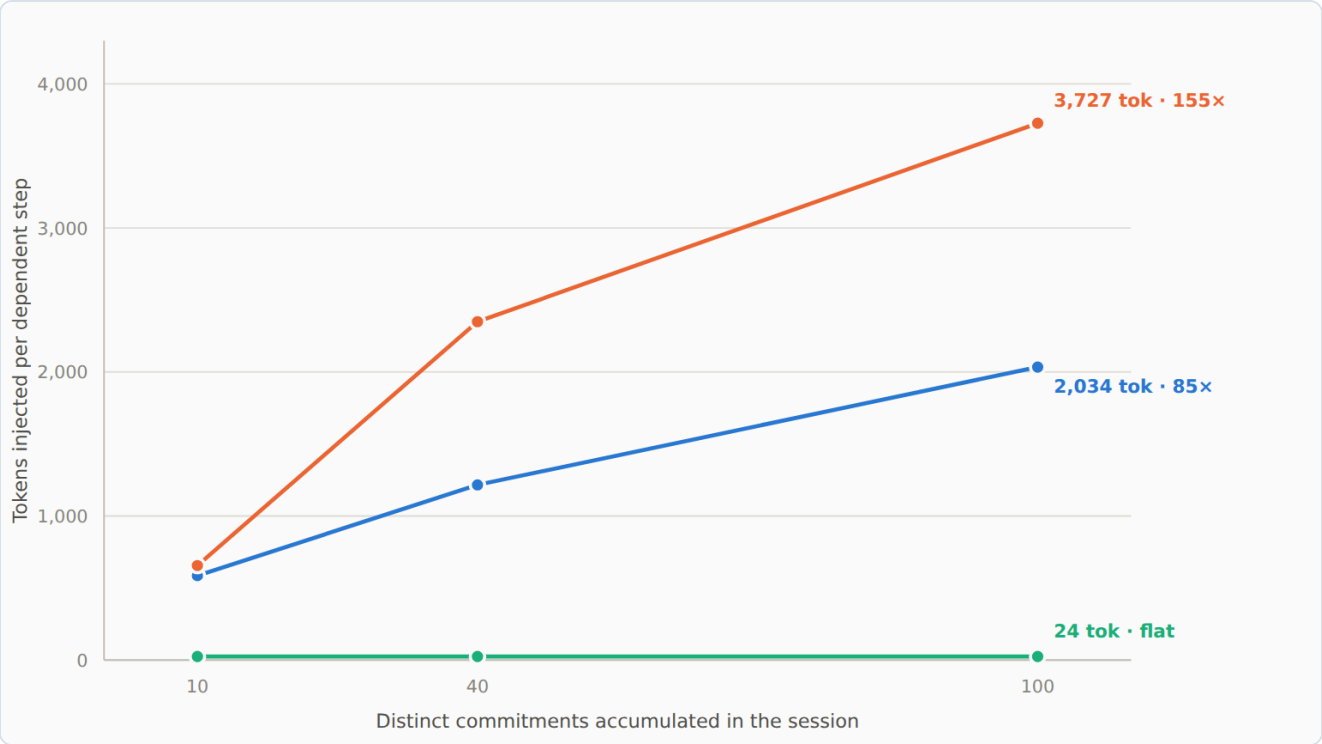

Figure 1. Tokens the agent must re-inject at a dependent step to recover one committed fact, as the number of distinct commitments in the session grows. AWS Bedrock in orange, Azure Foundry in blue, a deterministic committed-state layer in green. Both clouds recover the fact on every run. Bedrock rises to 155 times the deterministic layer at a hundred commitments and Azure to 85 times, while the deterministic layer holds flat at 24 tokens. Measured with identical arbitrary-name methodology on both clouds; the Azure figure is the mean of six independent extractions.

Azure adds a second property that a buyer should understand. Its extraction is stochastic, so the same session extracted twice yields different summary records and a different token cost, even though it recovered the specific commitment on every one of our runs. Bedrock is steadier from run to run and carries the higher absolute cost.

Either way the agent pays a model to rebuild its own past, again and again, and the bill scales with how much past there is.

There is a further property that no amount of token budget resolves. The premium path returns a probabilistic reconstruction rather than a stored record, so there is nothing an auditor can point to and nothing a control framework can cite. For a team operating agents in a regulated or high-assurance setting, that absence is the binding constraint, and it is structural to any approach built on a model summary.

The practical applications for builders

Three things follow for anyone running agents at length, whichever cloud they sit on.

1. **Know which configuration you actually ship.** The default path is where most agents run, and on Bedrock it loses the agent's own commitments once a build outgrows the window. Confirm the setting rather than assuming the premium path is on.
2. **Budget memory as a growing line rather than a fixed one.** The re-grounding cost bends upward with session length and with the number of distinct decisions the agent has made, which is exactly the regime long-horizon agents live in. A pilot that looks cheap at ten commitments can look very different at a hundred.
3. **If you are governed, treat the missing record as the first constraint.** Accuracy is necessary and it is not sufficient. A workload that has to satisfy an auditor needs a deterministic account of what the agent committed, and a model summary does not provide one.

A deterministic account of the agent's work via ERA's Fathom harness

The flat line in Figure 1 is a deterministic committed-state layer, part of our patent-pending agent harness Fathom, installed and measured on the same benchmark on both clouds. Fathom reconstructs the agent's committed state from the agent's own action stream, sized to the working set rather than to the whole growing history, and it produces a verbatim record by construction. It holds the re-grounding cost flat as the session accumulates state, it removes the run-to-run variation, and it gives a control framework something exact to cite. It runs on top of the platform's managed memory and leaves that memory in place, so the platform keeps doing what it already does well.

To request the benchmark dataset and a walkthrough under a mutual NDA, or to discuss the memory cost and audit exposure of a specific agent deployment on Bedrock or Azure, contact contact@embeddedriskanalytics.com or visit embeddedriskanalytics.com.
