How AI Agent Memory Works

The external scaffolding that lets a stateless model remember across turns, tasks, and whole sessions.

Alain AI Lab Research · Published July 14, 2026 · 10 min read

An AI agent that could not remember anything would be useless. It would forget your name between sentences, lose the task halfway through, and repeat work already done. Yet the language model at its core is, strictly speaking, forgetful by design: it holds nothing from one call to the next. Everything we experience as an agent "remembering" is scaffolding built around that forgetful core — a deliberate system for carrying information forward. Understanding AI agent memory means understanding that scaffolding: what it stores, where, and how it decides what to recall. It is the piece that turns a model that answers questions into an agent that works through tasks.

AT A GLANCE

THE CORE PROBLEM	SHORT-TERM	LONG-TERM
LLMs are stateless — no recall between calls	**The context window (working memory)**	**External stores + retrieval (RAG)**
MEMORY TYPES	MAIN FAILURE	STANDARD?
Episodic, semantic, procedural	**Wrong or stale context retrieved**	**None yet — open, fast-moving area**

01

Why Memory Is a Problem at All

The starting fact is counter-intuitive: a large language model is **stateless**. Each request it receives is processed in isolation, and the output depends only on the tokens supplied in that single call. The model keeps no hidden session, no running notebook, no recollection of the last thing it said. Ask it a question, get an answer, ask a follow-up, and — on its own — it has no idea the two are related. Statelessness here is the same idea as a stateless web

request: the same input produces the same kind of output, with nothing retained on the side.

So the “memory” in a chatbot or agent is an illusion produced by the surrounding software. When a conversation feels continuous, it is because the application is quietly re-sending the earlier turns with every new message. Memory, in other words, is not a feature of the model; it is an engineering job the system does on the model’s behalf. Every technique that follows exists to solve one problem — how to feed a forgetful engine the right information at the right moment without overwhelming it.

02

Short-Term Memory: The Context Window

The first and simplest form of memory is the **context window** — the block of text the model can read in a single pass, often described as its working memory. Everything the model “knows” in the moment must fit inside it: the system instructions, the conversation so far, retrieved documents, tool results, and the current question. Continuity comes from re-injecting the prior turns into this window on every call, which is why a longer chat costs more and runs slower — each message drags the whole history along behind it.

The window has a hard ceiling measured in tokens, and that ceiling has risen sharply. Early models could hold only a few thousand tokens; by 2024 and 2025 some models advertised windows of roughly a million tokens. But bigger is not simply better. A large advertised window is a capacity ceiling, not a promise of reliable recall across all of it, and models often use information near the start and end of a long input far better than material buried in the middle. Short-term memory is fast and precise, but it is finite, and it resets to empty the moment a session ends.

03

Long-Term Memory: Storing What Won’t Fit

Because the context window is bounded and temporary, anything an agent must remember across sessions has to live somewhere else — in an **external store** the agent can write to and read from. This is long-term memory, and it is what lets an agent recall a preference you stated last week or a fact it learned three tasks ago. The store is separate from the model entirely; the model gains access to it only when the relevant pieces are fetched and placed into the context window at the right time.

The dominant technology for this is the **vector database** paired with **embeddings**. An embedding model converts a piece of text into a list of numbers — a vector — positioned so that passages with similar meaning land near each other in mathematical space. Memories are embedded once and indexed. Later, when the agent needs to remember, it embeds the current query the same way and searches for the nearest stored vectors. This is **retrieval-augmented generation**, or RAG: the model's answer is augmented with memories retrieved by meaning rather than by exact keyword. Tools such as Pinecone, Weaviate, and Chroma provide this as a managed store, while FAISS offers the underlying similarity search as a library.

04

The Kinds of Memory an Agent Keeps

Not all memory is the same, and agent designers borrow a vocabulary from cognitive science to keep the kinds straight. **Episodic** memory holds specific past events — this user asked for that, on this date, and here is what happened. **Semantic** memory holds general facts and knowledge divorced from any single event — the user prefers metric units; the company's fiscal year ends in March. **Procedural** memory holds know-how: the steps and tool patterns for getting a kind of task done. The episodic-versus-semantic split traces specifically to the psychologist Endel Tulving in 1972; the procedural category comes from a broader later tradition, so the framework is rooted in cognitive science rather than a single source.

Layered underneath all of this is a deeper distinction between two kinds of knowledge. **Parametric memory** is what the model absorbed during training and baked into its weights — vast, but fixed at a knowledge cutoff and impossible to update without retraining. **Non-parametric memory** is the external, retrievable store described above — smaller, but current, editable, and traceable to a source. The term comes from the original RAG research, and the pairing captures the whole strategy: use the model's trained knowledge for fluency, and an external memory for facts that must stay fresh.

> Parametric memory is what the model learned once and cannot easily change. Non-parametric memory is what you can hand it fresh at runtime. Good agent design leans on the second for anything that must be current, personal, or verifiable.

05

How Retrieval Actually Works

The retrieval pipeline is worth seeing step by step, because most memory behaviour lives in it. First, source material is broken into **chunks** small enough to be useful. Each chunk is **embedded** into a vector and stored in the index. At query time, the agent embeds the incoming request, runs a **nearest-neighbour search** to pull the most similar chunks, and **injects** those into the prompt before the model answers. Chunk, embed, store, then embed the query, retrieve, and inject: that minimal loop is the spine of practically every long-term memory system, though production setups often add re-ranking and keyword filters on top.

Because the context window is finite, agents also need strategies to manage what stays in it. **Summarization** compresses old turns into a running summary so the gist survives while the token count shrinks. A **sliding window** keeps only the most recent exchanges and drops the oldest. **Selective retrieval** fetches just the relevant memories rather than everything. The frameworks explored in our guide to the major agent frameworks package these patterns, along with the "scratchpad" — a working notepad where the agent records its intermediate reasoning and tool results so a stateless model can carry state through a multi-step task.

06

When Memory Fails

Memory is where a lot of agent failures actually originate, and the modes are worth naming. The first is **context pollution**: retrieval fetches passages that are semantically close but not actually relevant, and the model dutifully treats the noise as signal. Closeness in vector space is not the same as usefulness, and a confidently wrong memory can be worse than none. The second is **staleness**: without a sense of time, an agent keeps surfacing outdated facts as current truth, and a store that accumulates long enough can hold two memories that flatly contradict each other.

The third is the **"lost in the middle"** effect, documented by Liu and colleagues in 2023: when relevant information sits in the middle of a long context, models use it markedly less well than the same information placed at the beginning or end, producing a U-shaped performance curve. Stuffing more into the window is therefore no guarantee it will be used. The fourth is simply **cost**: longer contexts and larger retrievals mean more tokens, more compute, and more latency, so memory is a budget to spend wisely, not a free resource.

07

Memory in Crypto Agents

For an agent that touches money, memory stops being a convenience and becomes a safety mechanism. A trading or wallet agent that forgets its open position, its risk limit, or an instruction you gave it earlier can repeat an action, breach a rule it was told to respect, or act on a stale view of the world. The stakes are the reason our report on what an AI agent in crypto is treats reliable state as a prerequisite, not a nicety. When a mistaken action is a final on-chain transaction, forgetting is expensive.

That raises the question of where memory should live. On-chain state is durable, public, and verifiable, but it is costly to write and effectively permanent; off-chain stores are cheaper, private, and editable. In practice most agent memory stays off-chain, with only the critical commitments — the ones that must be provable — recorded on-chain. There is an appealing idea that persistent, verifiable memory could plug into an on-chain agent-identity stack, of the kind the draft ERC-8004 standard sketches for identity and reputation. It is worth flagging honestly that this is a conceptual link only: that proposal addresses identity, reputation, and validation, and does not itself specify memory.

08

Poisoning, Privacy, and No Standard

Persistent memory introduces a distinct security risk: **memory poisoning**. If a hostile instruction can be slipped into an agent's long-term store — hidden in a document or a web page it reads — that instruction can be retrieved and acted on in a later session, long after the attack. This is close kin to prompt injection, which the OWASP security project ranks as the top risk to language-model applications in its 2025 list; the newer agentic guidance treats memory-and-context poisoning as its own category. The lesson is that a memory store is not just a filing cabinet but an attack surface that persists across time.

There is also a plainer concern: **privacy**. An agent that remembers is an agent that retains user data, which pulls in every question of consent, retention, and deliberate forgetting that data-protection regimes were built around. And underneath all of it sits an uncomfortable truth for anyone seeking certainty: there is no agreed standard for agent memory. It is an open, fragmented, fast-moving field of competing approaches — vector stores, knowledge graphs, summarization buffers, hybrid on-chain schemes — and any claim of a single "standard architecture" deserves scepticism. The useful way to judge an agent is not whether it has memory, but whether it remembers the right things, forgets the rest, and cannot be made to remember a lie.

“The memory of the just is blessed.”

PROVERBS 10:7

METHODOLOGY & SOURCES

This report was produced with a parallel multi-agent verification process, each claim checked and labelled verified, partial, or uncertain before inclusion. Numbers are kept approximate where precise public figures do not exist; no specific decimals, benchmark scores, or product token counts were invented, vector databases are named without ranking them, and the draft, memory-silent status of ERC-8004 is stated plainly rather than overstated.

Primary references include the standard treatment of stateless language models and context windows; the retrieval-augmented generation framing and the parametric versus non-parametric memory distinction from Lewis and colleagues (2020); the episodic–semantic memory split from Endel Tulving (1972) with procedural memory from the later declarative/procedural tradition; the “lost in the middle” positional finding from Liu and colleagues (arXiv:2307.03172, 2023); vector-store and framework memory patterns from LangGraph and common agent tooling; and the OWASP Top 10 for LLM Applications (2025) on prompt injection, with memory-and-context poisoning treated in the newer agentic security guidance. Educational content only; not financial advice.

How Do AI Agents Work? · AI Agent Frameworks: LangGraph, CrewAI & AutoGen · What Is an AI Agent in Crypto?

Alain AI Lab — independent crypto & AI research.
intelligencecrypto.org · This document is for educational purposes only and is not financial advice.