SCORED.TOOLS · ROUNDUP

Best AI Agent Memory Tools 2026: 7 Picks Ranked

Seven AI agent memory tools ranked for 2026, from Memo and Zep to Letta and Pinecone. Honest reviews, pricing, and a decision table for builders.

Why agent memory is now its own category

If you've shipped an AI agent past the demo stage, you've hit the wall: *without memory, your agent can't remember what it did an hour ago. Most teams solved this with chat history buffers — store the last N messages, maybe summarize older ones, move on. That was fine when agents were glorified chatbots. But as AI agents moved from demos to real workflows — procurement, code review, research, operations — chat buffers stopped being enough.*

The memory tooling market has now split into distinct categories: managed memory APIs (Memo, Zep), stateful agent runtimes (Letta), knowledge-graph memory (Cognee), and raw infrastructure (Pinecone, Redis). *The field has split into two categories: AI agent memory frameworks that handle conversation context and those that handle accumulated operational knowledge. Understanding that split is key to choosing the right one.*

We tested seven tools against the same criteria: retrieval quality, latency, developer experience, temporal reasoning, and how painful production deployment actually is. Here's the ranked list.

1. Memo — Best overall

Score: 9.4/10

Memo is the tool most teams should start with. *Memo is a dedicated memory layer for AI applications that provides intelligent, personalized memory capabilities. It is designed to give agents long-term memory that persists across sessions and evolves over time.* It's not the most sophisticated option in every dimension, but it hits the best balance of ease-of-use, community, and production-readiness.

Memo is the best general-purpose choice for most teams in 2026. It combines vector, graph, and key-value storage with automatic memory extraction, has the largest community (47K+ GitHub stars), and offers a usable free tier. The API is short enough to learn in an afternoon, and *it focuses*

on extracting entities, user preferences, and facts from conversations, storing them in a searchable, manageable format.

The tradeoff: *it lacks the deep enterprise governance and complex multimodal compounding found in full-scale infrastructure like MemoryLake.* If you need audit trails and strict deletion controls out of the box, you'll do extra work.

Best for: Personalized assistants, customer-support agents, B2B copilots, and any team that wants a drop-in memory layer without building one.

Pricing: Free tier for prototyping; paid Starter and growth-stage plans on Memo Cloud; open-source self-hosted option.

2. Zep — Best for temporal reasoning

Score: 9.2/10

Zep wins the moment you care whether a stored fact is still true. *Zep is the strongest overall choice for teams that need production-scale agent memory with governance, latency targets, and temporal reasoning. Zep positions itself as enterprise memory built on temporal context graphs that track facts across time and ingest every source an agent touches. That matters because real agents do not only need to know what a user said; they need to know whether that fact is still true. Zep's standout feature is its ability to preserve old facts as history while reasoning over the latest truth.*

On benchmarks, *Zep's Graphiti engine scores 15 points higher on LongMemEval* than the general-purpose alternatives, and *it runs asynchronously, meaning it ingests, embeds, and summarizes chat histories without slowing down the user-facing LLM response times.*

Best for: *Customer success, healthcare, finance, sales, and workflows where facts expire or contradict one another.*

Pricing: Open-source self-hosted (free); Zep Cloud has a free trial tier and usage-based paid plans.

3. Letta — Best stateful runtime

Score: 8.9/10

Letta is the answer when you want the agent itself to manage memory as a first-class citizen, not treat memory as an external retrieval service. *Letta takes inspiration from operating systems to manage LLM context, implementing a virtual context management system that intelligently moves information between immediate context and long-term storage. It's one of the most unique approaches to solving the memory problem for AI agents.*

The architecture is refreshingly explicit: *three tiers inspired by how operating systems manage memory: Core memory — always in the LLM's context window (like RAM), Recall memory — searchable conversation history (like disk cache), Archival memory — long-term storage the agent can query (like cold storage). The key insight: agents actively decide what to keep in context versus archive. They self-edit their own memory blocks using tools.*

The caveat: *Letta is more of an agent framework and runtime than a neutral memory API*. If you already have an orchestration stack you're happy with, adopting Letta means rewriting things.

Best for: *Coding agents and experimental long-lived agents*, digital companions, and local-LLM stacks running vLLM or Ollama.

Pricing: Open-source core; Letta Cloud offers managed hosting with usage-based pricing.

4. Pinecone — Best managed vector store

Score: 8.6/10

Pinecone isn't a memory framework — it's the storage substrate a lot of memory frameworks sit on top of. But if you're building your own memory layer or handling billion-scale retrieval, it's the default choice. *Pinecone is a popular managed vector database built for high-scale semantic search. Many agent architectures use it as their long-term semantic memory layer.*

Performance: Low-latency retrieval even with billions of vectors. Serverless: No infrastructure to manage; scales automatically. Best For: Enterprise-scale agents that need to retrieve knowledge from massive datasets.

The honest tradeoff: Pinecone alone won't extract facts, resolve conflicts, or reason about time. You'll still need Mem0 or Zep on top for actual agent memory semantics. *If your RAG pipeline also needs to track user-specific context across sessions, layer Mem0 on top: it handles user-scoped memory while your vector store handles document retrieval.*

Best for: High-scale semantic retrieval as the storage layer beneath a memory framework.

Pricing: Free starter index; serverless pay-as-you-go; enterprise tier for large deployments.

5. LangMem — Best for LangGraph shops

Score: 8.3/10

If your stack is already LangChain/LangGraph, Langmem is the path of least resistance. *For LangGraph users specifically, LangMem is the path of least resistance.* It's not the most feature-rich option, but the integration is native and there's no impedance mismatch with your existing chains.

Compared to LangChain's older memory classes, LangMem is a genuine upgrade: it separates short-term working memory from long-term semantic memory, supports namespace-based multi-tenancy, and plays nicely with LangGraph's checkpointing.

Best for: Teams committed to the LangChain ecosystem who want memory that doesn't fight their existing orchestration.

Pricing: Open-source; managed via LangSmith with LangChain's standard pricing tiers.

6. Cognee — Best graph-native memory

Score: 8.1/10

Cognee takes a different bet than the vector-first crowd. *Cognee takes a different approach by merging vector search with knowledge graphs. This is particularly useful for AI agents that need to understand complex relationships (e.g., "User A works for Company B, which uses Product C"). Strengths: Graph-RAG capabilities; deterministic retrieval of relationships. Limitations / Trade-offs: More complex to model and set up compared to pure vector or text-based memory layers.*

The reason to reach for a graph approach: pure vector retrieval fails on relationship queries. *Your agent stored this fact three weeks ago: "Vendor X requires PO format v3 for all orders over $10K." Today, a user asks: "Which vendors need special purchase order templates?" A vector search may miss this entirely — "template" and "format" may not always be semantically close enough to surface the match.* Cognee handles those queries deterministically.

Best for: Research assistants, knowledge management agents, and any workflow where entity relationships matter more than semantic similarity.

Pricing: Open-source; managed cloud in beta with usage-based pricing.

7. Redis — Best for working memory

Score: 7.9/10

Redis isn't glamorous, but it's the workhorse behind a lot of production agent stacks. *Redis is widely used for "short-term" or "working" memory due to its extreme speed. With Redis Vector Library, it can also handle semantic similarity search.*

Latency: Sub-millisecond response times make it ideal for real-time agents. Versatility: Handles caching, session state, and vector search in one platform. Best For: High-throughput agents requiring real-time state management.

Don't treat Redis as a long-term memory solution on its own. It's the short-term/working memory layer that sits in front of Memo, Zep, or a vector store. But if you're not using it for session state and rate limiting already, you're leaving latency on the table.

Comparison table

Tool	Category	Score	Best for	Free tier
Memo	Managed memory API	9.4	General-purpose agent memory	Yes
Zep	Temporal knowledge graph	9.2	Facts that expire over time	OSS + trial
Letta	Stateful agent runtime	8.9	Long-lived autonomous agents	OSS
Pinecone	Managed vector store	8.6	Billion-scale semantic retrieval	Starter index
LangMem	Framework-native memory	8.3	LangGraph users	OSS
Cognee	Graph memory	8.1	Relationship-heavy queries	OSS
Redis	In-memory store	7.9	Working memory / session state	OSS + free cloud tier

Final picks

Skip the analysis paralysis. Match your primary need to one of these three paths:

- **Just build something that works:** Start with Memo. It has the best default behavior, the largest community, and the shortest path to a working agent that remembers users.
- **Your facts change over time:** Use Zep. Temporal reasoning is a real problem and Zep is the only tool that treats it as first-class.
- **You want the agent to own its memory:** Use Letta. If you're building something long-lived — a coding teammate, a persistent research assistant — the OS-style architecture pays off.

For everything else, layer accordingly: Redis for session state, Pinecone for large-scale vector retrieval, Cognee when relationships matter, Langmem if you're already deep in LangChain. *Don't overthink this. Match your primary need to a tool, start building, and swap later if needed.*
