Agent Memory — What Works, What Doesn't

Academic consensus meets practitioner reality. A survey of memory architectures for AI agents — grounded in real failures from building 100-agent systems, 44GB of unmanaged logs, and the pipeline that finally worked.

February 2026

v1.0

Florian Ziesche · Ainary

CONTENTS

1. Executive Summary

Memory is the single biggest unsolved problem in production AI agents. Academic research has exploded — 100+ papers surveyed in February 2026 alone — but the gap between what papers propose and what practitioners can deploy remains enormous. This report bridges that gap with real numbers from real failures.

- **A 60-author survey** of memory mechanisms for foundation agents identifies five cognitive memory types and reviews hundreds of papers — confirming that the field has no consensus architecture[1] E
- **Memory-R1 demonstrates that agents can learn *when* to remember** using reinforcement learning, outperforming static heuristics with only 152 training examples[2] E
- **MIRIX introduces six-type modular memory** for multi-agent systems, achieving 85.4% accuracy on long-form conversation benchmarks while reducing storage by 99.9%[3] E
- **Our own system produced a 44GB debug log** in a single file, had 24 cron jobs writing to memory without coordination, and experienced context compaction destroying previously loaded memory — before we built the pipeline that now works E
- **The pipeline that works**: Daily notes → Weekly extraction → Monthly compression → MEMORY.md, validated across a 100-agent evolution experiment that produced 33,000 words of convergent protocol E

Keywords: *agent memory, episodic memory, semantic memory, working memory, reinforcement learning, multi-agent systems, memory management, context window, long-horizon reasoning*

2. The Academic Landscape 80%

(Confidence: High — based on comprehensive survey paper)

Memory is now the most active research frontier in agent AI. Four of the top ten agent papers in February 2026 focus on memory architectures — more than planning, tool use, or reasoning. E

In February 2026, Huang et al. published "Rethinking Memory Mechanisms of Foundation Agents in the Second Half" — a survey with **60 co-authors** spanning institutions from LMU Munich to Stanford, reviewing hundreds of papers on agent memory[1]. The survey establishes a taxonomy along three dimensions: **memory substrate** (internal vs. external), **cognitive mechanism** (episodic, semantic, sensory, working, and procedural), and **memory subject** (agent-centric vs. user-centric). E

The Five Memory Types

The survey draws directly from cognitive science to categorize agent memory:

Exhibit 1: Memory Taxonomy from the Survey

MEMORY TYPE	FUNCTION	AGENT EQUIVALENT	TRADE-OFF
Working Memory	Active manipulation of current information	Context window, scratchpads	Fast but extremely limited (tokens)
Episodic Memory	Specific past experiences with temporal context	Conversation logs, interaction histories	Rich but expensive to search
Semantic Memory	General knowledge and facts	Knowledge bases, extracted rules	Compact but loses context
Procedural Memory	How to do things	Tool usage patterns, workflows	Efficient but hard to update
Sensory Memory	Raw perceptual input	Screenshots, audio, multimodal input	Massive volume, needs filtering

Source: Huang et al. (2026), "Rethinking Memory Mechanisms of Foundation Agents in the Second Half" [1]. Taxonomy synthesized from cognitive science literature applied to agent architectures.

The critical insight from the survey: **most production agent systems use only one or two memory types** — typically working memory (context window) supplemented by some form of retrieval-augmented generation (RAG) acting as a primitive episodic memory. The full cognitive stack remains theoretical.

The survey also distinguishes between **agent-centric memory** (what the agent needs to complete tasks) and **user-centric memory** (what the agent needs to personalize for its user). This distinction matters: most research focuses on agent-centric memory, while most practitioner pain points are user-centric — remembering preferences, past decisions, evolving context.

The academic consensus is clear: **static, heuristic-driven memory pipelines are insufficient**. Agents need learned memory policies that adapt to task requirements. But the gap between this consensus and production reality is wide.

The survey documents hundreds of proposed architectures; the number deployed at scale is close to zero. I

> SO WHAT?
>
> If you're building an agent memory system, start by asking: which of the five memory types does your system actually implement? Most systems implement only working memory (context window) and call everything else "RAG." The taxonomy gives you a vocabulary for what's missing — and the survey shows you what academic solutions have been proposed for each gap.

3. Memory-R1: When to Remember 78%

(Confidence: High — peer-reviewed, benchmarked across three datasets)

The most important question in agent memory is not "what to remember" — it is "when to store, when to retrieve, when to update, and when to do nothing." Memory-R1 is the first system to learn these decisions via reinforcement learning. E

Yan et al. present Memory-R1[2], a reinforcement learning framework with two specialized agents: a **Memory Manager** that learns structured operations (ADD, UPDATE, DELETE, NOOP) and an **Answer Agent** that pre-selects and reasons over relevant memory entries. Both agents are fine-tuned with outcome-driven RL (PPO and GRPO). E

152

training QA pairs needed

Memory-R1 [2]

3B–14B

parameter range validated

Memory-R1 [2]

3

benchmarks (LoCoMo, MSC, LongMemEval)

Memory-R1 [2]

What makes Memory-R1 significant is not the performance numbers but the conceptual shift: **memory management is a learnable skill, not a fixed pipeline**. Static systems apply the same rules regardless of context — store everything, retrieve by similarity, hope for the best. Memory-R1's agents learn that sometimes the correct memory operation is NOOP — doing nothing. Sometimes the correct operation is DELETE — actively forgetting information that has become stale or misleading. I

The framework validates across three challenging benchmarks: LoCoMo (long-form conversation), MSC (multi-session chat), and LongMemEval (long-term

memory evaluation). With only 152 training examples, it outperforms strong baselines including full-context approaches. The low training data requirement is particularly important for practitioners: it means this approach could be fine-tuned on domain-specific memory management tasks without massive datasets. E

Why This Matters for Practitioners

Every production agent system we've seen (including our own) uses hardcoded rules for memory: "store every conversation," "retrieve top-k by embedding similarity," "compress after N tokens." These rules are brittle. They don't distinguish between information worth remembering for months and information that's relevant for five minutes. Memory-R1 proves the alternative works — but the gap from paper to production is the engineering, not the science. J

> SO WHAT?
>
> If you're running static memory heuristics (and you almost certainly are), Memory-R1 shows what the next generation looks like: learned policies that decide when to store, update, delete, or ignore. The 152-example training requirement means this is within reach of most teams. The question is whether your memory infrastructure supports dynamic operations, or whether it's append-only.

4. MIRIX: Shared Memory Across Agents 70%

(Confidence: Medium-High — strong benchmarks, limited production validation)

Multi-agent systems without shared memory are just multiple single agents. MIRIX is the first system to implement six distinct, coordinated memory types across agent teams — and it works. E

Wang and Chen introduce MIRIX[3], a modular multi-agent memory system with six memory components: **Core Memory** (persistent identity and preferences), **Episodic Memory** (specific past experiences), **Semantic Memory** (abstracted knowledge), **Procedural Memory** (learned workflows), **Resource Memory** (external references), and **Knowledge Vault** (verified, high-confidence facts). A multi-agent framework dynamically controls updates and retrieval across all six types. E

85.4%

accuracy on LOCOMO benchmark (SOTA)

MIRIX [3]

99.9%

storage reduction vs. raw retention

MIRIX [3]

+35%

accuracy vs. RAG baseline on ScreenshotVQA

MIRIX [3]

MIRIX addresses a problem that every multi-agent builder encounters: **how do agents share knowledge without stepping on each other?** The six-type architecture provides natural access control — procedural memory might be agent-specific, while semantic memory is shared across the team. The multi-agent framework coordinates updates, preventing the conflict resolution nightmares that plague naive shared-memory approaches. I

The ScreenshotVQA result is particularly striking: MIRIX processes nearly 20,000 high-resolution screenshots per sequence, a task where no existing memory system could previously be applied. By structuring memory into typed layers, MIRIX achieves dramatic storage reduction (99.9%) while *increasing* accuracy by 35% over RAG baselines. This validates the core thesis that **structured forgetting beats total retention**. E

The system also provides a packaged desktop application that monitors screens in real time, builds personalized memory bases, and offers visualization with local storage for privacy. This matters: it means the architecture has been built beyond paper-level prototypes. E

> SO WHAT?
>
> If you're building multi-agent systems, MIRIX's six-type taxonomy is the most mature reference architecture available. The 99.9% storage reduction proves that "keep everything" is not just wasteful — it actively hurts accuracy. Start by mapping your current memory to MIRIX's types: which do you have? Which are missing? The gaps will tell you where your agents are flying blind.

5. Case Study: What Happened When We Didn't Manage Memory 90%

(Confidence: Very High — first-party data, observed directly)

Before we built a memory pipeline, our agent system produced a single 44GB debug log file, ran 24 concurrent cron jobs all writing to memory without coordination, and experienced context compaction silently destroying previously loaded memories. These are not hypothetical risks — they are documented failures with exact numbers. E

The 44GB Log

On February 16, 2026, during a disk space audit, we discovered `cache-trace.jsonl` — a single debug log file consuming **44GB** of disk space. The file was a trace log that had been accumulating without any rotation, compression, or retention policy. On a MacBook Air with limited storage, this single file had consumed all available space, forcing emergency cleanup that included deleting browser caches (8.5GB) and npm caches (5.4GB) just to continue operating. E

The 44GB log is a perfect example of what happens when memory has no forgetting discipline: **unbounded accumulation eventually destroys the system it was meant to support**. The log contained useful debugging information. All of it. Forever. Until there was no disk space left for the actual work. I

The 24 Uncoordinated Crons

Our agent system ran 24 cron jobs, many of which wrote to shared memory files — MEMORY.md, daily notes, topic files. These crons had no coordination mechanism. Multiple crons could attempt to update the same file within the same minute. There was no locking, no conflict resolution, no awareness of what other crons had recently written. E

The result was memory corruption through overwrite: Cron A would read MEMORY.md, prepare an update, and write it back. Cron B, which had read the

same file milliseconds earlier, would write its own update — overwriting Cron A's changes entirely. Information was being both added and lost simultaneously. We discovered this only after noticing that memory entries would appear and disappear between sessions. I

The fix was structural: we added a rule that **crons MUST NOT modify SOUL.md, AGENTS.md, or MEMORY.md autonomously**. Only the main session, with human oversight, can write to core identity and long-term memory files. This is the agent equivalent of database write locking — simple, but it took a production failure to implement it. E

Context Compaction Destroying Loaded Memory

The most insidious failure was context compaction. When the agent's context window filled up, the platform automatically compacted older context to make room for new tokens. This compaction is designed to preserve the most relevant information — but "relevant" is determined heuristically, and it has no knowledge of what was *loaded from memory files* vs. what emerged during the conversation. I

The result: carefully loaded memory — preferences, project context, decision history read from MEMORY.md at session start — would be silently compressed away as the conversation progressed. The agent would begin the session knowing the user's communication preferences, active projects, and established patterns. Two hours later, after extensive work had expanded the context, compaction would strip those memory-loaded facts to make room. The agent would continue operating, but with its foundational context gone — making decisions that contradicted established preferences without any awareness that it had "forgotten." I

This failure mode is not documented in any of the academic papers we surveyed. It is uniquely a production problem: academic benchmarks use controlled contexts, not dynamic sessions where loaded memory competes with generated content for limited context space. J

44 GB

single unmanaged log file

24

concurrent crons writing to memory

First-party, Feb 16 2026

First-party, Feb 2026

11 GB

main memory database (main.sqlite)

First-party, Feb 16 2026

SO WHAT?

These failures share a common root: **no forgetting discipline**. The 44GB log never forgot anything. The 24 crons never coordinated who should remember what. Context compaction "forgot" the wrong things. Production memory systems need explicit policies for what to keep, what to discard, and who has write access — before the system teaches you the hard way.

6. The Memory Pipeline That Works 75%

(Confidence: High for our use case — generalizability unproven)

After the failures documented in Section 5, we built a memory pipeline through a 100-agent evolution experiment. Ten groups of ten agents, each using different thinking methods, converged on the same architecture independently. That convergence is the strongest signal we have that the pipeline works. E

The Experiment

We ran a 100-agent experiment to discover how self-improving agents should manage memory. Ten groups of agents (each using a different analytical method: First Principles, Inversion, Analogical, Adversarial, Quantitative, Socratic, Constraint, Narrative, Systems, and Random Mutation) independently designed memory protocols. The experiment produced **33,000 words** of transcripts across 222,207 characters of source material.[4] E

The convergence was striking: **10 out of 10 groups independently identified "Files = Intelligence"** as the foundational law — that external memory files are the *only* improvement mechanism for stateless agents. 8 out of 10 converged on multi-timescale feedback loops. The memory pipeline emerged not from one group's design but from the intersection of all ten. E

The Pipeline

Exhibit 2: The Memory Pipeline (Production)

STAGE	CADENCE	INPUT	OUTPUT	RULE
Raw Interaction	Continuous	Every conversation	Narrative daily log	Story format, not just facts
Daily Notes	Daily	Session events, decisions, corrections	`memory/YYYY-MM-DD.md`	Write immediately, not at session end
Weekly Extraction	Weekly	7 daily notes	Behavioral rules, pattern updates	Only patterns that repeat 5+ times
Monthly Compression	Monthly	4 weekly extractions	MEMORY.md updates	"Fits on one page? Keep. Doesn't? Cut."
Hub Memory ID	Monthly	Full memory graph	Top 10 most-connected nodes	Never auto-pruned, weight retrieval

Source: SYNTHESIS-v2.md [4], synthesized from 100-agent experiment. Pipeline has been in production since February 2026.

The File Structure

The experiment converged on a specific file structure that maps roughly to the cognitive memory types identified by the academic survey[1]:

- **SOUL.md** — Agent identity, anti-patterns, protocol (≈ procedural memory)
- **USER.md** — Co-authored user model (≈ user-centric semantic memory)
- **MEMORY.md** — Curated long-term wisdom (≈ semantic memory)
- **memory/YYYY-MM-DD.md** — Daily narrative logs (≈ episodic memory)
- **memory/kintsugi.md** — Golden repairs from failures (≈ episodic, high-value subset)

- **memory/graveyard.md** — Killed beliefs + reasoning (≈ negative semantic memory)
- **memory/preferences.md** — Structured behavioral preferences (≈ user-centric procedural)

The Forgetting Discipline

The experiment identified three tiers of memory persistence — a direct parallel to MIRIX's observation that 99.9% storage reduction *improves* accuracy:

- **Tier 1 (Permanent):** Identity core, hub memories, kintsugi entries
- **Tier 2 (90-day half-life):** Active project context, current preferences
- **Tier 3 (30-day half-life):** Ephemeral context, temporary states

Monthly: anything unaccessed for 60 days and not in Tier 1 → archive. The critical rule: **"Ask once before major deletions."** Automated forgetting without human oversight produced the same corrupted-memory problems that motivated the pipeline in the first place. E

What the Experiment Found That the Papers Didn't

Three findings from the 100-agent experiment that we did not find in any surveyed paper:

1. **The Belief Graveyard.** Killed beliefs must be permanently archived and searchable, or they return as "zombie beliefs" — assumptions the agent re-derives from the same signals that produced them originally. No academic system we surveyed implements negative memory (things the agent has explicitly decided are *not* true). J
2. **Hub Memories.** Some memories are "mother trees" — they connect to many other memories and explain multiple behaviors. These should never be auto-pruned and should weight retrieval. This is conceptually similar to MIRIX's Core Memory, but identified independently via biological analogy (mycorrhizal networks). I
3. **"Mental notes" don't survive.** The simplest, most important lesson: if the agent thinks "I should remember this" but doesn't write it to a file, it is lost. 100% of the time. Every agent group rediscovered this independently. The

implication: **write-first architectures** (write to memory immediately, not at session end) dramatically outperform write-deferred architectures. E

SO WHAT?

The pipeline is simple: daily notes, weekly extraction, monthly compression, explicit tiers, and write-first. The value is not in complexity but in consistency — a pipeline that runs every day beats a sophisticated architecture that runs sporadically. Start with daily notes. Everything else builds on that foundation.

7. What's Missing 65%

(Confidence: Medium-High — informed by both literature gaps and practitioner needs)

Despite the academic explosion and our own production experience, five critical capabilities remain unsolved in both research and practice.

1. Active Forgetting

Memory-R1 proves that agents can learn DELETE operations[2]. MIRIX achieves 99.9% storage reduction[3]. Our experiment produced explicit forgetting tiers. But **no production system implements relevance-scored, learned forgetting at scale**. Current forgetting is either manual ("archive after 60 days") or heuristic ("drop lowest-similarity entries"). An agent that actively learns what to forget — not just what to remember — does not yet exist outside research papers. J

2. Relevance Scoring That Adapts

Current retrieval relies on embedding similarity — cosine distance between a query and stored memories. This is better than nothing but worse than what's needed. A memory's relevance depends on the current task, the user's current goals, the time since storage, and the memory's connection to other memories. **Static embedding similarity captures none of this.** Memory-R1's approach (learned retrieval) is the right direction, but it has not been validated outside controlled benchmarks. J

3. Shared Memory Between Sessions

Our pipeline uses files as the inter-session bridge: today's session reads yesterday's daily notes. This works for a single agent with a single user. But it does not solve: (a) multiple agents sharing memory about the same user, (b) the same agent running in multiple concurrent sessions, or (c) agents handing off context to different agents mid-workflow. MIRIX addresses multi-agent memory[3], but the practical integration challenge — how do you connect a MIRIX-style system to existing agent platforms? — remains unsolved. I

4. Context Compaction Awareness

No research paper we surveyed addresses the production problem of context compaction destroying loaded memory (Section 5). This is because academic benchmarks operate in controlled contexts where the full memory is always available. In production, where agents run for hours and context windows fill up, **the platform's own memory management (compaction) fights against the agent's memory management (loaded files)**. A memory system that is aware of — and can interact with — the platform's compaction layer does not exist. J

5. Static Retrieval vs. Learned Policies

The gap between the academic frontier (Memory-R1's learned retrieval policies) and production reality (hardcoded "retrieve top-5 by similarity") is approximately 2-3 years of engineering. Memory-R1 works with 152 training examples — the science is solved. But integrating RL-trained memory management into existing agent frameworks (LangChain, CrewAI, AutoGen, OpenClaw) requires infrastructure that doesn't exist: **reward signals from task outcomes piped back to memory operations, fine-tuning loops that run alongside production inference, and evaluation frameworks that measure memory quality beyond benchmark accuracy**. J

> SO WHAT?
>
> These five gaps define the next 12-18 months of agent memory development. If you're building today, invest in the pipeline (Section 6) — it's the best available. But architect for extensibility: your memory system should support dynamic operations (not just append), learned policies (not just hardcoded rules), and multi-agent coordination (not just single-agent files). The systems that can evolve toward these capabilities will outperform those that can't.

8. Recommendations 70%

(Confidence: Medium-High — grounded in practice, limited by single-team experience)

These recommendations target practitioners building agent memory systems — developers, architects, and technical leads. They synthesize the academic research (Sections 2-4) with our production experience (Sections 5-6) and the gap analysis (Section 7).

Start Today

1. **Implement daily notes immediately.** One file per day, narrative format, written during the session (not after). This is the foundation everything else builds on. Cost: zero. Value: everything. J
2. **Add write locking on critical memory files.** If multiple agents or crons can write to the same file, you will lose data. Our rule: only the main session can write to SOUL.md, MEMORY.md, and AGENTS.md. Crons can write to daily notes only. J
3. **Set a forgetting policy.** Define what gets archived after 30, 60, and 90 days. If you can't decide, start with: "If nobody accessed it in 60 days and it's not in the identity core, archive it." You can always restore from archive. J

Build This Quarter

4. **Implement the weekly extraction loop.** Every 7 days, process the week's daily notes into behavioral rules and pattern updates. Only promote patterns observed 5+ times. This is where noise becomes signal. J
5. **Map your memory to the five-type taxonomy.** Working, episodic, semantic, procedural, sensory — which do you have? MIRIX's six-type structure[3] is the most comprehensive reference architecture. Start with the gaps. J
6. **Implement a belief graveyard.** When you discover an agent assumption is wrong, don't just correct it — archive it with reasoning. Without this, the agent will re-derive the same wrong conclusion from the same signals. J

Plan For Next Year

7. **Watch Memory-R1 for production readiness.** The 152-example training requirement means RL-based memory management is within reach. When frameworks support reward-signal integration for memory operations, adopt early. J
8. **Architect for dynamic memory operations.** Your memory layer should support ADD, UPDATE, DELETE, and NOOP — not just append. This is a prerequisite for any learned memory policy. J
9. **Invest in context compaction awareness.** If your platform compacts context automatically, you need to understand what it preserves and what it discards. Consider re-loading critical memory entries after compaction events, or flagging memory entries as "compaction-resistant." J

Exhibit 3: Memory System Maturity Model

LEVEL	DESCRIPTION	INDICATOR
0: None	Context window only. Nothing persists.	Every session starts from zero
1: Append-Only	Logs stored, maybe RAG. No curation.	Storage grows unbounded
2: Curated	Pipeline with extraction + compression. Forgetting policy.	Storage is bounded, quality improves over time
3: Typed	Multiple memory types (episodic, semantic, procedural).	Different information stored differently
4: Learned	RL-based memory management. Dynamic operations.	Memory quality measurably improves via feedback
5: Coordinated	Multi-agent shared memory with access control.	Agents share knowledge without conflicts

Source: Author framework [J]. Most production systems are at Level 0-1. The pipeline in Section 6 achieves Level 2. MIRIX targets Level 3-5. Memory-R1 targets Level 4. No production system we're aware of has achieved Level 5.

9. Methodology & Limitations

Overall Confidence	72% (Medium-High). Academic sections rely on peer-reviewed papers with strong benchmarks. Practitioner sections rely on first-party data from a single team. Recommendations are judgment calls informed by both.
Sources	3 primary academic papers [1-3], 1 internal experiment synthesis [4], 2 daily memory logs (first-party), 1 memory structure document (first-party). Web search for supplementary context.
Strongest Evidence	Memory-R1 benchmarks (3 datasets, multiple model scales); MIRIX benchmarks (SOTA on LOCOMO, 99.9% storage reduction); 44GB log and 24-cron coordination failure (first-party, exact numbers).
Weakest Point	The pipeline (Section 6) has been validated by one team for one use case over weeks, not months. The 100-agent experiment produced convergent design but has not been A/B tested against alternatives. Generalizability to different agent platforms, team sizes, and domains is unproven.
What Would Invalidate	If (a) append-only memory with better retrieval outperforms curated pipelines, (b) context window sizes expand enough to eliminate the need for external memory, (c) platform-level memory management (e.g., built into OpenAI, Anthropic APIs) makes application-level pipelines unnecessary.

Confidence Per Section

SECTION	CONFIDENCE	BASIS
2. Academic Landscape	80%	60-author survey, hundreds of papers reviewed
3. Memory-R1	78%	Peer-reviewed, 3 benchmarks, multiple model scales
4. MIRIX	70%	Strong benchmarks, limited production validation
5. Case Study	90%	First-party data, exact numbers, directly observed
6. Pipeline	75%	100-agent convergence, weeks of production use
7. What's Missing	65%	Gap analysis from literature + practice, judgment-heavy
8. Recommendations	70%	Practice-grounded but single-team experience

Conflict of Interest

The publisher of this report builds and operates AI agent systems — and has a commercial interest in the conclusions presented here. The case study data is first-party: we are reporting our own failures and successes. Evaluate evidence independently; claims marked [J] reflect judgment, not evidence.

How This Report Was Produced

This report was produced using a multi-agent research pipeline: automated literature search, paper analysis, cross-referencing with production logs, and synthesis — each performed by specialized agents. The irony is not lost on us: the memory pipeline described in Section 6 is the same pipeline that enabled this report to be written.

References

[1] Huang, W.-C., Zhang, W., Liang, Y., et al. (60 authors). (2026). "Rethinking Memory Mechanisms of Foundation Agents in the Second Half: A Survey." arXiv:2602.06052. Feb 6, 2026. https://arxiv.org/abs/2602.06052

[2] Yan, S., Yang, X., Huang, Z., Nie, E., Ding, Z., Li, Z., Ma, X., Bi, J., Kersting, K., Pan, J.Z., Schütze, H., Tresp, V., Ma, Y. (2025). "Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning." arXiv:2508.19828. https://arxiv.org/abs/2508.19828

[3] Wang, Y., Chen, X. (2025). "MIRIX: Multi-Agent Memory System for LLM-Based Agents." arXiv:2507.07957. Jul 10, 2025. https://arxiv.org/abs/2507.07957

[4] Ainary Research. (2026). "THE GRAND SYNTHESIS v2.0 — Full-Transcript Analysis of 100-Agent Evolution Experiment." Internal document. 222,207 characters across 10 groups.

[5] Ainary Operations. (2026). Daily memory logs: memory/2026-02-16.md, memory/2026-02-17.md. First-party operational data.

[6] Ainary Operations. (2026). MEMORY.md — Production memory structure with layered loading protocol. First-party.

Cite as: Ainary Research. (2026). "Agent Memory — What Works, What Doesn't." AR-034, v1.0.

About This Report

AI strategy · research · implementation. By someone who built the systems first. This report was produced by Ainary's multi-agent research pipeline. ainaryventures.com

Ainary

AI strategy · research · implementation.
By someone who built the systems first.

Contact · Feedback