Remembering Without Room: A Memory Design for Local Small Language Models

A design viewpoint paper

James Sparenberg Brain-Research Project August 2026

Abstract

Small language models running on local hardware have a memory problem that big hosted models can paper over. When the context window is a few thousand usable tokens, an agent forgets what it decided ten turns ago, and the usual fix — stuff the whole transcript back into the prompt every turn — makes accuracy worse, not just slower. This paper argues that the answer is not a bigger window. It's a memory layer that keeps a short sketch in the prompt, keeps the evidence on disk, and keeps a working path between the two. This paper proposes a design built from four modules and four distinct ways of getting at stored information, and it argues that one constraint usually treated as a weakness in local deployment (slow disk access) is actually the thing that makes a better design affordable. This paper also lays out how one would know if any of this works, and what would tell developers to stop.

---

1. The problem is not forgetting. It's hoarding.

Watch an agent forty turns into a coding job. It has read nine files, run a dozen commands, and it's about to ask you a question you already answered on turn four [16].

Nothing malfunctioned. That's the design working as built. An agent is a stateless model wrapped in a loop that hands it the transcript again every single turn. Nothing persists on its own. So the loop compensates by pasting everything back in: files, tool output, error messages, all of it. Cost climbs with the square of turn count, because turn forty re-sends turns one through thirty-nine.

The obvious fix is a bigger window. Buy more room and stop worrying.

That fix doesn't work, and the reason is the interesting part.

Chroma tested eighteen frontier models on tasks a child could do [2]. Copy this text. Find this sentence. As the input got longer, the models got worse at it. Not at the edge of the window either. A 200,000-token window can start degrading somewhere around 50,000. The number printed on the box isn't the number you get in practice.

If the only cost of a long prompt were money, a bigger window would solve everything. Accuracy falling before the window fills is a different kind of problem. And for a 3B model with maybe four thousand tokens of reliable attention, the degradation zone starts almost right away.

So the agent that remembers by hoarding gets slower, more expensive, and dumber, all at once.

2. What the field already knows

Two findings from the 2026 survey literature shaped this proposal more than anything else.

First, from a survey covering work from 2022 through early 2026 [1]:

> The gap between "has memory" and "does not have memory" is often larger than the gap between different LLM backbones. Investing in memory architecture can yield returns that rival — or exceed — model scaling.

The ablations behind that claim are stark. Strip reflection out of Generative Agents and coherent multi-day planning collapses into repetitive noise inside 48 simulated hours [5]. Take the skill library away from Voyager and it reaches tech-tree milestones 15.3 times slower [18]. On MemoryArena, swapping an active memory agent for a long-context-only baseline drops task completion from over 80% to about 45% [15].

For anyone building on small local models, that's the whole argument. A small model with excellent memory isn't a compromise. It's a strategy.

Second finding, same survey: long context is not memory. Models that score near-perfectly on passive recall benchmarks like LoCoMo [17] fall to 40–60% on MemoryArena, where memory has to actually drive decisions across linked sessions. Passive recall aces make poor memory agents. That's an uncomfortable result for anyone planning to test a memory system by asking it to retrieve facts, and take it as a warning about then evaluation plan.

The survey also names the hard open question directly: when does an episodic record graduate into a semantic fact, and when does a semantic fact get pulled back into working memory for a specific job? Most systems, it says, implement two layers well and handle the transitions between them with crude heuristics. That's not a solved problem one can go look up. It's the frontier.

3. The design

Four modules. Each one useful on its own, testable on its own, and able to be thrown away on its own if it turns out to be wrong.

3.1 Working memory

Every tool call appends to a plain append-only ledger on disk. The full output never enters the prompt. What goes into the prompt instead is a compact sketch built by the harness, not by the model: what was tried, what worked, what blew up, what it learned. Every node in that sketch carries an ID that points back at the ledger record.

Two rules make this work.

The sketch is capped in tokens, hard. When it hits the cap, old branches fold into a single node and keep their ID. Facts already established — including every answer the user has already given — sit in a pinned section that survives folding. That pin is the actual fix for the turn-four/turn-forty bug.

The second rule matters more. Compression has to be reversible. A summary is one-way; once the detail is gone it's gone. A folded structure with live IDs can be opened again. The 2026 survey describes what happens when you get this wrong: each compression pass quietly drops low-frequency details, and after enough passes the agent remembers a sanitized, generic version of history — "precisely the kind of memory that fails on edge cases" [1].

For a coding agent the edge cases *are* the job. A memory that keeps "we set up plugins" and loses "the loader needs exactly two levels of directory nesting" is worse than no memory, because it looks like it worked.

3.2 Consolidation

Turning episodes into durable knowledge. This runs off the critical path: at session end, or when the machine is idle, never in the middle of a turn.

That placement is what makes quality achievable. Consolidation can afford several passes, contradiction checks, even a different and larger model than the one driving the session. Letta ships exactly this pattern under the name sleep-time compute, with a second agent updating the primary agent's memory in the background [4]. This process didn't invent it and that's good news — it means the shape already has had the risk removed.

One correction taken from the literature. Tencent's implementation extracts memories every five turns regardless of what happened [10]. Generative Agents fires reflection when the cumulative importance of recent observations crosses a threshold instead [5]. That's better. Ten turns of failed greps deserve one atom of memory. One turn where the user states a hard constraint deserves consolidation right now.

What gets written matters as much as when. Storing every interaction verbatim is tempting and almost always wrong, because noise degrades retrieval precision for everything else in the store [1]. The survey breaks the write path into five stages: filter out low-signal records, canonicalize dates and names and quantities into a normal form, deduplicate overlapping entries, score by relevance and novelty, then tag with timestamp and source and task. It had all of that as one undifferentiated step called "extract atoms." Canonicalization in particular is one that would have been skipped and regretted, since half of memory retrieval failures are really just the same fact written two ways.

3.3 The textbook

The durable artifact, and the piece that makes a *new* session competent rather than amnesiac.

It's markdown in git. A spine file (the table of contents), chapters underneath it, and the raw ledger below that. Four levels, each roughly ten times the size of the one above:

- **Spine**, 300–500 tokens. Read in full at every cold start.
- **Chapter summary**, about 100 tokens each. Read when the spine entry looks relevant.
- **Chapter body**, 1–3k tokens. Read when the summary isn't enough.
- **Ledger record**, unbounded. Read only when exact wording or raw output is needed.

Cold start reads the spine and a couple of pinned chapters. Call it 800 tokens for a session that knows what it knows.

Every spine entry has to be useful *without* expanding it. A bare ID tells the model nothing about whether to spend a retrieval on it. Twenty tokens of gist buys the entire cascade. And the addresses are IDs, never line numbers, because line numbers rot the first time somebody edits a file.

This idea has a name in the literature. A-MEM built the same thing from the Zettelkasten method: structured notes with keywords and tags, linked against the existing corpus as they're written [8]. They report up to 6× on multi-hop reasoning and 85–93% fewer tokens spent on memory operations. Notably, that's a peer-reviewed number rather than a vendor number, and the linking step appears to be doing real work rather than

decorating.

3.4 Forgetting on purpose

Here's a thing you know: Paris is the capital of France. Here's a thing you don't know: exactly when and where you learned it.

Now think about something you picked up this week from a news segment. You've still got the detail, the source, roughly when you saw it. Give it two years. The detail goes first, then the source, and there's a decent chance the whole item goes with them.

Tulving called this the split between episodic and semantic memory [13], and the direction of travel is the design. Episodic records are rich, sourced, timestamped, and fragile. Semantic facts are sparse, timeless, lack source, and durable. The thing that gets dropped first on the way from one to the other is the origin.

Which is exactly the part it can't afford to lose. Source amnesia in a person is a harmless quirk. In an agent it's the mechanism by which a poisoned tool output becomes standing policy three sessions later. Brains discard citations because storing them costs something. In software it costs twenty bytes.

So the rule is: strip the content, keep the address. A session's worth of debugging collapses down to "the plugin loader needs two levels of directory nesting," and that one line still carries a pointer to the ledger entry where it found out.

Second break with biology, and this one is pure upside. When a memory ages out, you evict it from the hot index. The record itself is not deleted. Disk is cheap and the archive is addressable, so the forgetting is reversible in a way a brain's isn't. Re-index and it comes back.

For deciding *what* ages out, Generative Agents already published the scoring function: recency plus importance, relevance, each normalized, summed [5]. The model gets a fourth signal it didn't have. Because the system logs which recalled memories the model actually used, the system can observe importance instead of asking a model to guess it. A memory that keeps getting pulled and used is load-bearing. One that gets retrieved and ignored every time isn't.

That maps cleanly onto the spaced repetition literature. FSRS models a memory with three numbers: difficulty, stability, and retrievability, where retrievability is the odds you can recall the item right now and it decays with time since the last review [12]. Use retrievability as the eviction score and the whole thing is maybe fifteen lines. The "review" event is free — a memory that was retrieved and used is a successful review. Take the shape, though, not the parameters. FSRS fits about twenty of them to 700 million human flashcard reviews, and porting those numbers to agent memory would be cargo cult.

Worth flagging that this is thin ice. The survey's blunt assessment is that nobody evaluates forgetting well, that most systems handle it with hard expiration or storage limits or nothing at all, and that of the four major benchmarks only MemoryAgentBench tests selective forgetting at all — where most systems fail conspicuously [1]. So this section describes the least validated part of the design.

One simplification is worth arguing for. It's tempting to build this as two separate stores, a hot one for recent work and a cold one for distilled knowledge. Don't. One store with a stage column and a retention score does the same job, and two physical indexes is two things to keep in sync for a corpus belonging to one person on one machine.

3.5 Governance

Human memory strips the source and keeps the fact. You know Paris is the capital of France. You have no idea what afternoon you learned it.

Copy that and you've built a security hole. An agent reads a poisoned file, writes down what it read, and three sessions later the poison comes back as trusted standing instruction. Memory turns into a privilege escalation path that crosses sessions.

So this design breaks with biology on purpose: **strip the content, keep the address**. Brains drop citations because storing them is expensive. A citation costs twenty bytes.

The rest of the trust model comes from OB1's agent memory schema [11], which had it right first. Generated memory is usable as evidence by default and as instruction never, until a human confirms it. Memories carry a lifecycle — active, stale, superseded, disputed — plus provenance and a confidence score. Nothing gets deleted; things get superseded, with a pointer to whatever replaced them.

A 2026 governance paper puts formal weight behind this [9]. Its central claim is that errors in evolving memory are cumulative and persistent, unlike static retrieval where a mistake stays inside one query. Worse, drift has direction. Their example: a mild user preference, rewritten a few times, gets progressively intensified until it causes an actual violation. The paper's answer is to decouple the agent's generative policy from the memory store with a governance layer that runs consistency checks and decay modeling before anything gets consolidated.

For a coding agent, the sharpest version of this is procedural drift. "We always fix plugin problems by doing X" hardens into law after two coincidences. The survey names the same failure from the other side: an agent that concludes "approach A always fails" will never test approach A again [1]. So gotcha chapters get a confidence and an expiry, and they never get promoted to instruction grade on their own. One bad night of debugging shouldn't become permanent doctrine.

4. Four ways in

Most memory papers talk about retrieval as if it were one thing. It's four, and conflating them caused an error in this project's own earlier design work.

Pointer dereference. Expand an ID, get the record. No search at all. Nearly free, perfectly precise, and it requires already knowing the ID.

Graph traversal. Follow edges from a known node, forward or backward. Also nearly free — it's a join. No embeddings, no model call.

Similarity search. Rank a pool of candidates against a query. Needs an index, and needs a decent query.

Structural read. Read the spine top to bottom at cold start. That's orientation, not retrieval.

Which mode a question needs depends entirely on the question. "What do we know about X" wants similarity. "Why did that crash" and "did we already try this" want the graph, walked backward. "Give me the exact error text" wants the pointer. "What is this project" wants the spine.

RAPTOR's result matters here and it's easy to overapply [6]. They built trees by recursively clustering and summarizing text, then compared descending the tree against flattening it and searching every node at once. Flattening won, consistently. Traversal loses because level membership forces the same ratio of abstract to granular material no matter what you asked.

But that finding is about abstraction hierarchies searched by similarity. Both strategies RAPTOR compared rank by cosine distance; the only difference is which nodes get scored. A causal graph isn't that structure at all. You don't descend it by abstraction level, you follow edges, and similarity never enters the operation. So the finding applies to the textbook and not to the session sketch, and a round of design work went into getting that wrong.

Graph traversal deserves more weight than it first received. The survey complains that an agent's immediate input often makes a poor retrieval query: someone asking "why did that crash" needs a log from two sessions back, not the most semantically similar sentence [1]. That's not a bad query. It's the wrong tool, and no amount of query rewriting fixes a mode mismatch. Edge walking is the right tool, it costs nothing, and it needs no capability from the model at all.

There's convergence here worth pointing at. Tencent's node-ID canvas [10], HippoRAG's knowledge graph with Personalized PageRank [7], A-MEM's Zettelkasten links [8], and the survey's proposal to annotate a causal parent at write time [1] are all the same idea: graph structure as the index. RAPTOR is the other idea: abstraction hierarchy as the index. Both are real and they answer different questions.

The causal graph comes almost for free, incidentally. The ledger is causal by construction, *because* this tool call happened because that one failed. For an agent whose job includes root cause analysis, that may be the cheapest good idea in the whole literature.

5. Why local changes the math

Two constraints run opposite directions, and both come from running on your own machine.

The budget is brutal. A frontier design that spends 20% of a 200,000-token window on a memory canvas is spending 40,000 tokens. The same 20% of a 4,000-token usable window is 800. Every threshold in every published design has to be re-costed, not copied. Raw conversation never fits in context. It lives on disk, always, no exceptions.

Slow disk is an asset. This is the part that was surprising.

Production memory systems budget 200–500ms for retrieval [1]. Tencent's plugin caps recall at five results with a five-second timeout and skips injection rather than stall the turn [10]. Those are sensible decisions for a chat product where somebody is watching a cursor blink.

This is not chat. Ten seconds is a long time to a CPU and barely noticeable to a person waiting on an agent that's doing real work. That gap buys things nobody serving a chat UI can afford: multi-hop retrieval that reads a document and follows its links to the next one, reading whole files instead of chunks, model-in-the-loop re-ranking that pulls fifty candidates and injects three, checking a memory against current source before trusting it.

It also buys the RAPTOR result. Flattening the tree and searching every node is expensive at scale, which is why hierarchical systems traverse. Over one user's corpus with ten seconds to spend, searching everything is trivial. That's the first case found where the latency

asymmetry buys a *better answer* rather than just a cheaper one.

Where you put the memory matters as much as how much of it there is. Injecting recalled memory means rewriting the front of the prompt every turn. Hosted providers cache identical opening tokens and bill them at a discount, so a prompt whose head changes every turn quietly gives back part of whatever the memory layer saved. Tencent has this open as a live issue against their own plugin [10].

Locally it's worse, and in a different currency. Inference engines cache the KV state for a stable prefix. Change the opening tokens and you don't just pay more, you recompute the prompt, every turn, and the user feels it. So the layout isn't cosmetic:

- **Stable head**: system instructions and the persona block. Changes rarely, stays cached.
- **Semi-stable middle**: the current project or task scene.
- **Volatile tail**: the session sketch and anything recalled this turn. Always last.

Which gives a hard rule for the consolidation module: the persona layer gets rebuilt at session boundaries only, never mid-session, no matter how interesting the new information is.

One caution, though. Generous isn't the same as universal. Forty turns at ten seconds each is seven minutes of pure waiting, most of it wasted on turns that needed nothing. So the spend gets tiered: most turns pay nothing and answer from what's already there, some pay under a second for an index lookup, and a few pay the full ten for cold start or when the agent is genuinely stuck. The survey calls this dynamic routing and it appears this process reinvented it [1].

6. How to know

The design is worthless without a way to kill it. Six checks, cheapest first.

Before any of them can run, though, a corpus is needed. Instrument the harness to write the ledger, work normally for a week, and *then* start measuring. Nothing here can be tested against sessions that were never recorded.

Degradation curve. Where does this model start rotting? Bury one fact in filler, sweep the filler from 500 to 8,000 tokens, measure accuracy. This isn't pass/fail. It's calibration, and until it runs, every token budget in this paper is a guess.

Can a small model use pointers? Give the model a raw truncated log, a pointer sketch, and a prose summary at matched token counts, then ask twenty questions about the session. Then the harder half: ask ten questions the gist can't answer, and see whether the model expands the pointer or just confidently answers wrong. Confidently wrong is the dangerous outcome, worse than useless. If that's what happens, pointers are actively harmful with this model and the architecture has to change.

Are embeddings needed at all? Compare keyword search, vector search, and hybrid over the same store. The guess is that keyword wins inside a single session, where the vocabulary is small and shared, and loses across sessions. A split result would be fine.

What does consolidation cost? Extraction needs a model call. Measure total wall-clock with and without it, extraction included. If latency regresses badly, push consolidation further off the critical path.

Cold start. The one that tests the actual promise. Run a session that reaches real decisions, consolidate it, open a *fresh* session with only the spine, and ask fifteen questions about yesterday. Watch for invention — a fresh session that confidently makes up an answer where the spine is silent has failed worse than one that says it doesn't know.

Is ten seconds worth it? Compare fast keyword lookup against the full multi-hop-plus-rerank pipeline. If slow doesn't beat fast on quality, the latency advantage this design leans on is imaginary and the simple version is the one to build.

Existing benchmarks cover some of this better than anything written from scratch here. MemoryAgentBench is the only one that tests selective forgetting seriously [1], and MemoryArena has the right shape for measuring memory that drives decisions [15].

7. What would make the project stop

Written down now, while currently not invested.

If the model rots before 2,000 tokens and can't use a compressed sketch, it can't hold anything useful and the answer is a different model, not a memory layer.

If consolidation costs more wall-clock than the session saves, and moving it off the critical path doesn't fix that.

If the baseline doesn't fail. Run the turn-four/turn-forty test without any memory layer first. If the agent handles it fine already, the problem doesn't exist at this scale and the honest move is to walk away.

And if the model answers from gist instead of expanding, and pre-expanding everything blows the budget, then

pointers are the wrong primitive and it will require building something else.

8. Where this stands

Nothing is yet built and finalized. The design above is assembled from published work, a vendor implementation whose benchmark numbers nobody outside the vendor has reproduced [10], and one repository whose trust model believed to be better than its storage layer [11]. Every number quoted from that vendor should be read as a hypothesis.

What this study is reasonably confident about is the shape. Keep a sketch in the prompt. Keep the evidence on disk. Keep the path between them working. Let the memory decide what to forget, but never destroy the address of what it forgot.

The rest is measurement, and measurement needs a week of recorded sessions that have not been collected yet.

References

[1] Du, P. (2026). *Memory for Autonomous LLM Agents: Mechanisms, Evaluation, and Emerging Frontiers.* arXiv:2603.07670.

[2] Chroma Research (2025). *Context Rot: How Increasing Input Tokens Impacts LLM Performance.* https://www.trychroma.com/research/context-rot

[3] Packer, C., et al. (2023). *MemGPT: Towards LLMs as Operating Systems.* arXiv:2310.08560.

[4] Letta (2026). *Sleep-time Compute.* https://www.letta.com/blog/sleep-time-compute/

[5] Park, J. S., et al. (2023). *Generative Agents: Interactive Simulacra of Human Behavior.* arXiv:2304.03442. UIST '23.

[6] Sarthi, P., Abdullah, S., Tuli, A., Khanna, S., Goldie, A., & Manning, C. D. (2024). *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval.* arXiv:2401.18059. ICLR 2024.

[7] Jiménez Gutiérrez, B., et al. (2024). *HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models.* NeurIPS 2024.

[8] Xu, W., et al. (2025). *A-MEM: Agentic Memory for LLM Agents.* arXiv:2502.12110. NeurIPS 2025.

[9] Lam, C., Li, J., Zhang, L., & Zhao, K. (2026). *Governing Evolving Memory in LLM Agents: Risks, Mechanisms, and the Stability and Safety Governed Memory (SSGM) Framework.* arXiv:2603.11768.

[10] TencentCloud (2026). *TencentDB Agent Memory.* https://github.com/TencentCloud/TencentDB-Agent-Memory (MIT license). Benchmark figures cited here are vendor-reported and have not been independently reproduced.

[11] Jones, N. B., et al. *OB1 / Open Brain.* Agent memory schema and API. https://github.com/NateBJones-Projects/OB1

[12] Anki / FSRS contributors. *Free Spaced Repetition Scheduler.* https://faqs.ankiweb.net/what-spaced-repetition-algorithm — the Difficulty/Stability/Retrievability model used as the shape of the decay function.

[13] Tulving, E. (1972). Episodic and semantic memory. In *Organization of Memory*, Academic Press.

[14] Ebbinghaus, H. (1885). *Über das Gedächtnis.* The forgetting curve.

[15] He, et al. (2026). *MemoryArena: Benchmarking Agent Memory in Interdependent Multi-Session Agentic Tasks.* Cited via [1].

[16] Cloud Codes (2026). *China Just Open-Sourced Humanlike Memory for AI Agents (Tencent DB).* https://youtu.be/5AkurBDSYwo — the framing example in §1 is drawn from this video.

[17] Maharana, A., et al. (2024). *Evaluating Very Long-Term Conversational Memory of LLM Agents* (LoCoMo). Cited via [1].

[18] Wang, G., et al. (2023). *Voyager: An Open-Ended Embodied Agent with Large Language Models.* Cited via [1].