

Chronomem (忆时塔): Time-Layered External Memory as an MCP Service for Agent Recall and Summarization

YSTYLE¹ • ORCID: 0009-0009-6929-0070 • lxy5266@live.com

DeepSeek V4.1 Flash² • AI co-author

¹ Position / design paper — no empirical evaluation yet. Implementation and full design spec: <https://atomgit.com/ystyle/chronomem> (the memory service plus `dsh-mem-bridge/`, the host plugin that records agent-session activity automatically — the source of the activity corpus in §6). ² AI co-author, as credited on the AiraXiv preprint. See the AI-use disclosure at the end.

Abstract

Long-horizon agents need to remember what happened over weeks and months, yet today's external memory systems either store flat recent history or perform destructive one-shot compaction that discards the detail needed for evidence and reasoning. We present **Chronomem** (忆时塔, "Tower of Remembering Time"), a time-layered external memory service exposed through the Model Context Protocol (MCP). Inspired by two everyday mechanisms that roll detail up layer by layer—financial consolidation and the daily-report chain (task → daily → weekly → monthly)—Chronomem organizes episodic records into a hierarchy of time windows—from ten minutes up to a week—where each level is a materialized snapshot that references finer levels and can be drilled down to original records. Its central design principle is the separation of two kinds of recall with different costs: **exact recall**, which navigates reference chains to raw records and costs no LLM tokens, and **semantic recall**, which reads LLM-generated summaries and is the form that truly compresses coarse windows. Summary records are themselves append-only temporal entries, so summaries can be recursively summarized, realizing progressive, non-destructive compression. Memory types are user-defined with per-type granularity chains, so high-frequency channels (chat, tool calls) keep minute-level granularity while sparse channels (work logs) converge to day/week. We position the design, discuss open problems (semantic drift, token-budgeted trigger strategies, privacy), and outline an evaluation plan; empirical results are forthcoming.

1. Introduction

Agents operating over long horizons—iterating on a codebase, writing a thesis, maintaining a project over months—face a fundamental limitation: their context window is bounded, while the events worth remembering are unbounded. Existing approaches fall into two camps with opposite failure modes:

- **Flat external memory** (store everything, retrieve by recency or similarity) preserves detail but provides no compression: after months, no level of the system tells you "what happened, in summary," short of reading everything.
- **One-shot compaction** (periodically compress the whole conversation into a single report) destroys the evidence: decisions are remembered, but *why* and *how* — the raw events — are gone, and re-deriving them is impossible.

Chronomem is designed to occupy the middle ground, which we call **time-layered memory**: organize episodic records into a materialized hierarchy of time windows, where each layer is both (a) a *navigation* structure referencing finer layers down to the raw records, and (b) optionally a *semantic* summary readable at a glance. This mirrors two mundane mechanisms that already work this way: the *daily-report chain* in ordinary work (task records roll up into daily, weekly and monthly reports, each level terser than the one below) and *financial consolidation* (a raw ledger is never deleted, statements are recomputed progressively, and each statement can be audited back to its entries).

We expose Chronomem as an **MCP (Model Context Protocol) service**—a single daemon that any MCP-capable client (opcode, Cherry Studio, custom agents) can attach to. Memory as a *protocol* rather than a private library is, we argue, a systems contribution in itself: it lets heterogeneous tools share one memory substrate without code coupling.

This paper is a **position and design** paper. We motivate the design from two everyday roll-up practices (financial consolidation and daily/weekly/monthly reporting), relate it to concurrent work in LLM memory systems, state the dual-track cost model that distinguishes our approach, and lay out the open problems we intend to address empirically.

2. Background and Related Work

Layered roll-up in everyday practice. Two mundane mechanisms work by rolling detail up layer by layer, and both were the direct inspiration for this design: the *daily-report chain* (task records → daily → weekly → monthly reports, each level terser than the one below) and *financial consolidation* (a raw ledger → periodic statements, where the ledger is never deleted, statements are recomputed progressively, and every statement can be audited back to its entries). What they share—and what we borrow—is that the detail is retained in full while each level still answers "what happened at this scale" at a glance. Chronomem engineers that mechanism into agent memory: fine-grained records are kept, and each coarser layer is simultaneously a reference chain into finer layers and a readable semantic summary. The resulting structure also resonates with the intuition behind memory consolidation in cognitive science (consolidation; sleep-based replay), though we make no claims about biological fidelity. What transfers from reporting practice—drawn from the authors' experience building ERP and reporting systems—is deliberately the *discipline* rather than the aggregation semantics: an immutable detail layer, auditable drill-down, and idempotent progressive recomputation. Unlike ERP consolidation, where the roll-up is a deterministic sum over structured entries on a legally fixed calendar, the layers here must summarize arbitrary JSON events and natural-language records with an LLM, the granularity chain is configurable per memory type, and the cost asymmetry between navigation and compression becomes a first-class design concern.

Time-series databases and rollups. Layered time windows with materialized aggregates are standard in time-series databases (rollups / continuous aggregation / downsampling, e.g., TimescaleDB continuous aggregates, Prometheus downsampling). Chronomem reuses this machinery (built on the `tsdb` library) but extends it in a direction databases do not: the coarse layers are not only numeric aggregates but **LLM-generated semantic summaries** of arbitrary JSON events.

LLM memory systems. A rapidly growing line of work addresses long-term memory for LLM agents: MemGPT [Packer et al.] introduces hierarchical context management and a memory hierarchy; Generative Agents [Park et al.] couple a memory stream with retrieval and reflection; A-MEM [arXiv:2502.12110] and Mem0 [arXiv:2504.19413] develop agentic/organized memory; **HiMem** [arXiv:2601.06377] proposes hierarchical long-term memory for long-

horizon agents; **RaMem** [arXiv:2606.22844] studies contextual reinstatement; **AriadneMem** [arXiv:2603.03290] threads lifelong memory for agents. Our work differs in emphasis: (i) we separate *exact* (zero-token, reference-chain) recall from *semantic* (LLM, compressed) recall as an explicit cost model; (ii) summaries are first-class append-only temporal entries enabling recursive summarization; and (iii) we standardize the whole substrate behind an MCP interface.

Thematic vs. temporal memory. "LLM wiki" systems organize knowledge by topic/semantic entry—thematic memory, analogous to a Wikipedia. The pattern—compiling sources into a persistent, LLM-maintained wiki instead of retrieving raw chunks at query time—was proposed by Karpathy [Karpathy, *LLM Wiki*, 2026] and has since drawn design and production analyses (*Beyond RAG: LLM Wikis as Living Semantic Memory* [Zenodo 20078453]; *Compiled Memory vs Governed State: Karpathy's LLM Wiki and the Reflexive Laboratory* [Zenodo 19507593]; *Managed Memory Is Still Not Governed State: Production Lessons from LLM Wiki v2* [Zenodo 19545947]). Chronomem organizes by time—temporal memory, analogous to a ledger. They are complementary: a time log can precipitate thematic entries, and a thematic entry can trace back to time evidence. We make this contrast explicit because we believe many "memory" systems conflate the two axes, and the choice of axis should follow the use case (a thesis is a process—temporal; a reference manual is a body of knowledge—thematic).

Model Context Protocol. MCP (modelcontextprotocol.io) standardizes tools, resources, and prompts between clients and servers. Chronomem's contribution here is to treat *memory* as a first-class MCP capability through a `mem/*` toolset, so any MCP client can adopt a shared, evolving memory without integrating a proprietary SDK. (Memory is currently exposed as tools only; resources and prompts are natural later extensions of the same substrate.)

3. Design Overview

Chronomem is a standalone daemon serving a single streamable-HTTP MCP endpoint (the Web console, when enabled, shares that same port, with MCP mounted at `/mcp`). Storage is a single time-series database (`tsdb`); each memory type owns two series—one for raw entries and one for its summaries. The design has five pillars.

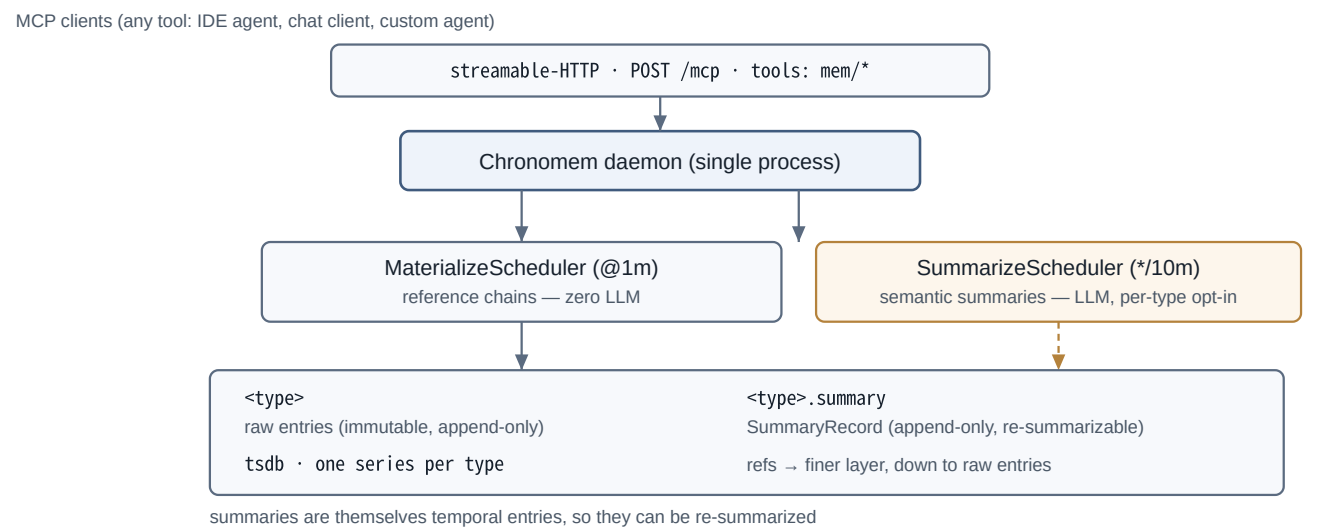


Figure 1. Architecture. A client reaches one daemon over MCP; two independent schedulers materialize a type's two series — reference chains (free, minute-level) and semantic summaries (LLM-costly, per-type opt-in). Summaries live in their own append-only series and reference finer layers, which is what makes recursive summarization possible (§ 3 P5).

P1 — Memory types are user-defined. A *type* is a named series with an optional description and schema (for documentation and LLM prompting; not enforced), a declared retention field (currently metadata only — the MVP does not evict), and a *granularity chain* (below). Clients create types such as `work/chat`, `paper/decisions`, `tool/errors`. Type names are whitelisted (`[A-Za-z0-9_-]`) to prevent path traversal.

P2 — Per-type granularity chains. Each type declares its own materialization levels, drawn from {10min, 30min, 1h, 3h, day, week, month}, defaulting to 10min → 30min → 1h → day → week. High-frequency channels keep minute-level levels for *same-day exact recall*; sparse channels (e.g., a work log with one entry a day) use day → week and avoid building mostly-empty fine buckets. Week is the default semantic-summary ceiling: a full year of weekly summaries is only ~52 records, cheap to load, and whole-year recall is rarely needed.

P3 — Materialization modes. A tower layer can in principle be materialized in three ways, with different costs. The current implementation ships the first two. The third is not implemented: the store exposes only a minimal set of aggregation primitives (count, latest, uniform sampling), and generic numeric roll-ups (avg/min/max/percentiles) would have to be added before metric-shaped types could use a zero-token overview layer.

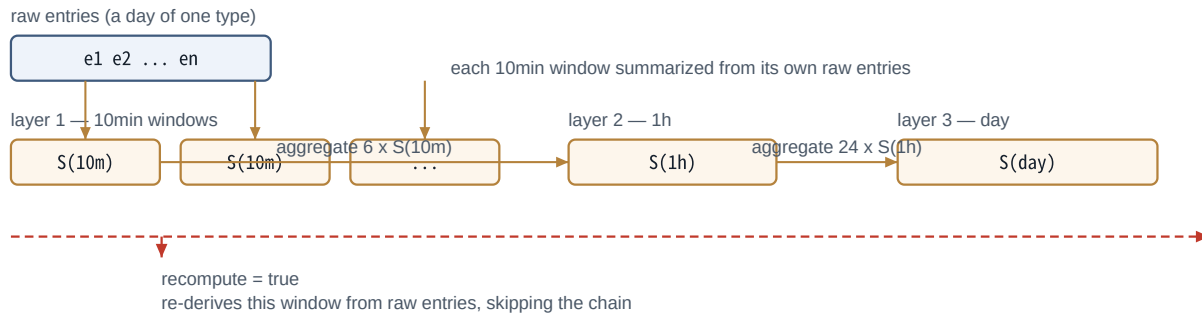
Mode	Content of a layer	Supports	LLM cost	Status
Reference chain (default, events)	PointRef / BucketRef to finer layers	exact drill-down	none	implemented
Semantic summary (optional)	LLM SummaryRecord	semantic recall	yes	implemented
Numeric aggregation (metrics)	count/avg/min/max JSON	overview + drill-down	none	planned (only count/latest/sample primitives exist today)

P4 — Dual-track recall. Two read operations with an explicit cost difference:

- `mem/recall mode=exact` : navigate the reference chain to the *raw entries*—lossless, zero LLM tokens, but returns raw volume (reference chains *navigate*, they do not *compress*).
- `mem/recall mode=summary` : return the *semantic summary* of a window—this is the form that genuinely compresses coarse windows, and it is *dependent on a summary having been generated* (previously via `mem/summarize`, or computed on demand when `summarize_on_miss=true`).

P5 — Summaries are first-class temporal records. `mem/summarize` reads a window (by granularity+anchor, or by explicit `from / to` for irregular "task-phase" windows), asks an LLM to produce a structured summary (anchors: decisions / conclusions / todos / state changes / keywords), and appends a `SummaryRecord` to the type's `.summary` series: the core fields are {`ts`, `windowFrom`, `windowTo`, `model`, `refs`, `instructions`, `text`, `groupKey`}, plus generation-provenance fields {`strategy`, `chunks`, `inputTokens`, `clippedEntries`} that record whether the summary was produced in one call or by chunked map-reduce, and how many entries were clipped for budget. Because summaries are themselves temporal entries, they can be re-summarized: raw → 10min summary → 1h summary → day summary is the semantic analogue of the structural tower, giving **recursive, non-destructive compression**. A `recompute` flag forces re-derivation from raw entries. Concretely, summarization proceeds *bottom-up by default*: before summarizing a coarse window the service makes sure every finer sub-window inside it has a summary—generating the missing ones from raw entries first—and then aggregates those summaries, so a coarse summary costs

one call per group instead of a re-read of the whole window, while `refs` records which layer the material actually came from. A caller who wants to avoid chained paraphrase altogether can bypass the recursion with `recompute`.



Cost: one LLM call per group per layer — a coarse layer never re-reads raw entries.

Provenance: `refs` records which layer the material actually came from.

Drift: chained paraphrase is cheap but drifts; `recompute` is faithful but costs tokens — the caller chooses.

Figure 2. Bottom-up summary recursion (§ 3 P5). Each layer is summarized from the layer beneath it, and a missing finer layer is generated on demand; `recompute` bypasses the chain and rebuilds a window from raw entries. Every summary is itself append-only, so nothing in the chain is destroyed.

4. The Dual-Track Cost Model

Our central conceptual claim is that *recall* must not be treated as a single operation. We distinguish:

- **Exact recall** = zero-token navigation. It is reliable and lossless, but its information content is the raw volume; a coarse-layer exact recall returns everything, so it provides no compression benefit. This is *navigation*, not *summary*.
- **Semantic recall** = reading a summary. It is the only form that truly satisfies "the coarser the window, the smaller the answer." But it has a strict *precondition*: the summary must exist (generated earlier, or generated now at LLM cost). Hence "recall is zero-LLM" holds **only for exact recall**; coarse-grained overview recall is inherently summary-dependent.

This distinction, trivial in retrospect, has concrete consequences that we argue prior systems blur:

1. **Trigger strategy.** Structural materialization (reference chains, aggregates) is a minute-level job that costs no tokens. Semantic summarization is the opposite: an ungarded scheduler would silently burn tokens on empty or low-value windows, and would keep retrying a window that cannot succeed. Chronomem therefore makes it **per-type opt-in** (new types default to off) and, once enabled, bounds it with five mechanisms: (a) only the most recent *closed* window is summarized, after a settle delay, so a window is never summarized while still being written; (b) summarization is idempotent per window and per group, and a successful run clears prior retry state; (c) empty windows cost nothing and are skipped; (d) repeated failures back off and, after a bounded number of attempts, are recorded as abandoned instead of being retried forever every ten minutes; and (e) each scheduler run is capped, so a backlog cannot become an unbounded burst. The built-in `worklog` type ships with this enabled—one summary per closed day per project—while arbitrary new types default to off, and the semantic-summary "miss" path on recall remains explicit (`summarize_on_miss`).
2. **Semantic drift.** Recursive summarization of summaries is cheap but drifts; re-deriving from raw is faithful but expensive. Exposing both, with an explicit `recompute`, turns a silent quality problem into a caller-controlled trade-off.

3. **Context-hygiene.** The service only stores and retrieves; it never pushes content back into a client's context. Clients decide whether and how much recall to inject, preserving a constant-context discipline.

5. Open Problems

- **Semantic drift quantification.** How quickly does a chain of summaries diverge from the underlying events, and when is `recompute` worth its tokens? We plan to measure information retention along the chain with a fixed set of probe questions.
- **Token-budgeted triggering.** Given a token budget, when should summaries be computed proactively (e.g., end-of-hour/end-of-week) versus lazily on recall? This is an optimization problem over {window, value, cost}.
- **Retention and privacy.** MVP stores payloads in full with no privacy tiers; type-level privacy/encryption is future work, motivated by the observation that memory becomes *more* sensitive as it grows.
- **Evaluation methodology.** Agent-memory evaluation is immature. We propose: fixed long-horizon task suites, query sets with ground-truth answers, and metrics {recall precision/recall, token cost, latency} against baselines {flat recency memory, one-shot compaction, a MemGPT-style memory hierarchy}. One lesson from operating our own deployment is that **materialized-view consistency deserves to be a metric rather than an assumption**: a layered store can return a stale window (entries written after that window's last materialization are simply missing) or a neighboring window's records altogether, while the returned evidence looks entirely self-consistent and carries no error signal. Evaluations of hierarchical memory should therefore include an explicit consistency check between each coarse view and the raw records it claims to cover.

6. Evaluation Plan (Forthcoming)

Empirical results are out of scope for this position paper and will appear in a follow-up. Planned studies: (i) a controlled long-horizon agent task (e.g., incremental project maintenance across simulated weeks) comparing Chronomem against flat memory and one-shot compaction on recall accuracy and token cost; (ii) a semantic-drift study along the summary chain; (iii) a qualitative study of the work-log use case (day/week granularity with task-phase summaries) by the authors. The authors have been running a single-user deployment continuously since August 2026; as of October 2026 it holds over 450 structured work-log records, over 1,000 agent-activity records, and more than 100 recursively generated day- and week-level summaries; this corpus will serve as the material for (iii) and will supply realistic window-size and compression-ratio distributions for calibrating (i) and (ii).

7. Conclusion

We presented Chronomem, a time-layered external memory service over MCP, whose contributions are: (1) a dual-track recall model that separates zero-token exact recall from summary-dependent semantic recall; (2) summaries as first-class append-only temporal records enabling recursive, non-destructive compression; (3) per-type granularity chains adapting to the frequency distribution of each memory channel; and (4) memory as a standardized MCP capability shared across tools. The idea is neither a claim of biological fidelity nor of algorithmic novelty in any single component—each is borrowed from databases, everyday reporting practice, or LLM-memory systems—but the *combination*, and the honest cost accounting between navigation and compression, is the contribution we put forward for discussion and future empirical validation.

References

1. Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., Gonzalez, J. E. *MemGPT: Towards LLMs as Operating Systems*. arXiv:2310.08560, 2023.
2. Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., Bernstein, M. S. *Generative Agents: Interactive Simulacra of Human Behavior*. UIST 2023 / arXiv:2304.03442, 2023.
3. Zhang, N., Yang, X., Tan, Z., Deng, W., Wang, W. *HiMem: Hierarchical Long-Term Memory for LLM Long-Horizon Agents*. arXiv:2601.06377, 2026.
4. Yang, W., Kan, B., Li, S., Li, L., Qin, Y., Li, J., Bogdan, P., Thomason, J. *RaMem: Contextual Reinstatement for Long-term Agentic Memory*. arXiv:2606.22844, 2026.
5. Zhu, W., Chen, X., Wang, Z., Wang, J., Dong, X., Huang, M., Cai, R., Sang, H., Wang, H., Qiu, P., Deng, Y., Tiwari, P., Rappazzo, B. H., Wang, Y. *AriadneMem: Threading the Maze of Lifelong Memory for LLM Agents*. arXiv:2603.03290, 2026.
6. Zhang, B. *Beyond RAG: LLM Wikis as Living Semantic Memory: Patterns, Empirical Findings, and Open Problems*. Zenodo, 2026. DOI: 10.5281/zenodo.20078453.
7. Miteski, S. *Memory as Metabolism: A Design for Companion Knowledge Systems*. arXiv:2604.12034, 2026 (concept DOI: 10.5281/zenodo.19501651).
8. Model Context Protocol specification. <https://modelcontextprotocol.io>
9. Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., Zhang, Y. *A-MEM: Agentic Memory for LLM Agents*. NeurIPS 2025 / arXiv:2502.12110.
10. Chhikara, P., Khant, D., Aryan, S., Singh, T., Yadav, D. *Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory*. arXiv:2504.19413, 2025.
11. Letta (formerly MemGPT): stateful agents with long-term memory. <https://github.com/letta-ai/letta>
12. Karpathy, A. *LLM Wiki: A pattern for building personal knowledge bases using LLMs*. GitHub Gist, April 2026. <https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f>
13. Bell, P. *Compiled Memory vs Governed State: Karpathy's LLM Wiki and the Reflexive Laboratory*. Zenodo, 2026. DOI: 10.5281/zenodo.19507593.
14. Bell, P. *Managed Memory Is Still Not Governed State: Production Lessons from LLM Wiki v2*. Zenodo, 2026. DOI: 10.5281/zenodo.19545947.

AI-Use Disclosure

This paper was written in collaboration between a human author and an AI system. **DeepSeek V4.1 Flash** is credited as an AI co-author, in line with the AiraXiv preprint platform's policy of admitting AI scientists as authors; the system contributed substantially to the drafting, structuring, and revision of the text, and to the implementation described in §3.

The human author directed the work, performed and verified the operational measurements reported in §6, checked every factual claim and bibliographic reference, and takes full responsibility for the content. Where this paper describes behaviour observed in a live deployment, those observations come from the human author's own system.

For venues that do not admit AI authors (e.g. arXiv), the AI contribution should instead be recorded as a use disclosure of this kind rather than as authorship.