MeaningSpace: Keep Every Word, Load Only the Meaning

A lossless conversational memory with line-level addresses, threshold-gated skills, and a time-aware read layer — a three-year single-builder case study

Koji Okuda (SubtractLab, independent) · ORCID: 0009-0003-9251-8455 · https://subtractlab.com Draft v0.2 — 2026-11-20 — not peer reviewed

---

Abstract

Assistants that work with a person every day forget everything between sessions. The common remedies either summarize the history and discard the original, or reload and search the raw history on every question. We describe **MeaningSpace**, a local memory layer that takes a third path: the full conversation log is kept as an immutable, line-numbered *ground*, and a *meaning layer* of sections ("crystals") is placed on top of it. Every section records its outcome (*decided / overturned / open*) and the addresses of the log lines it was distilled from, so a new session loads only a few sections and can still descend to the exact original lines. Working procedures ("skills") live in the same layer and surface only when their similarity to the task clears a fixed threshold, so no skill catalog is ever placed in the prompt. The system has been in daily use by one builder since 2023; it now holds 4,722 conversations (1,567 imported from ChatGPT, the rest with Claude) and 35,789 sections, plus 78 skills. We report three measurements taken in November 2026 on the author's personal computer: (1) moving the read path from a single DuckDB file to a Parquet read layer reduced the median latency of one retrieval from 1,551 ms to 457 ms with identical top-5 results on 30 of 30 queries, shrank the data needed for retrieval from 2.3 GB to 157 MB, and allowed 5 concurrent readers to succeed on 50 of 50 reads while a writer held the database (0 of 50 before); (2) adding dates, a one-line summary and the newest same-topic conversation to each retrieval result raised the accuracy of memory-less agents on time-sensitive questions from 3/11 to 8/11 (Sonnet 5.5) and from 4/11 to 9/11 (Opus 5.5); (3) a second AI agent, treated as a second user, read the memory and edited code through the same host and passed the same procedure three times in a row. The evaluations are small and single-user; we state their limits explicitly.

---

1. Introduction

A person who works with an AI assistant all day accumulates hundreds of hours of shared work: designs that were decided, fixes that failed, decisions that were later reversed. Each new session starts from zero. Two families of remedies dominate.

- **Summarize and discard.** Facts or notes are extracted and the original dialogue is dropped or expires. The memory fits in context, but the reasoning that led to a decision is gone, and when understanding changes there is nothing to return to.
- **Reload or search everything.** Nothing is lost, but every question pays for a large amount of reading, and lexical search often lands in the wrong file. Long contexts are known to be read unevenly [Liu et al. 2024].

MeaningSpace keeps the ground and carries the meaning. Its design principle is subtractive: *what* is loaded at session start is reduced to almost nothing, while *nothing* is deleted from storage. The contributions of this report are:

1. A memory model in which every distilled section carries an outcome label and **line-level addresses** into an immutable log, with addresses *selected from a closed candidate set* rather than recalled by the model (§3).
2. **Threshold-gated skills**: procedures stored as crystals that are returned — title and a query only — when similarity to the task clears 0.80, with no catalog in the prompt (§3.4).
3. A **time-aware retrieval result** that gives the reading agent dates, a one-line summary per conversation and the newest conversation on the same topic, and a small evaluation of its effect (§5.2).
4. A **single-writer, many-reader** storage layout using a nightly Parquet copy, with latency, size and concurrency measurements (§5.1), and a first test of multiple agents as users (§5.3).
5. A dated record of a system used daily for three years (§4, Appendix A).

2. Related work

Hierarchical and agent memory. MemGPT treats the context window like virtual memory and pages information in and out [Packer et al. 2023]. Generative Agents keep a memory stream with reflection [Park et al. 2023]. RAPTOR builds a tree of recursive cluster summaries over static documents for retrieval [Sarthi et al. 2024]. MeaningSpace shares the idea of layered summaries but keeps the raw dialogue as the ground of record and links every summary section to its source lines.

Episode-preserving memory. Graphiti/Zep keeps raw input as episodes and traces facts in a temporal knowledge graph back to them [Rasmussen et al. 2025]. TencentDB

Agent Memory (open-sourced 2026-04-07) arranges raw logs into four layers and lets the agent descend to the original by node id. These systems arrived independently at the same refusal to discard the original. MeaningSpace differs in what sits on top: narrative sections with outcomes and behavior-changing skills, rather than an entity graph.

Skills and procedures. Progressive disclosure keeps a catalog of skill names and descriptions in context and loads bodies on demand [agentskills.io]. Threshold skills remove the catalog itself (§3.4).

Retrieval. We use BM25 [Robertson & Zaragoza 2009] and dense vectors fused by rank [Cormack et al. 2009], with an optional cross-encoder reranker. Storage is DuckDB [Raasveldt & Mühleisen 2019] and Apache Parquet.

3. Design

3.1 Ground: the immutable log

Every conversation is stored as a cleaned, line-numbered text log. Logs are never edited or deleted. ChatGPT conversations from 2023-02-07 onward were imported in the same form, so the oldest record in the system is the first conversation with an AI.

3.2 Crystals: sections with outcomes and addresses

After a conversation, a crystallization pass divides it into sections. Each section has a title, a few sentences of meaning, an **outcome** — *decided*, *overturned* or *open* — and the **addresses** of the log lines it came from. The model is not asked to remember line numbers; candidate addresses are computed from the log, and the model selects among them. A section is therefore a pointer as much as a summary: when a summary is in doubt, the reader descends to the lines.

An *overturned* section is kept, not merged away. When a later conversation reverses a decision, both are retrievable, and the reversal is visible as such.

3.3 Retrieval

A query is answered by lexical search (BM25) and vector search over sections, fused by rank and optionally re-ranked. The result lists conversations and their matching sections, each with its outcome and line range. A query that lands on a project or task returns that box and its open sections. A query never returns whole logs; descending is a separate, explicit step.

3.4 Threshold skills

Procedures (how to publish, how to check a report, what never to touch) are stored as crystals whose first section states when to use them. Just before the agent *makes, fixes or decides* something, it issues one sentence describing the task. The sentence is compared with the whole skill layer; below a similarity of **0.80** nothing is returned. Above it, only the title and a query to read the sections are returned. Thus a result is itself the signal that a skill applies, and the number of skills does not grow the prompt. On 2026-10-24 a 25-line skill catalog was removed from the always-loaded profile after confirming that each of the 25 skills appeared within the top 3 for its own task.

3.5 Boxes and the progress map

Projects and tasks are stored as *boxes* with status and progress facets; conversation sections can be linked to a box. Files and code get the same treatment in a *progress map*: each edit is checked, removed lines are counted, and a ledger row with a reason and a reverse patch is tied to the conversation section that caused the edit.

3.6 Time-aware results

Retrieval by meaning does not know which result is newest. In practice this returned an older plan as if it were current. Each result now carries (a) the start date of its conversation, (b) a one-line summary of what the conversation was about, and (c) the newest conversation on the same topic, kept only if a reranker scores its sections as relevant to the query. The result also carries one line telling the reader how to use them: for "what is the state now" questions, prefer the newest conversation after checking that its content matches; for "where/why was this decided" questions, prefer the best semantic match.

3.7 Storage: one writer, many readers

All layers live in one DuckDB database, which allows a single query to go from meaning to section to log line to file to the reason for a change. DuckDB admits one writing process; while a long crystallization held the file, readers failed. Since 2026-11-19 the meaning layer is also exported to a folder of Parquet files every night, and each crystallization adds a small delta file. Retrieval reads the Parquet copy; the DuckDB database remains the only writer.

4. Deployment

MeaningSpace runs locally on one personal computer and is used through a chat interface; the assistant calls it as a tool. It has been used daily by one builder for software development, data analysis, document and video production. Table 1 gives its size on 2026-11-20.

Item	Count
Conversations with sections (since 2023-02-07)	4,722 (ChatGPT 1,567; Claude 3,100+; other 19)
Sections (crystals)	35,789
Skills	78
Embedding model	ruri-v3-310m, 768 dimensions

Table 1. Contents of the deployed system (counted 2026-11-20).

5. Measurements

All measurements were taken on the author's personal computer in November 2026. They are engineering measurements of one deployment, not a benchmark.

5.1 Parquet read layer

One retrieval (lexical + vector + section details) was timed with 2 threads, before and after moving the read path to Parquet.

	DuckDB file only	Parquet read layer
One retrieval, median	1,551 ms	457 ms
Top-5 agreement with the old path	—	30 / 30 queries
Data needed for retrieval	2.3 GB	157 MB
Rebuild of the read layer	—	3.6 s
5 concurrent readers while a writer holds the DB for 40 s	0 / 50 reads succeeded	50 / 50 succeeded

Table 2. Read-layer measurements. The concurrent-reader test ran the read path as five separate processes, not five full agents; vector look-ups were still served from DuckDB at that time.

Keeping the lexical scores as numbers inside DuckDB was 2× faster than reading them from Parquet (13 ms vs 30 ms with 2 threads), but the difference is about 4 % of a

retrieval, so the Parquet layout was kept for ease of distribution.

5.2 Time-aware results

We wrote time-sensitive questions whose correct answer is the *current* state of a topic (for example, the latest version of a plan that had changed several times). A fresh agent with no memory of the conversations answered each question using only retrieval results. We compared results without and with the fields of §3.6.

Reader	Questions	Before	After
Opus 5.5 (first test set)	8	2	7
Opus 5.5	11	4	9
Sonnet 5.5	11	3	8

Table 3. Correct answers on time-sensitive questions. The questions are listed in Appendix B. Runs vary: in a repeated run with the same 11 questions, Opus 5.5 answered 5 (not 4) correctly before the change and 9 after. Two misses initially attributed to retrieval failures were re-examined: one was caused by a project name in the question pulling in a different conversation, the other by the expected answer living in a planning conversation rather than the one judged correct. One remaining Sonnet error chose the older of two recent conversations; the fix was to mark the newest one explicitly rather than to reword the instruction.

5.3 Agents as users

A second AI agent, running in a separate chat session, was connected to the same host and used the same tools as the main agent to read memory and to create, read and edit a small program. Treating each agent as one user, this is a first test of multi-user operation. The procedure passed three times in a row once a fixed connection order was followed; two earlier attempts failed because the session was not linked to the host. Writes from both agents go through the single host, which orders them.

6. Limitations

- **Single user.** All data come from one builder. Multi-user operation has been tested only with AI agents as users, not with several people; read permissions and conflicting edits are not designed yet.
- **Small evaluations.** The time-aware test has 8 and 11 questions written by the system's builder. The latency figures come from one machine.

- **Not released.** The implementation is not published; the design is published as dated records (Appendix A).
- **Crystallization cost and errors.** Sections are written by a language model and can be wrong; the immutable log and the addresses are the remedy, not a guarantee.
- **Model dependence.** The reading and crystallizing agents are commercial models; results may change with model versions.

7. Conclusion

Keeping every word and loading only the meaning is practical for one person over three years: the ground is never lost, the meaning layer is small enough to load on demand, skills cost nothing until they apply, and a read layer separates many readers from one writer. The next steps are a multi-person test and a larger, independently written evaluation set.

Appendix A. Dated public record

Date	Record
2023-02-07	First conversation with an AI; kept in the ground
2025-03	Memory-first concept recorded in conversation
2026-07-16	Prior-art timeline first published at subtractlab.com
2026-09-21	Threshold skills in use
2026-10-04	Progress map in use
2026-10-07	Design records `meaningspace`, `ai-native-groupware` published on GitHub
2026-11-07	`progress-map`, `threshold-skills` published
2026-11-19	Parquet read layer; time-aware results
2026-11-20	Second agent on the same host
2026-11-20	Design records archived on Zenodo with DOIs (CC BY 4.0)

Repositories: https://github.com/subtractlab · Timeline: https://subtractlab.com/history.html

Archived design records (Zenodo, concept DOIs): MeaningSpace 10.5281/zenodo.24388284 · AI Native Groupware 10.5281/zenodo.24388287 · Progress Map 10.5281/zenodo.24388289 · Threshold Skills 10.5281/zenodo.24388291

Appendix B. Time-sensitive questions

Each question was asked "as of" a fixed time T, and the correct answer is one or more conversations chosen by the author before the test. *now* questions ask for the current state; *origin* questions ask where something was decided. Questions are paraphrased in English; topics from the author's employer are described generically. The first test set (8 questions) is C1–C5, C8, O1 and O2; the 11-question set adds N1–N3. The last column is the Sonnet 5.5 run counted in Table 3.

ID	Type	Question (paraphrased)	Sonnet 5.5 before → after
C1	now	Let's plan the script for the next video episode.	✗ → ✗
C2	now	Check the latest audience numbers of the video channel.	✗ → ✗
C3	now	Continue the reconciliation between two finance databases (asked at one time).	✗ → ✓
C4	now	The same question, asked 80 minutes earlier.	✗ → ✓
C5	now	Continue the work on the internal BI application.	✗ → ✓
C8	now	What happened to the preview of the next episode in episode 3?	✗ → ✓
O1	origin	In which conversation did we decide not to use git for the BI application?	✗ → ✗
O2	origin	Where did we decide the procedure for handing work to a Sonnet agent?	✓ → ✓
N1	now	How far along is the BI application now?	✗ → ✓
N2	now	What is the current state of a personal trading experiment?	✓ → ✓
N3	now	What happened to the document-capture app a colleague asked for?	✓ → ✓

References

- Cormack, G. V., Clarke, C. L. A., Büttcher, S. (2009). Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. SIGIR '09, 758–759. doi:10.1145/1571941.1572114
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P. (2024). Lost in the Middle: How Language Models Use Long Contexts. TACL 12, 157–173. doi:10.1162/tacl_a_00638

- Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
- Park, J. S. et al. (2023). Generative Agents: Interactive Simulacra of Human Behavior. UIST. *(not yet checked)*
- Raasveldt, M., Mühleisen, H. (2019). DuckDB: an Embeddable Analytical Database. SIGMOD '19 (demo), 1981–1984. doi:10.1145/3299869.3320212
- Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., Chalef, D. (2025). Zep: A Temporal Knowledge Graph Architecture for Agent Memory. arXiv:2501.13956.
- Robertson, S., Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in IR. *(not yet checked)*
- Sarthi, P., Abdullah, S., Tuli, A., Khanna, S., Goldie, A., Manning, C. D. (2024). RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. ICLR 2024. arXiv:2401.18059.
- Tencent (2026). TencentDB Agent Memory. https://github.com/Tencent/TencentDB-Agent-Memory
- Agent Skills specification, client implementation. https://agentskills.io/client-implementation