Week 3: Unified Memory Search — The Swarm Remembers

February 12, 2026 · by Mark Bartlett

Download this post as PDF

📚 **More on this topic:** Session-as-RAG · Best Local LLMs for RAG · Distributed Wisdom · Planning Tool

Last week I had persistent memory and document RAG, but they were separate systems. The model could remember facts I told it and search files I'd indexed. What it couldn't do was search its own conversation history.

"What did I ask about Tailscale last week?" Blank stare. The model had no idea. The conversations were stored as flat summary logs, one blob per session, no embeddings, no semantic search. Usable for "load the last 10 sessions as context" but useless for "find the thing I said about bee hive ventilation three days ago."

This week I unified them. Three releases in two days to get it right.

The problem: two memories that don't talk

Week 2's memory had two layers. Facts (explicit `/remember` commands stored in JSON) and documents (PDFs, markdown, code files chunked and embedded in ChromaDB). When you asked a question, the model could pull from your documents or from its training data. It couldn't pull from what you discussed last Tuesday.

That's a problem for real work. I'd been using mycoSwarm for InsiderLLM research all week. I had conversations about GPU pricing trends, Tailscale configuration for the WiFi laptop test, and drafting notes for the salvage hardware philosophy piece. All of that context existed in session logs on disk. None of it was searchable.

The document RAG pipeline was already there. ChromaDB, Ollama embeddings, chunk-and-retrieve. All I needed to do was feed session history through the same pipeline.

It took about a day to build. It took two more days to fix the problems that created.

Session-as-RAG

Every conversation now gets chunked and embedded into ChromaDB, right alongside documents. When you finish a session, the conversation text gets split into segments, each segment gets an embedding from the same model that handles documents (nomic-embed-text on the M710Q), and the vectors go into a `sessions` collection in ChromaDB.

The retrieval pipeline doesn't care whether a chunk came from a PDF or a Tuesday afternoon conversation. It searches both, ranks by relevance, and hands the top results to the chat model.

```
🍄> what did I figure out about Tailscale subnet routing?

   📚 Searching memory... 3 session excerpts, 1 document excerpt

Last week you worked through a Tailscale cross-subnet issue where
the 3090 workstation was only advertising its ethernet IP. WiFi
peers couldn't reach it. You fixed it by advertising all
non-loopback IPs and TCP-probing each address during discovery.
You also noted that Tailscale's free plan includes unlimited
subnet routers. [S:Feb 10] [D:PLAN.md]
```

That answer pulls from a conversation I had five days ago and a line in my planning doc. Before this week, the model would have either hallucinated an answer or told me to check my notes.

Topic splitting

The first version of session-as-RAG had a problem. I'd have a long chat session where I discussed three or four unrelated topics. GPU pricing, then Tai Chi class schedules, then bee hive inspection notes, then back to code. The whole session got embedded as one block.

When I later asked "what did I say about the bees?", the retrieval pulled the entire session, which included GPU pricing context that had nothing to do with bees. The model's answer was contaminated with irrelevant information. Bee hive ventilation mixed with RTX 3090 thermals.

The fix: topic detection at embedding time. When a session gets processed, the system looks for topic shifts (significant changes in subject matter between turns) and splits the conversation into segments at those boundaries. Each segment gets its own embedding.

A single session that covered GPUs, Tai Chi, bees, and code becomes four separate chunks in ChromaDB. When you ask about bees, you get the bee segment. The GPU discussion stays out of it.

Before topic splitting:

```
Query: "what did I say about hive ventilation?"
Retrieved: [full session with GPU pricing, Tai Chi, bees, code]
Answer: mixed context, mentions "thermal management" (from GPU discussion)
```

After:

```
Query: "what did I say about hive ventilation?"
Retrieved: [bee segment only]
Answer: "You noted that screened bottom boards help with ventilation
in Layens hives during summer, and you wanted to check whether
the solid boards you have can be swapped." [S:Feb 11]
```

Clean retrieval. Relevant context only.

[S] and [D] citations

You probably noticed the tags in the examples above. When the model answers using retrieved context, it now marks where the information came from:

- **[S:date]** — from a past session on that date
- **[D:filename]** — from an indexed document

This was a small addition (a few lines in the system prompt instructing the model to cite its sources) but it changes how much you trust the output. When I see `[S:Feb 10]`, I know the model is pulling from a real conversation I had. When I see `[D:PLAN.md]`, I know it checked the planning doc. When there's no citation, I know it's using training knowledge, and I know to verify.

It also makes debugging retrieval quality easy. If the model cites a session from the wrong date, or a document that shouldn't be relevant, I can see it immediately instead of wondering why the answer feels off.

Context pollution (the hard bug)

This was the bug that caused v0.1.3.

After shipping session-as-RAG in v0.1.2, I started using it normally. Within a few hours, something was wrong. Answers to simple questions were getting contaminated with unrelated

session context. I'd ask "what time is Tai Chi on Sunday?" (a fact stored via `/remember`) and the model would pull in session excerpts about Tai Chi from earlier conversations, some of which had outdated or contradictory information.

The problem: the retrieval pipeline was searching sessions for every query, even queries that should have been answered from facts alone. The model's own `/remember` facts were competing with retrieved session chunks, and sometimes the session chunks won because they had higher semantic similarity scores.

The fix was a retrieval priority system. Facts (explicit `/remember` entries) get checked first. If there's a direct fact match, session retrieval is skipped for that topic. Session-as-RAG only fires when the query doesn't match any stored facts and the model's classifier tags it as needing historical context.

Simple in retrospect. Took me most of an evening to diagnose because the symptoms were subtle. The model was still answering correctly most of the time. Just occasionally pulling in stale context that made the answer slightly wrong in ways that took a minute to notice.

This is the kind of bug that makes me appreciate the [S]/[D] citation system. Without the tags, I might not have noticed the contamination for days.

Dashboard: memory stats

The web dashboard from Week 2 showed node health and swarm topology. This week it got a memory panel:

- Total indexed sessions (count and date range)
- Total indexed documents (count and file types)
- Total chunks in ChromaDB
- Embedding model in use
- Storage used by the vector database
- Last embedding run timestamp

Nothing fancy. Just enough to answer "is my stuff actually indexed?" without opening a terminal. I kept seeing questions in the local RAG space where people couldn't tell if their documents were actually embedded or if the ingestion silently failed. The stats panel makes the state visible.

Three releases in two days

This was not planned. I intended to ship one release with everything. Instead:

v0.1.2 (Tuesday morning): Session-as-RAG, topic splitting, [S]/[D] citations, dashboard memory panel. Everything worked in testing. Pushed to PyPI.

v0.1.3 (Tuesday evening): Context pollution fix. After using v0.1.2 for real work all afternoon, the fact-vs-session priority bug became obvious. Rebuilt the retrieval priority system, tested against the specific cases that failed, pushed the fix.

v0.1.4 (Wednesday morning): Edge case in topic splitting. Very short sessions (under 5 turns) were getting split into single-turn segments that lost conversational context. Added a minimum segment length so short sessions stay as one chunk. Also fixed a bug where sessions with only one topic were still being processed through the splitting pipeline unnecessarily.

Three releases in two days because I was using my own software for actual work and finding the gaps that test suites don't cover. The test count went from 94 to 127. Every bug got a regression test before the fix shipped.

The stack right now

Node	Hardware	Cost	Role
Miu	RTX 3090, 64GB RAM	~$850	GPU inference, gemma3:27b, Qwen 2.5 32B
naru	M710Q, 8GB RAM	$50	Embeddings (nomic-embed-text), ChromaDB
uncho	M710Q, 8GB RAM	$50	Web search, file processing
boa	M710Q, 8GB RAM	$50	Web search, code execution
raspberrypi	Pi 2, 1GB RAM	$35	Search, lightweight tasks

127 tests. On PyPI at v0.1.4. Session history, documents, and facts all searchable through one pipeline.

If you want to build a similar cluster, the full parts list is on InsiderLLM.

What's next

Week 3 was about memory. Week 4 is about action.

- **Agentic planner** — multi-step task execution that breaks a complex request into subtasks and routes them across nodes. "Research X, draft Y, then review Z" as one command.
- **mTLS** — encrypted, authenticated node-to-node communication. Right now the swarm trusts anything on the LAN. That's fine for my desk. Not fine for anyone else's network.
- **Memory decay** — session chunks from months ago should be weighted lower than yesterday's conversations. Time-aware retrieval.
- **The demo video** — I keep saying this. This week I mean it.

The gap between "local AI chatbot" and "local AI that knows your work" turned out to be smaller than I expected. It's the same RAG pipeline, pointed at a different data source. The hard part wasn't building it. The hard part was making it not break the things that already worked.

---

mycoSwarm is open source. MIT license. Two commands to start:

```
pip install mycoswarm
mycoswarm chat
```

Repo: github.com/msb-msb/mycoSwarm Site: mycoswarm.org

Every node counts.
