The Episodic Buffer for AI Agents

Oct 19, 2026 · @Kelsi Guidry

Executive summary. Long-running AI agents lose the middle of their conversations: once a thread outgrows what the agent replays to its model, older turns vanish, even when their facts were saved. This paper proposes an **episodic buffer** for agents: a small, bounded index of older stretches of conversation, each line tied to the place in long-term memory where that stretch was stored. The design comes from four findings in human memory research. Built on a memory-writing step an agent already has, it adds no model calls. We describe the research, a reference design, how to implement and evaluate it, where it helps, and its limits. One production implementation, in Bizer's agent Miles, closes the paper as a case study.

The problem: agents lose the middle

Every AI agent replays only part of a conversation to its model on each turn. Most replay a fixed number of recent messages; the agent in the case study below replays 30. In a long working thread, message 31 simply disappears from what the model can see.

The facts in that lost stretch are often not gone. A good agent files decisions, prices and preferences into long-term memory as it goes. The problem is that nothing in front of the model says the thread ever covered them. An agent cannot look up what it does not know it once discussed.

The usual fixes either cost more or lose detail:

- **A bigger window** costs tokens on every turn. Models also use the middle of a long context worst (Liu et al., 2023), so more room is not the same as more memory.
- **A prose summary** of the old turns needs an extra model call. It also becomes a second copy of facts the agent already stored, and the two copies drift apart.

Human memory solved this problem long ago, without growing its capacity. This paper describes how, and how an agent can copy it.

What human memory research says

Four findings from memory research point to one design.

1. **Working memory is small, and training does not grow it.** Current estimates put short-term capacity at about four chunks, not Miller's seven (Short-term memory). Working memory training improves the trained task, with little transfer to anything else (Working memory training).
2. **What grows is the size of a chunk.** A runner known as SF raised his digit span from 7 to almost 80 over about 20 months of practice. He did it by grouping digits into race times he already knew, and his span for letters stayed near normal (Ericsson, Chase and Faloon, 1980). His capacity never grew. His chunks got bigger because each one was tied to long-term memory.
3. **First and last are remembered best.** Recall follows a serial position curve. The last items come from short-term memory (recency). The first items are recalled well because they had time to reach long-term memory (primacy) (Short-term memory).
4. **The episodic buffer binds the two.** In 2000, Alan Baddeley added a fourth part to his model of working memory: a limited-capacity buffer that links what is held now with long-term memory into single episodes (Baddeley, 2000).

Long-term memory itself is several systems. Semantic memory holds facts with no "when". Episodic memory holds events tied to a time and place, and needs that context to be recalled (Long-term memory). An agent's notes are usually semantic. What it lacks is the episode: when and in which conversation a fact was learned.

How agents handle long conversations today

Agents use three approaches, alone or together. Each one either loses the thread or copies it.

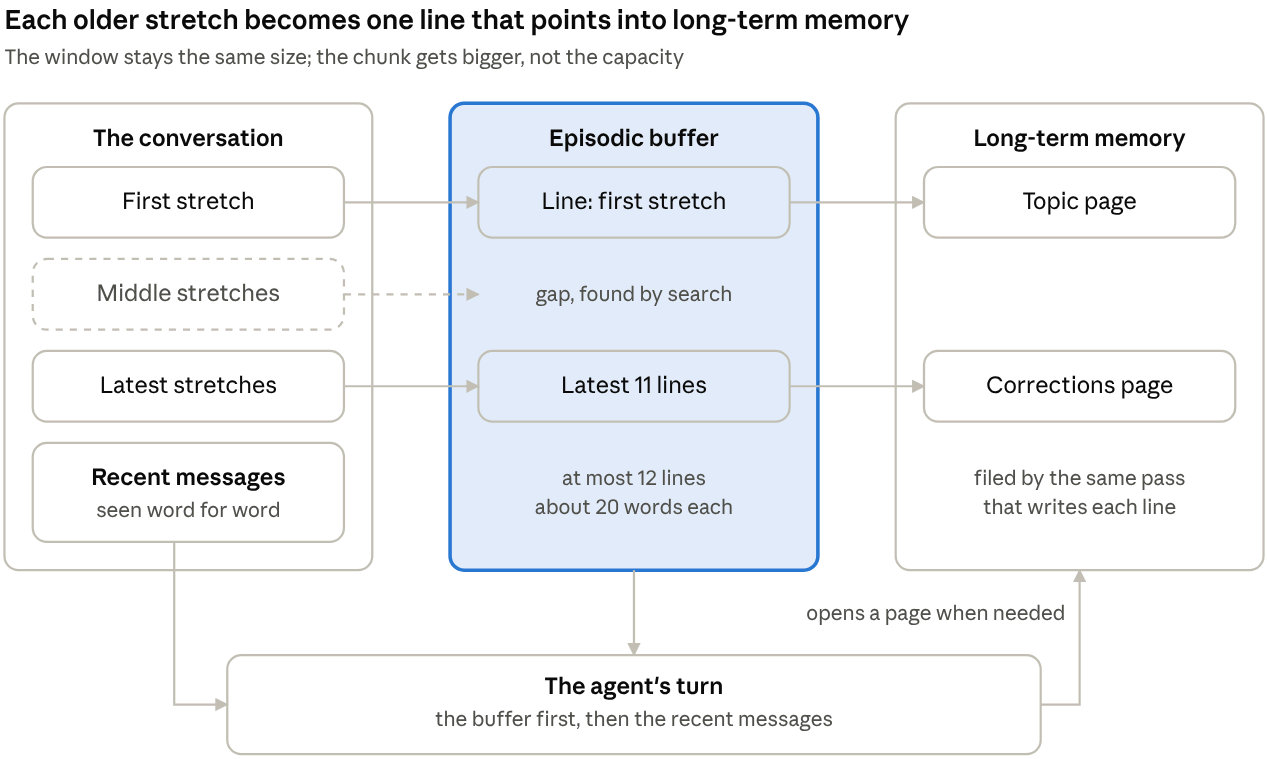
Approach	What the model sees	Extra model call	Weak point
Truncation	Only the newest messages	None	Everything older is invisible, including that it happened
Prose summary of the middle (Hermes, MemGPT)	First and last turns, plus a running summary	One per compression	A second, lossy copy of what memory already holds; it drifts and can't be checked

Approach	What the model sees	Extra model call	Weak point
Search over old messages	Whatever a search tool returns, when the model thinks to search	None	The model must already suspect something is there
Episodic buffer (this paper)	First and last turns, plus one line per older stretch naming where it was filed	**None**	Middle stretches beyond the line limit need a search

Hermes Agent protects the head and tail of a conversation and has a cheaper model summarize the middle, building each summary on the last. MemGPT evicts old messages into a recursive summary and keeps them readable in recall storage. Both are good designs. Both keep what was said as a summary, separate from the agent's long-term notes.

The episodic buffer for agents

An episodic buffer turns each stretch of conversation that scrolls out of view into one short line, and ties that line to the long-term memory pages the stretch was filed into. The model keeps a small window of recent messages, plus a handful of lines that say what came before and where to find it.



the episodic buffer · conversation, buffer, long-term memory

Read left to right: the memory pass writes one line per stretch, each line points at the pages it filed, and the agent's turn sees the buffer ahead of its recent messages. The middle stretches beyond the line limit stay reachable by search.

Design rules

Six rules make the buffer work. Each one maps to a finding above.

1. **Keep the window small; grow the chunk.** The agent's message window stays the size it is. Each older stretch of conversation becomes one line of about 20 words. Its detail stays in long-term memory, where it already was.
2. **Bind each chunk to long-term memory.** A line names the exact memory pages its stretch was filed into. That is what turns a vague summary into a pointer the agent can follow.
3. **Primacy and recency.** The buffer always shows the thread's first stretch (what the conversation was started for), then the stretches just before the visible window, oldest first. At most 12 lines.

4. **An index, not a summary.** Code builds the list. The model decides which line to open, with a tool that reads the page or one that finds the exact words. Nothing matches the user's words to pick for it.
5. **No extra model call.** The line is written by the same memory pass that already files facts after each reply. The buffer adds one short field to that call's output.
6. **Honest about what it is.** The lines are the agent's own notes. They are scanned for injected instructions, marked as not the user's words, and never accepted as proof of a number. A gap is shown only when a stretch was provably left out.

Reference implementation

The buffer needs three parts: a record, a writer and a reader. It fits any agent that already saves memory after a reply. An agent with no memory step has to build one first; the buffer is only as good as what it points at.

1. The episode record. One per consolidated stretch of a conversation, stored where only the server writes.

Field	Meaning
thread_id	The conversation it belongs to
from, to	Timestamps of the first and last message the stretch covers
date	The stretch's day, printed on the line
gist	One line, under about 25 words: what the stretch was about and where it ended
pointers	IDs of the memory records this same pass actually wrote

Key the record by `thread_id` and `to`, so a retried write replaces itself instead of duplicating.

2. The write path. Most agents with memory run a pass after replying: read the new messages, pull out durable facts, write them. Add one field to that call's structured output:

```
episode: ONE line, under 25 words, saying what this stretch of the
conversation
was about and where it ended up (decided, done, asked, left open). Name
things
concretely. Return an empty string only for small talk.
```

Then save the record. Its pointers are the records the pass **wrote**, not the ones the model proposed, so a line never points at something that isn't there.

3. The read path. Run it only when the history read comes back full, which means older messages exist:

```
if len(history) == WINDOW:
    start  = history.oldest().created_at
    first  = episodes(thread).order_by("to").first()
    recent = episodes(thread).where(to < start).order_by("to", desc).limit(N)
    if first in recent:
        shown, gap = sorted(recent), False
    else:
        shown, gap = [first] + sorted(recent)[-(N - 1):], len(recent) == N
    prepend(render(shown, gap), before=history.oldest())
```

Reading exactly N recent records makes the gap marker honest: if the first stretch isn't among them, the record dropped to make room for it proves something sits in between.

4. Guardrails. Run each gist through the same injection scan as any other stored text that reaches a prompt. Tell the model in the block that the lines are its own notes, not the user's words, and not evidence for a number. Cap the lines (12 works) and the characters per line (about 220). Never write a buffer line from a speaker you wouldn't trust in long-term memory, such as a customer in a support thread being read by the business's own agent.

Use cases

The buffer helps any agent whose conversations run longer than its window and that already files what it learns. These are the cases where we expect it to pay off; only the last has been built.

Agent	What scrolls away	A buffer line might read	It points at
Customer support	The customer's account details and the fixes already tried	Tried a password reset and a cache clear; issue persists on iOS only	The ticket and the troubleshooting log
Sales and CRM assistant	Early discovery: budget, decision makers, objections	Prospect's CFO signs off; budget fixed until Q1	The deal record and contact notes
Coding agent	Why a design was chosen, and the approaches already rejected	Rejected a queue-based retry; went with idempotent writes	The design note and the commit
Executive assistant	Standing preferences and decisions from earlier in the week	No meetings before 10 on Fridays; travel booked for the 14th	The preferences page and the calendar
Multi-agent handoff	Everything the previous agent heard	Agent A gathered requirements; open question on hosting region	The shared task record
Business operations (Bizer)	Plans, prices and decisions made over a week of chat	Compared two lease quotes; leaning to the Main St one	The business's memory pages

Where it won't help. Short, single-session chats never outgrow the window. An agent with no long-term memory has nothing for a line to point at. And an agent whose memory pass rarely writes anything will produce a buffer of thin lines that point nowhere; fix the memory pass first.

Evaluating it

The question to answer is narrow: does a fact said long ago reach the model when it is needed? Test that directly, before and after.

1. **Build threads with a planted fact.** State a specific fact early (a price, a name, a date), bury it under enough unrelated turns to push it out of the window, then ask a question that needs it.

2. **Run three arms on the same threads.** Recent messages only; recent messages plus your existing memory recall; and both plus the buffer.
3. **Run two regimes.** One with the fact outside the window, and a control with it still inside, so you can tell a retrieval gain from noise.
4. **Score the answer, not the prompt.** Did the reply use the planted fact correctly? Did the agent open the pointed-at record? Compare arms with paired confidence intervals over the same threads, not single runs.

Also measure what it costs: prompt tokens added per turn, extra reads, and turn latency. A buffer that wins recall but adds a second of latency on short threads is built wrong; it should cost nothing until a thread outgrows the window.

Case study: Miles at Bizer

Bizer, a business development platform, built the buffer into its agent Miles in October 2026. Miles already had what the design needs: a page of compiled facts per topic, recall on every turn, and a memory pass after each reply that files facts and holds anything only the agent said until the owner confirms it.

Choice	Value at Bizer
Window	Last 30 messages
Lines shown	First stretch plus the 11 latest before the window
Gist	Up to 220 characters, from the existing memory pass
Pointers	Memory pages the pass wrote; held facts get none
Who writes	Server only; owner and team threads only, never customer threads
Extra cost	No model calls; two small reads on long threads only

In practice the block reads like this (an illustrative example):

```
[EARLIER IN THIS THREAD]
- 2026-10-02: Owner wants a second location in Houma; budget under $2,500/mo
(filed in pages/Second Location)
- … more of this thread happened here; search_chat_history finds it.
- 2026-10-11: Compared two lease quotes; leaning to the Main St one (filed in
pages/Second Location)
```

Asked "which lease did we like?", the agent can see the thread covered it and open that page, instead of asking the owner to repeat themselves. Bizer has tested the design in code but has not yet measured it on live conversations.

Limits and open questions

- **No published measurement yet.** Ours included. The evaluation above is the next thing anyone adopting this should run.
- **The far middle needs search.** Stretches between the first line and the latest ones aren't listed. A very long thread depends on the agent choosing to search.
- **Only as good as the memory pass.** A pass that files little produces lines that point nowhere.
- **Gists are model-written.** They can be wrong or vague. Keeping them short, scanned and marked as notes limits the damage, but does not prevent it.
- **Is it new?** We have not found an agent design that makes compressed history a set of pointers into the agent's own long-term memory, written by the same pass that stores it. We have not surveyed every system; treat this as a claim to check.

Three extensions look promising:

1. **Feed recall.** Use the visible lines to steer the agent's memory search, so an old topic is found even when the user's words don't name it.
2. **Chunks of chunks.** Periodically fold a day's lines into one, the way expertise builds larger chunks over time, giving long threads a hierarchy under the same line cap.
3. **Reconsolidation.** When a recalled record proves wrong, have the next memory pass repair it, not just rank it lower.

Conclusion

A bigger context window is the agent version of working-memory training: it improves the measure and transfers poorly, because models use the middle of a long context worst. Human memory points to a different fix. Keep the working set small, make each chunk point into long-term memory, and protect the first and the last. An agent that already files what it learns can add an episodic buffer with one extra output field and one small read, and stop losing the middle of its conversations.

Sources

- Baddeley, A. (2000). The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences* 4(11): 417–423. Summarized at Baddeley's model of working memory.
- Ericsson, K. A., Chase, W. G. and Faloon, S. (1980). Acquisition of a memory skill. *Science* 208(4448): 1181–1182. Summarized at Chunking (psychology).
- Working memory training, Wikipedia.
- Short-term memory, Wikipedia.
- Long-term memory, Wikipedia.
- Liu, N. F. et al. (2023). Lost in the Middle: How Language Models Use Long Contexts. *TACL*.
- Packer, C. et al. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
- Nous Research. Hermes Agent, context compressor (`agent/context_compressor.py`).