Make0 Brain: Portable, User-Owned Memory Shared Safely Across AI Assistants

Saurabh Singh
Make0
saurabh@unrealai.xyz, s@make0.ai

October 2026

Abstract

People now work across several AI assistants, and each keeps its own memory. What a user teaches one assistant is missing from the next, and no single place exists where the user can see, correct, or delete what is remembered. We argue that long-term memory should be a layer the user owns and every assistant borrows, not a feature each assistant keeps to itself, and we present *Make0 Brain*, a system built on that idea. Make0 keeps one memory graph per user, outside any assistant, and serves it to ChatGPT, Claude, Cursor, and Make0's own agent through the Model Context Protocol (MCP). Sharing memory makes it writable by many agents, so the design centers on safe multi-agent writes. Read and write are separate, per-brain grants. Every memory records which assistant created and last changed it. Every change becomes a version, and deletions by assistants or by the system archive the memory instead of erasing it. Assistants can add and edit memories, but only the user can restore an earlier version or delete a memory permanently, so every change any agent makes can be undone. Behind this interface, a write path keeps the shared graph clean (extraction, contradiction handling, merging, linking), and a read path combines BM25 and dense search, reranking, score gating, and graph expansion. On LoCoMo, Make0 reaches 83.64% overall accuracy in its standard configuration, above published results for Zep (79.09%), LangMem (78.05%), and Mem0 (62.47%) but below Memori (87.0%), and 92.0% in an experimental high-budget configuration, while passing 85% fewer context tokens than the full conversation. On SWE-ContextBench Lite, held out from tuning, the gold prior task appears in Make0's top 10 for 92.93% of related tasks, against 73.34% for the best published system. We are explicit that these benchmarks measure a single agent's retrieval; the benefits of sharing across assistants are argued from the design, and we give a protocol and production metrics to measure them.

1 Introduction

Large language model (LLM) assistants have gained memory: they remember a user's preferences, projects, and past corrections across sessions. But this memory is built into each product. A preference learned by ChatGPT does not reach Claude; a correction made in Cursor does not reach either. Users who move between assistants repeat themselves, receive inconsistent answers, and cannot inspect or control what each product has kept. We call this *memory fragmentation*.

Fragmentation has three costs. First, *repetition*: a fact must be taught once per assistant. Second, *inconsistency*: a fact corrected in one assistant stays stale in the others. Third, *loss of control*: the memory lives inside each vendor, so the user cannot see one complete record, edit it, or remove it everywhere at once.

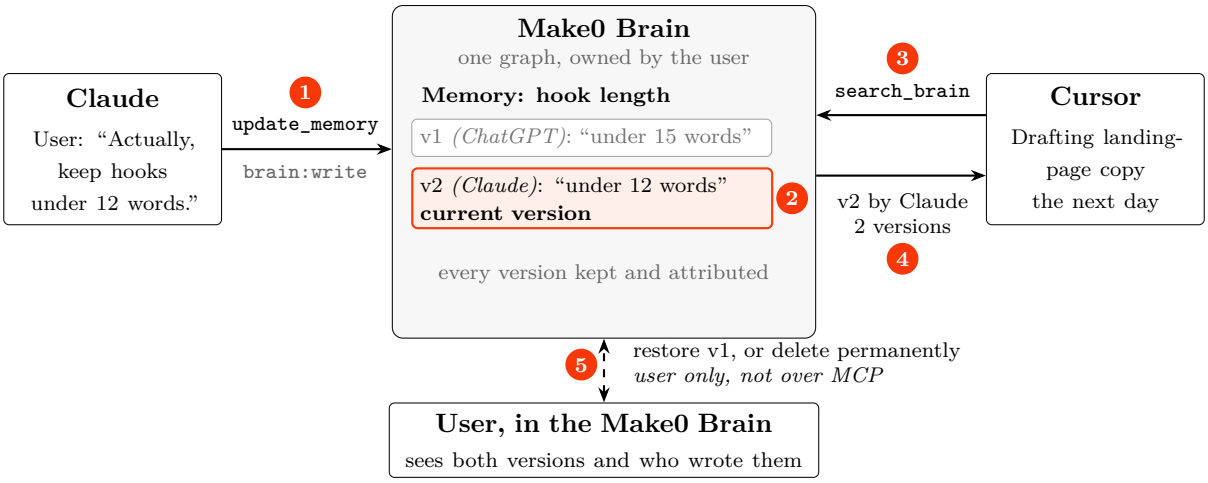


Figure 1: Product flow. **(1)** The user corrects a preference in Claude, which calls `update_memory` under its write grant. **(2)** Make0 records the change as a new version attributed to Claude; the earlier version from ChatGPT is kept. **(3)** The next day, Cursor calls `search_brain` under its read grant. **(4)** It receives the current version, with its author and version count, and writes copy that respects the correction. **(5)** If the correction was wrong, the user restores the earlier version in one place, and every assistant sees it on its next search. Assistants can add, update, and archive memories; only the user can restore versions or delete permanently.

Thesis: memory the user owns. Make0 Brain moves memory out of the assistant and into a single graph owned by the user. Any assistant that speaks the Model Context Protocol (MCP) [1] can search the graph, and, if the user grants it, write to it. The user decides which assistants see which brains, can read every memory and who wrote it, and can take access away without losing what an assistant contributed. Memory becomes portable: a new assistant starts with everything the user has already taught the others.

The hard part: many writers. Moving memory out of the assistant creates a problem that per-product memory never had. Several agents now write to the same facts. Two assistants may change the same preference; one may extract something wrong; one may be prompt-injected. A shared memory is only trustworthy if every write is attributable and reversible. Make0 therefore records the author of every memory and every version, turns deletions into archives, and reserves restore and permanent deletion for the user. An agent can change what the user remembers but cannot erase the record of that change. Figure 1 shows the resulting flow end to end: a correction made in Claude becomes a new, attributed version, which Cursor retrieves on its next search and which the user can undo.

Retrieval that scales with sharing. A shared graph accumulates facts from many sources, so retrieval must find the few relevant memories among many and do so within a small token budget, because each assistant pays for the context it receives. Make0 combines lexical search, dense search, reranking, and graph expansion, and keeps the graph clean on the way in.

Contributions.

1. **Portable, user-owned memory.** Building on recent MCP memory servers, an architecture that keeps one memory graph per user outside every assistant and serves it to independent

assistants, with separate read and write grants scoped to personal, project, and standalone brains (§3, §6).

2. **Safe multi-agent writes.** Per-assistant provenance, a version history for every memory, and archive-instead-of-delete for both agents and automatic maintenance, so that assistants can edit shared memory but only the user can restore or permanently delete (§6.2).
3. **A write path for a shared graph.** Conservative extraction, an LLM pair judge that decides whether an overlapping memory is distinct, a duplicate to merge, or a contradiction to archive, and linking, which together keep many writers from filling the graph with noise (§4).
4. **Hybrid graph retrieval.** BM25 and dense search, reranking, adaptive score gating, and graph expansion with hop decay (§5).
5. **Evaluation and a measurement plan.** We report accuracy, hit rate, and context cost on LoCoMo [8] and SWE-ContextBench [21] (§7), and a protocol and production metrics for measuring cross-assistant benefits directly (§10).

2 Related Work

Retrieval-augmented generation. RAG [6] conditions generation on retrieved passages rather than parametric knowledge alone. Long-context models offer an alternative, placing the full history in the prompt, but accuracy degrades for information in the middle of long inputs [7], and cost grows with length. Agent memory systems apply retrieval to a user's own history.

Agent memory systems. MemGPT [15] pages information between a small working context and external storage. Generative Agents [16] score memories by recency, importance, and relevance. Mem0 [3] extracts salient facts from conversations and maintains them with add, update, and delete operations, with a graph variant. Zep [17] stores memory as a temporal knowledge graph. LangMem [5] provides memory primitives for LangGraph agents. Make0 shares many of their techniques (fact extraction, graph structure, hybrid search).

Memory shared across MCP clients. Sharing memory across assistants over MCP is not new on its own. Mem0's OpenMemory MCP [10] and Supermemory MCP [19] expose one memory store to several MCP clients, such as Claude Desktop and Cursor, so context saved in one can be retrieved in another. Make0 builds on the same premise. Its focus is on what happens once several agents *write* to the same memory: who wrote each fact, how conflicting writes are resolved, and how any change an agent makes can be undone. Concretely, Make0 records per-assistant provenance on every memory and version, resolves overlapping writes with a constrained LLM pair judge, archives instead of deleting for both agents and automatic maintenance, and reserves restore and permanent deletion for the user. We are not aware of prior work that describes and evaluates such a safety model for multi-agent writes to shared memory, together with retrieval results on public benchmarks.

Graph-based retrieval. GraphRAG [4] builds an entity graph over a corpus and summarizes communities. Make0 uses a lighter graph: memories are nodes, and retrieval expands from directly matched memories to their linked neighbors, so related facts surface even when they share no words with the query.

Reranking. Cross-encoders such as BERT-based rerankers [12] trained on MS MARCO [2], including compact MiniLM variants [20], are a standard second stage after BM25 [18] or dense retrieval.

3 Memory Model

A *brain* is a directed graph $G = (V, E)$. Each memory $m \in V$ stores:

- **content**: a short natural-language statement or markdown note, e.g. "The user prefers short hooks under 12 words", capped at 2,500 characters;
- **embedding**: a 384-dimensional vector of the content, kind, entity type, and category (see *Why 384 dimensions* below);
- **kind**: `profile` (about the person) or `work` (about their work);
- **entity type**: one of `project`, `artifact`, `source`, `workflow`, `preference`, `correction`, `topic`, `person`;
- **confidence**: an integer in $[0, 100]$;
- **provenance**: the assistant that first saved the memory (`origin_provider`) and the last one to change it (`last_written_via`), each one of Make0, Claude, ChatGPT, Cursor, or another MCP host; plus links to the source conversation or message where available;
- **versions**: every saved revision, with its author, of which exactly one is current;
- **archive state**: whether the memory was archived, when, and by whom (an assistant or Make0 itself) (§6.2).

Edges $e = (m_i, m_j, r, w) \in E$ carry a relation type r and a weight $w \in [0, 1]$. Relation types are `related_to`, `part_of`, `produced`, `references`, `prefers`, `corrected_by`, and `works_on`.

Why 384 dimensions. Every write path, including the MCP tools, rejects content longer than 2,500 characters, and categories are capped at 80. The embedded text is therefore at most about 2,600 characters, roughly 600–650 tokens of English at the usual 4 characters per token. Two consequences follow. First, a memory always fits in a single embedding call (the default model accepts 8,191 tokens), so memories are never chunked and each memory has exactly one vector. Second, a short, single-topic text carries far less information than a long document, so a compact vector is a reasonable fit. We store 384 dimensions for four reasons:

- **Short inputs.** Memories are one fact or one short note, not documents, which is the setting where low-dimensional sentence embeddings are typically used.
- **Truncation is supported.** The default model, `text-embedding-3-small`, is trained so that its vectors can be shortened through the API's `dimensions` parameter, and its provider reports that shortened vectors keep most of their retrieval quality [14].
- **Storage and speed.** A 384-dimensional float32 vector takes 1.5 KB against 6 KB at the model's native 1,536 dimensions. At the largest plan's cap of 80,000 memories per user, that is about 123 MB of vectors instead of about 492 MB, and similarity computations are four times cheaper.

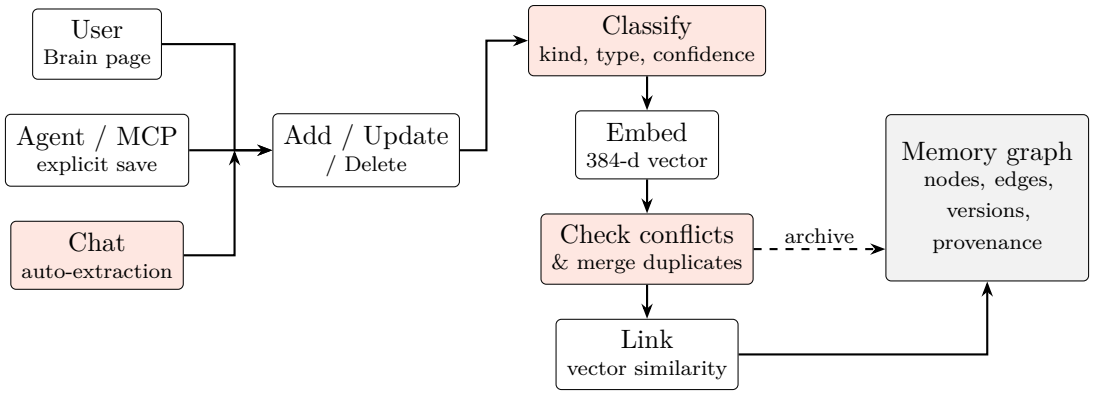


Figure 2: Write path. Shaded boxes call an LLM. Updates keep the previous wording as a version. Deletes from an assistant archive the memory (§6.2); only a permanent delete by the user removes the memory and its edges.

- **Portability.** 384 is also the output size of common open-source sentence encoders such as MiniLM [20], so the embedding model can be swapped for a self-hosted one without changing the storage schema.

Vector similarity is also only one of several signals: BM25, reranking, and graph expansion (§5) compensate for some of what a smaller vector loses. We have not ablated the dimension; see §9.

Brains are scoped. Each user has one *personal* brain; *project* brains are shared by project members; *standalone* brains hold a separate body of knowledge. An assistant only sees the brains it has been granted (§6).

4 Write Path

Memories enter from three sources (Figure 2): the user editing their brain directly; an agent saving a fact explicitly, either Make0's own agent or an external assistant over MCP; and automatic extraction from chats, uploaded documents, generated artifacts, onboarding, brand kits, and writing-style analysis.

Conservative extraction. Automatic extraction is the main source of noise, so it is gated. Short or low-signal inputs, such as very brief chat prompts, skip the LLM entirely. Each extraction may produce only a few new memories and edges, and a memory is kept only if the extractor's confidence clears a high threshold, set higher still for memories drawn from generated artifacts. Extracted memories must share content with the user's own words, so the system does not record what the assistant said as a fact about the user.

Classification and embedding. An LLM assigns kind, entity type, category, and confidence. The content is then embedded. Updates re-embed the new wording and keep the old wording as a version.

Resolving conflicts: an LLM pair judge. When many agents write to one graph, a new memory often overlaps an old one. It may restate it, refine it, or contradict it ("prefers dark mode" versus "no longer uses dark mode"). Each case needs a different action, and lexical or embedding

similarity alone cannot tell them apart: a contradiction usually looks *more* similar to the original than an unrelated fact does. Make0 therefore asks an LLM to judge each overlapping pair.

For an incoming memory m, a cheap lexical-overlap prefilter shortlists existing memories that could be about the same thing, and the strongest few, up to a fixed budget per write, go to the judge. For each pair (m_{old}, m) the judge, run deterministically with a JSON output format, returns one of three verdicts:

- **distinct**: both can be true and they state different facts. Nothing changes.
- **merge**: they state the same fact. The two are combined into one memory and the absorbed one is archived. The judge may propose the merged sentence, but Make0 accepts it only if it adds nothing new: every content word of the merged text must appear in one of the two inputs, with only connecting words ("and", "the", "with") allowed in addition. Formally, with $W(\cdot)$ the set of normalized tokens and G a small set of function words,

$$\text{accept}(m_{\text{merged}}) \iff W(m_{\text{merged}}) \subseteq W(m_{\text{old}}) \cup W(m) \cup G. \tag{1}$$

  Otherwise a deterministic merge of the two texts is used. This check stops the judge from inventing a name, date, or preference while merging.
- **contradict**: they cannot both be true. The older memory is archived (§6.2), so retrieval never returns both, but the user can still restore it.

Two other signals still apply. The extractor may mark a new memory as correcting an older one (`corrected_by`). Pairs beyond the judge's budget, or any pair where the judge call fails, fall back to the earlier checks: opposite-polarity patterns ("prefers" versus "no longer" or "avoids") confirmed by a MiniLM cross-encoder trained on MS MARCO for contradictions, and a high cross-encoder score for duplicates. The budget bounds the cost and latency of a write; the fallback keeps every pair covered.

Who can retire whom. A contradiction does not let any writer overrule any memory. Memories the user wrote by hand, and memories an assistant saved explicitly over MCP, are protected: a contradicting memory archives them only if it is itself a correction, written by the user, or saved by an assistant. A fact merely extracted from a chat cannot retire something the user or an assistant stated deliberately. Explicit edits made through `update_memory` do not go through the judge at all; they keep the previous wording as a version.

Linking. Each memory is compared against its nearest neighbors by cosine similarity. Pairs above a similarity threshold become `related_to` edges weighted by similarity, with a cap on links per memory so that densely connected topics do not dominate expansion.

Maintenance. A background job archives auto-extracted memories whose confidence has fallen below a floor, removes edges that have become weak, and periodically consolidates related memories. Consolidation uses the same pair judge: duplicates are merged into the richer memory, and of two contradicting memories the older one is archived. Memories the user added by hand are protected from automatic removal. No automatic step deletes a memory permanently: contradiction handling, duplicate merging, pruning, and consolidation all archive the memory with its versions, and only the user can delete permanently.

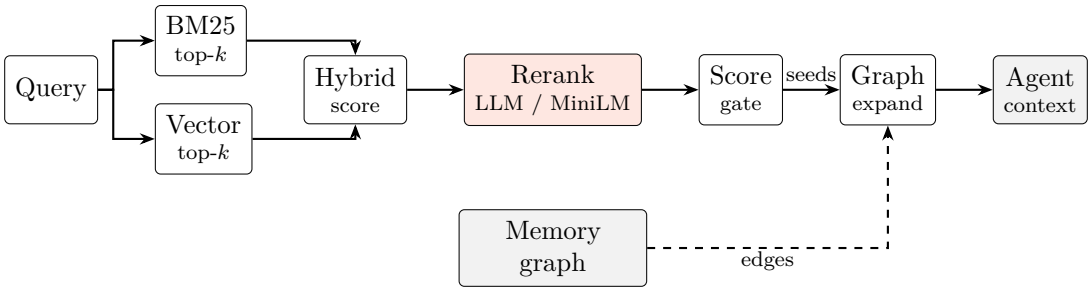


Figure 3: Read path. The shaded stage calls an LLM; it falls back to a MiniLM cross-encoder, and to the hybrid score alone if that also fails.

5 Read Path

Given a query q, retrieval proceeds in five stages (Figure 3).

1. Candidate generation. Two retrievers run in parallel: cosine similarity between embeddings, and BM25 over the memory text. Dense search finds paraphrases ("brief openings" for "short hooks"); BM25 finds exact terms that embeddings blur, such as names, product terms, numbers, and dates. Both matter for personal memory, where many questions hinge on a specific name or day.

Dense scoring with cosine similarity. Let E be the embedding model and $\mathbf{e}_q = E(q)$, $\mathbf{e}_m = E(m) \in \mathbb{R}^{384}$ the query and memory embeddings. The memory's embedding covers its content together with its kind, entity type, and category labels, so "a preference about writing" can match on the labels as well as the words. The dense score is the cosine of the angle between the two vectors:

$$v(q,m) \;=\; \cos\theta \;=\; \frac{\mathbf{e}_q \cdot \mathbf{e}_m}{\|\mathbf{e}_q\|\,\|\mathbf{e}_m\|} \;=\; \frac{\sum_{i=1}^{384} e_{q,i}\, e_{m,i}}{\sqrt{\sum_i e_{q,i}^2}\,\sqrt{\sum_i e_{m,i}^2}} \;\in\; [-1,1]. \tag{2}$$

Three properties make this the right measure here:

- **Direction, not size.** Cosine depends only on where a vector points, so a one-line fact and a long note about the same topic score alike. Dividing by the norms also re-normalizes the shortened 384-dimensional vectors, so no separate normalization step is needed.
- **Precomputed on write.** This is a *bi-encoder*: the query and the memory are embedded separately. Every memory's vector is computed once, when it is written or edited, so a search costs one embedding call for the query plus one dot product per memory, $O(Nd)$ for N memories of dimension d.
- **Exact search.** Because a brain is bounded by its plan's memory limit, Make0 scores every live memory in the brain rather than using an approximate nearest-neighbor index. Recall at this stage is therefore never lost to index approximation.

The cost of a bi-encoder is that the query and memory never "see" each other: the score cannot model which words in the query matter for this particular memory. That is the job of the reranker (stage 3).

Lexical scoring with BM25. We use Okapi BM25 [18]. The collection is the brain itself: the N live memories in the brain being searched, with archived memories excluded. For a query q with term set $T(q)$ and a memory m, the score is

$$\mathrm{BM25}(q,m) \;=\; \sum_{t\in T(q)} \mathrm{IDF}(t)\cdot \frac{f(t,m)\,(k_1+1)}{f(t,m)+k_1\left(1-b+b\,\dfrac{|m|}{\overline{|m|}}\right)}, \tag{3}$$

where $f(t,m)$ is how often term t occurs in m, $|m|$ is the memory's length in tokens, and $\overline{|m|}$ is the mean length over the brain. The inverse document frequency is the non-negative form used by Lucene,

$$\mathrm{IDF}(t) \;=\; \ln\!\left(1+\frac{N-n(t)+0.5}{n(t)+0.5}\right), \tag{4}$$

with $n(t)$ the number of memories that contain t. The three factors each do one job:

- **Rarity, per user.** IDF rewards terms that are rare *in this brain*. Because it is computed over the user's own memories, a word that appears everywhere for one user (their company or main project) counts for little in their searches, while the same word can be decisive for another user. A global corpus would not capture this.
- **Saturating term frequency.** The fraction in Eq. (3) grows with $f(t,m)$ but approaches k_1+1, so repeating a word does not keep raising the score. We set k_1 so that it saturates quickly: memories are short, and one mention of a term is already strong evidence.
- **Length normalization.** The b term penalizes memories longer than average, on the assumption that a long text matches many terms by chance. Memories range from one-line facts to notes of up to 2,500 characters, and a longer note is usually more detailed rather than less relevant. We therefore set b well below the value commonly used for long documents, so notes are penalized only mildly against short facts.

Terms. Text is lower-cased, split on non-alphanumeric characters, and lightly normalized: ordinal suffixes are removed ("5th" → "5"), a trailing plural "s" is dropped, and very short tokens are discarded except digits and two-letter terms such as "AI" or "JS". Three additions adapt BM25 to memory:

- **Phrase terms.** Each pair of adjacent words also becomes a term, so "short hooks" matches a memory containing that phrase more strongly than one containing "short" and "hooks" far apart. Pairs that include a stopword are skipped.
- **Date aliases.** Month names and numbers are expanded into each other ("March", "3", "03"), and parsed dates add their alias tokens. A question about "03/2024" then matches a memory that says "March 2024".
- **Query-only stopwords.** Function words such as "what", "did", or "my" are removed from the query but not from memories. Question words would otherwise match many memories and add noise.

For example, the query "short hooks" becomes the terms {`short`, `hook`, `short_hook`}. The memory "Prefers short hooks under 12 words" contains all three, so it outranks a memory that only mentions hooks.

Normalization. Raw BM25 scores have no fixed scale: they depend on the brain's size and the query's length. Before blending with cosine similarity, which lies in a bounded range, each score is divided by the best score for the query:

$$\hat{b}(q,m) \;=\; \frac{\mathrm{BM25}(q,m)}{\max_{m'} \mathrm{BM25}(q,m')} \;\in\; [0,1]. \tag{5}$$

The best lexical match thus always scores 1, and the hybrid score compares like with like across queries and brains.

Candidate set. Each retriever nominates its top K memories, keeping only memories with a cosine score above a small floor and a BM25 score above zero. The candidate set is their union:

$$C(q) \;=\; \mathrm{Top}_K\{\, m : v(q,m) > v_{\min} \,\} \;\cup\; \mathrm{Top}_K\{\, m : \mathrm{BM25}(q,m) > 0 \,\}, \qquad |C(q)| \le 2K. \tag{6}$$

A memory found by only one retriever still enters the set, so a paraphrase with no shared words and an exact name match with a weak embedding both survive. When a brain holds few memories, $C(q)$ is simply every memory in it.

2. Hybrid score. Each candidate receives one score that combines the two signals, plus a date signal when the query refers to a time. Let v be the cosine score of Eq. (2) and $\hat{b}$ the normalized BM25 score of Eq. (5). The date-match score $d(q,m) \in [0,1]$ compares dates parsed from the query and from the memory, in decreasing order of strength:

- the same day, month, and year;
- the same month and year;
- if the query asks about time ("when did…") without naming a date: memories that carry a full date, then those with a month, then any other dated memory;
- otherwise $d = 0$.

The hybrid score is a convex combination:

$$s_{\mathrm{hyb}}(q,m) = \begin{cases} \alpha_d\, v + \beta_d\, \hat{b} + \gamma_d\, d & \text{if } \exists\, m' \in C(q) : d(q,m') > 0, \\ \alpha\, v + (1-\alpha)\, \hat{b} & \text{otherwise,} \end{cases} \qquad \alpha_d + \beta_d + \gamma_d = 1. \tag{7}$$

The dense score carries slightly more weight than BM25, and when a date is involved a small share moves to the date match. Four choices are deliberate:

- **Score fusion rather than rank fusion.** Reciprocal rank fusion would combine only the two rank lists. A weighted sum of scores keeps the *margins*: whether the top memory is far ahead of the rest or nearly tied. The relative cutoff in stage 4 depends on exactly that margin. Normalizing BM25 to $[0,1]$ (Eq. 5) is what makes the sum meaningful.
- **One weighting per query.** The date weights apply to every candidate as soon as any candidate matches a date, not only to the matching ones. All scores within a query are therefore on the same scale, and a dated memory is lifted relative to undated ones rather than undated ones being scored by a different rule.
- **The date is a nudge, not a filter.** With a small γ_d, a memory from the right month still has to be about the right thing. An exact date match alone cannot outrank a strongly relevant memory.
- **Graceful degradation.** If a memory has no embedding yet, or the query embedding fails, s_{hyb} falls back to $\hat{b}$, so the memory can still be found by its words.

3. Reranking. The hybrid score is fast but shallow: cosine compares two independent summaries, and BM25 counts words. Reranking reads each candidate *together with* the query, which is too expensive for the whole brain but affordable for the shortlist $C(q)$. Make0 reranks with an LLM and falls back to a cross-encoder.

LLM rerank (primary). A fast LLM, run deterministically (temperature 0) with a JSON output format, works in two steps.

1. **Filter.** The LLM sees the query and the numbered candidates, each with its hybrid score, and returns the subset $K(q) \subseteq C(q)$ that could help answer the query. It is told to drop memories that merely share a common word, to prefer precision, and never to return an empty set. The filter is a hard gate: memories outside $K(q)$ leave the result.
2. **Score.** If more than a few memories survive, the LLM assigns each a relevance $r(q, m) \in [0, 1]$ and is told to rank *complementary* facts above near-duplicates, so the context covers more ground. If only a few survive, or if this scoring step fails, the filtered memories keep their hybrid order.

Cross-encoder (fallback). If the filter step fails, times out, or returns nothing usable, a MiniLM cross-encoder trained on MS MARCO [2, 20] scores every candidate. Unlike the bi-encoder of Eq. (2), it encodes the concatenated pair, so attention runs across query and memory tokens, and outputs the probability that the memory is relevant:

$$s_{\text{ce}}(q, m) \;=\; P(\text{relevant} \mid [\,q;\, m\,]) \;=\; \sigma\big(g([\,q;\, m\,])\big) \in [0, 1], \tag{8}$$

where g is the transformer's relevance logit and σ the logistic function. Long memories are truncated to the model's input budget.

Blending. Neither reranker replaces the hybrid score; each is blended with it:

$$s(q, m) = \begin{cases} \lambda\, r(q, m) + (1 - \lambda)\, s_{\text{hyb}}(q, m) & m \in K(q),\ \text{LLM rerank,} \\ s_{\text{hyb}}(q, m) & m \in K(q),\ \text{few survivors or scoring step failed,} \\ \lambda\, s_{\text{ce}}(q, m) + (1 - \lambda)\, s_{\text{hyb}}(q, m) & m \in C(q),\ \text{cross-encoder fallback,} \\ s_{\text{hyb}}(q, m) & m \in C(q),\ \textit{fast}\ \text{mode or both rerankers unavailable,} \end{cases} \tag{9}$$

with $\lambda > \frac{1}{2}$, so the reranker dominates. Keeping a share of s_{hyb} has two benefits. It preserves exact lexical and date evidence, which rerankers sometimes undervalue, and it breaks ties: an LLM tends to give coarse scores such as 0.9 or 0.8 to several memories, and the hybrid score orders them. The latency-sensitive *fast* mode used inside chat skips reranking entirely. Make0 Max (§7) uses a larger model, GPT-5.6 Luna, for the LLM rerank.

4. Score gate. A memory is kept as a *seed* if $s \geq \tau$ and $s \geq \rho \cdot s_{\max}$, where $s_{\max}$ is the top score, τ is a low absolute floor, and $\rho < 1$ is a relative cutoff. If fewer than a minimum number of memories survive, the gate is padded from the shortlist up to that minimum. The relative cutoff adapts to the query: a sharp query keeps few memories, a broad one keeps more.

5. Graph expansion. Seeds bring in their neighbors. A neighbor reached at hop h through an edge of weight w from a node with score s_{parent} scores

$$s_{\text{child}} = s_{\text{parent}} \cdot w \cdot \delta, \qquad \delta < 1, \tag{10}$$

expansion runs to at most 2 hops, and an edge must have weight $w \geq w_1$ at hop 1 and $w \geq w_2$ at hop 2, with $w_2 > w_1$ so that only strong links are followed twice. Expansion is how the system

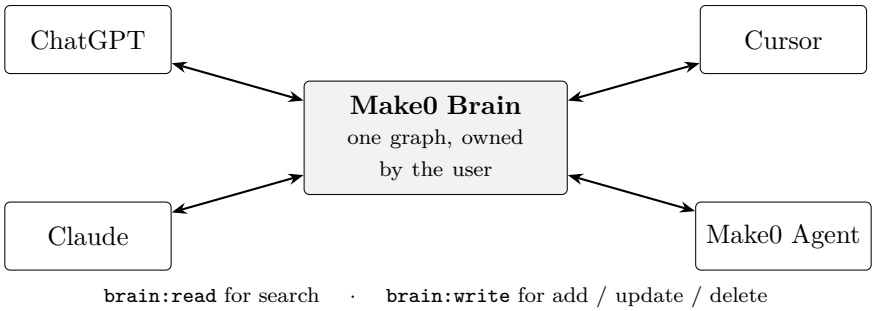


Figure 4: Every assistant reads and writes the same graph over MCP. Read and write are separate grants.

answers multi-hop questions. For example, a query about writing style matches "prefers short hooks" directly, and expansion adds "writes in a direct tone" and "one idea per slide" as neighbors. If the result exceeds the budget, hop-2 neighbors are dropped first, then hop-1, then the lowest-scoring nodes; seeds are always kept.

The final list is returned as context, with each item marked as a seed or a neighbor, its score, and the relation that brought it in. Retrieval takes about 0.85 s from query to ranked context.

How the parameters were chosen. The parameters in this section and in §4 (k_1, b, α, α_d, β_d, γ_d, λ, τ, ρ, δ, w_1, w_2, candidate counts, and the extraction, merge, contradiction, and linking thresholds, and the pair judge's per-write budget) were set empirically. We describe their roles and relative sizes but do not publish their values. We adjusted them over repeated benchmark runs and kept the values that gave the best accuracy. These runs included LoCoMo. SWE-ContextBench was not used during tuning; it was run afterwards with the parameters fixed, so it serves as a held-out check (§7). No model weights were trained: the embedding model, the cross-encoder, and the LLMs are used as released.

6 Cross-AI Memory over MCP

6.1 Interface

Make0 exposes the brain as an MCP server with seven tools:

- `list_brains`: the brains this connection may access, and whether it may write to each;
- `search_brain`: the read path of §5, with optional filters by kind and entity type and a limit of 1–64 results; it also returns relevant snippets from the user's workspace files;
- `get_memory` and `get_memory_versions`: a full memory, and its revision history;
- `add_memory`, `update_memory`, `delete_memory`: the write path of §4; `delete_memory` archives rather than deletes (§6.2).

Connections are authorized with OAuth. The `brain:read` scope allows search; `brain:write` is a separate grant that the user must give explicitly. Grants list the specific brains a connection may see, so a work assistant can be given a project brain without the personal one. Each search result includes who first added the memory and how many versions it has, so an assistant can tell a fact the user typed from one another assistant inferred.

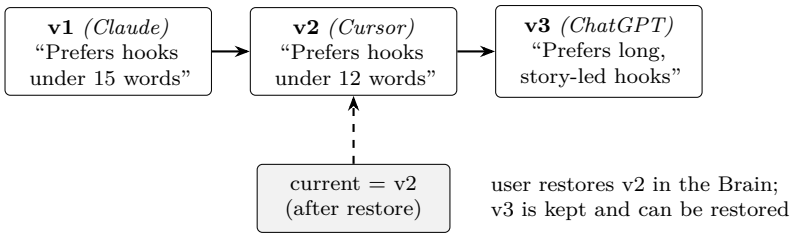


Figure 5: One memory edited by three assistants. Each write adds a version with its author. A restore moves the *current* marker back; it does not delete later versions. Every assistant's next search returns the current version.

6.2 Versions and archive: many writers, one history

When several assistants can write, the same memory will be changed by more than one of them. A user may tell Claude a preference, refine it in Cursor, and have ChatGPT record something that contradicts both. Without history, the last writer silently wins, and an earlier, correct statement is lost. Make0 keeps every revision (Figure 5).

Recording. Each change to a memory's content, category, kind, entity type, or confidence adds a numbered version holding those fields and its author: the user, Make0, or the MCP host (Claude, ChatGPT, Cursor, or another). The new version is written in one database transaction that also clears the previous current marker. Two constraints keep the history consistent when assistants write at the same time: version numbers are unique per memory, and at most one version per memory is current. The memory row always mirrors its current version, and the content is re-embedded when it changes, so retrieval and every assistant see the same text.

Reading. Every search result reports how many versions a memory has. When there is more than one, an assistant can call `get_memory_versions` to read the full history, newest first, with each version's author. An assistant can therefore see that a preference was changed and by whom, rather than treating the current text as the only thing ever said.

Restore and delete. The user can restore any earlier version. A restore copies that version's fields back into the memory, re-embeds it, and marks it current. It does not create a new version or delete later ones, so the user can move forward again. The user can also delete a single version; deleting the current one promotes the newest remaining version, and the last version cannot be deleted.

Who can do what. Assistants with `brain:write` can add and update memories, which adds versions. Restoring a version and deleting a single version are only possible for the user, in the Brain interface; neither is exposed over MCP. An assistant that *updates* a memory wrongly therefore cannot also rewrite the history that lets the user undo the change. The same holds for deletion: `delete_memory` over MCP *archives* a memory rather than removing it. An archived memory is hidden from search and from memory limits, keeps all its versions, and records who archived it: an assistant, or Make0 itself when contradiction handling, merging, or pruning retires a memory (§4). The user sees archived memories in the Brain and can restore one, with its full history, or delete it permanently. Only the user can permanently delete a memory, so every change to the graph, by any assistant or by the system, can be undone.

6.3 What sharing provides

We separate four benefits. The first two follow from the design; the last two are measured in §7.

One teaching, many assistants. With n assistants and separate memories, a fact must be learned up to n times, once in each product, and each product may extract it differently. With a shared brain the fact is learned once. A preference the user states in Claude ("keep hooks under 12 words") is returned by `search_brain` when Cursor or ChatGPT next drafts copy for that user.

Corrections propagate, and can be undone. Because search reads the current version of each memory, an update made through any assistant is what every other assistant sees on its next query; the superseded wording is kept as a version with its author, and a contradicted memory is archived rather than erased. With separate memories, a correction in one product leaves the stale fact in all others, and the user may not know it is there. If a correction turns out to be wrong, the user restores the earlier version once, in one place, and every assistant sees the restored text on its next search (§6.2).

Retrieval quality on a shared graph. A shared graph is larger and more varied than any single assistant's memory, which makes retrieval harder. Results on LoCoMo and SWE-ContextBench (§7) show the read path stays accurate.

Lower context cost per assistant. The alternative to shared memory is pasting history into each assistant. On LoCoMo, Make0 supplies the needed context in a mean of 4,004 tokens per question, against 26,031 for the full conversation [11]. Each assistant pays only for the shortlist.

User control. Memory that sits outside every assistant can be inspected, edited, versioned, exported, and deleted in one place. Provenance fields show which assistant wrote each fact, and the brain can be filtered by author. Revoking an assistant's grant removes its access without losing what it contributed.

7 Evaluation

7.1 Setup

Configurations. *Make0 Standard* is the production read path of §5. Retrieval takes under 1 s, and end-to-end latency, including answering, is 1–3 s per question. *Make0 Max* is an experimental configuration that changes two things. It reranks the shortlist with GPT-5.6 Luna instead of the fast reranking LLM, and it doubles the context-token budget, so more memories reach the answer model. Everything else, including the hybrid score, score gate, and graph expansion, is the same as Standard. The larger reranker makes retrieval slower, up to 5 s per query, and end-to-end latency is 8 s or more per question.

LoCoMo. LoCoMo [8] contains very long multi-session conversations with questions in four categories: single-hop, multi-hop, open-domain (world knowledge), and temporal. We ingest each conversation into a brain, retrieve context for each question, generate an answer, and score it with an LLM judge. For Make0, both answerer and judge are GPT-5.6 Luna. Standard was evaluated on 15 Sep 2026 and Max on 10 Sep 2026. Results are also published on the Make0 benchmark pages [9].

Table 1: LoCoMo accuracy (%) by question category. Make0 rows are our evaluation; other rows are published scores with different answer and judge models. Full context passes the entire conversation to the answer model and serves as a reference ceiling. Best in each column in bold.

System	Single-hop	Multi-hop	Open-domain	Temporal	Overall
Make0 Max (experimental)	**94.40**	**90.60**	**76.90**	**97.30**	**92.00**
Make0 Standard	92.51	72.34	64.58	76.01	83.64
Full context [11]	88.53	77.70	71.88	92.70	87.52
Memori [11]	88.50	83.70	72.90	90.30	87.00
Zep [17]	79.43	69.16	73.96	83.33	79.09
LangMem [5]	74.47	61.06	67.71	86.92	78.05
Mem0 [3]	62.41	57.32	44.79	66.47	62.47
OpenAI memory [3]	63.79	42.92	62.29	21.71	52.90

SWE-ContextBench. SWE-ContextBench [21] tests whether a system can find the right prior experience for a new software-engineering task. For each experience trajectory, Make0 stores its closing summary as a memory. For each related task, we query with the task and check whether the gold experience is among the returned results. We report the Lite split (300 experience trajectories, 99 related tasks; evaluated 2 Oct 2026) and the full set (1,007 trajectories, 357 related tasks). Make0 returns its top 10. SWE-ContextBench was not used to tune any parameter: both runs used the settings already fixed during tuning, so these results are held out.

Baselines. Baseline numbers are published results, not our runs. LoCoMo accuracy and context-token figures for Memori, Zep, LangMem, Mem0, and the full-context baseline come from Memori's benchmark report [11]. The OpenAI memory row comes from the Mem0 paper [3], which reports Mem0 itself at 66.88% overall under its own protocol; we use Memori's Mem0 figure so that most baselines share one source. SWE-ContextBench baselines come from Table 5 of the benchmark paper [21]. See §9 for what this implies.

7.2 LoCoMo results

Table 1 and Figure 6 report accuracy. Make0 Standard reaches 83.64% overall: 4.55 points above Zep, 5.59 above LangMem, and 21.17 above Mem0, but 3.36 points below Memori (87.00%) and 3.88 below the full-context reference (87.52%). It has the highest single-hop accuracy of any row except Make0 Max (92.51% versus 88.50% for Memori), the category most directly served by hybrid search and reranking. On multi-hop questions it leads Zep by 3.18 points but trails Memori by 11.36.

Make0 Standard is also behind published results on open-domain questions (64.58% versus 73.96% for Zep and 72.90% for Memori) and temporal questions (76.01% versus 90.30% for Memori and 86.92% for LangMem). Open-domain questions depend on world knowledge outside the conversation, which memory retrieval cannot supply. Temporal questions need event dates; Standard's date handling is a small additive boost (§5), which the Max configuration's 97.30% suggests leaves room for improvement.

Make0 Max leads every category, including the full-context reference, and reaches 92.00% overall, 8.36 points above Standard. That gain comes from a stronger reranker and twice the context budget, at a retrieval cost of up to 5 s per query against under 1 s for Standard. The largest gains are on

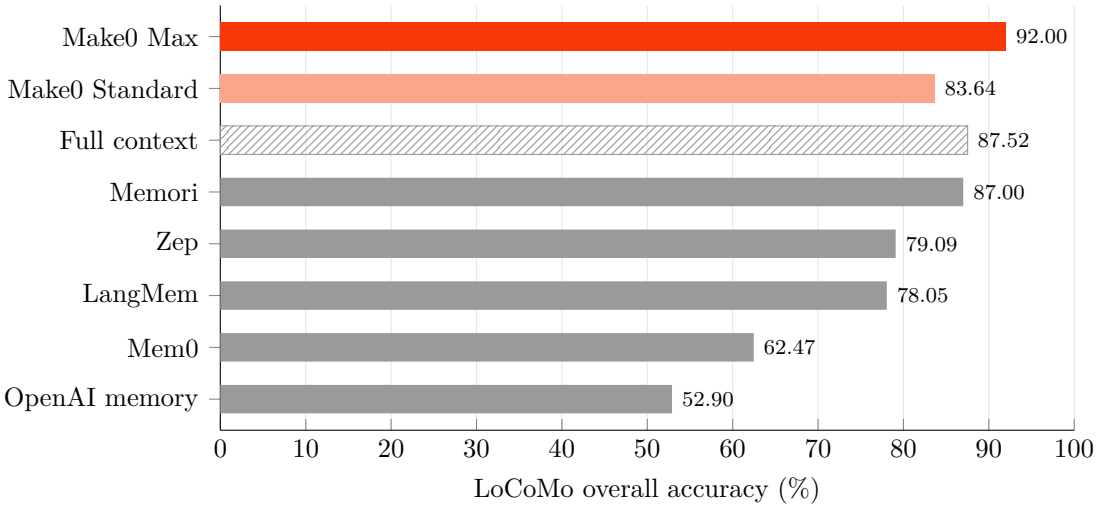


Figure 6: LoCoMo overall accuracy. Grey bars are published results; the hatched bar is the full-context reference.

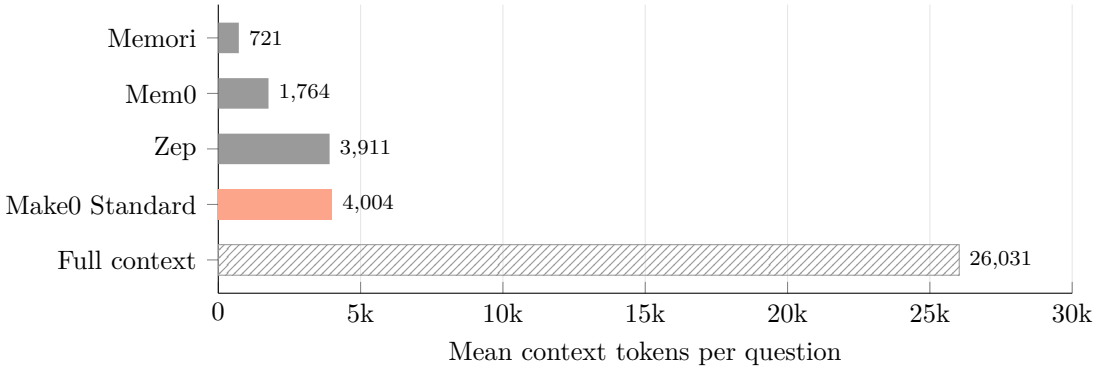


Figure 7: Mean context tokens per LoCoMo question. Make0 is our measurement; other rows are from Memori's report [11].

multi-hop (+18.26) and temporal (+21.29) questions, where Standard is weakest. This suggests that Standard's shortfall there is mostly in ranking and context size rather than in what the graph stores.

Retrieval recall is high: Make0 Standard has a Hit@10 of 98.6%, meaning relevant evidence is among the top 10 retrieved memories for nearly every question. This suggests most remaining errors arise in answering rather than retrieval.

7.3 Context cost

Figure 7 shows the context each method passes to the answer model. Make0 Standard uses a mean of 4,004 tokens per question, 84.6% less than the full conversation (26,031 tokens).

Make0 is not the most compact memory system. It uses about the same context as Zep (3,911 tokens) while scoring 4.55 points higher, but Mem0 (1,764) and Memori (721) use far less: Mem0 at a cost of 21.17 points of accuracy, and Memori while scoring 3.36 points higher. Reducing context without losing recall is an open direction for Make0. One candidate is the score gate's minimum result count (§5), which pads the context even when only a few memories are relevant. Token

Table 2: SWE-ContextBench: share of related tasks (%) whose top-1 retrieved context matches the gold experience (Top-1), and whose returned contexts include it at any position (Any). Baselines are from Table 5 of [21].

System	Returned	Top-1	Any
Lite (300 experiences, 99 related tasks)			
Make0 Standard	Top 10	**70.71**	**92.93**
LangMem	Top 10	55.56	73.34
Supermemory	Top 15	23.23	59.60
OpenViking	Top 3	38.38	51.52
Mem0	Top 3	25.25	39.39
Free Summary Learning	Agent chooses	34.34	36.36
Free Context Learning	Agent chooses	18.18	18.18
Full (1,007 experiences, 357 related tasks)			
Make0 Standard	Top 10	70.87	86.83

counts depend on the tokenizer and prompt format, so small differences between sources should not be over-read.

In a cross-AI setting the saving over full context repeats per assistant and per query. A user who brings their history to three assistants by pasting the conversation would spend roughly $3 \times 26{,}031 \approx 78{,}000$ tokens of context; retrieving from a shared brain costs roughly $3 \times 4{,}004 \approx 12{,}000$. This is an illustrative calculation from the measured means, not a separate experiment.

7.4 SWE-ContextBench results

Table 2 reports SWE-ContextBench. On Lite, Make0 places the gold experience in its top 10 for 92.93% of related tasks, 19.59 points above LangMem at the same depth, and ranks it first for 70.71%, 15.15 points above LangMem. Systems that return fewer results (Mem0 and OpenViking at top 3) are not directly comparable on the Any column, but Make0's Top-1 rate alone exceeds their Any rate.

On the full set, which has more than three times as many stored experiences, Top-1 holds at 70.87% and Any falls to 86.83%. The stable Top-1 rate suggests that ranking quality holds as the brain grows, which matters for a shared brain that accumulates memories from many assistants.

Because SWE-ContextBench was held out from tuning, these results also test whether parameters chosen on conversational memory carry over to a different domain: coding experiences rather than personal conversations, with longer stored texts and more technical vocabulary. They do. With unchanged settings, Make0 has the highest Top-1 and Any rates in Table 2.

8 System and Infrastructure

Make0 Brain runs as a cloud service. Memories, edges, versions, and embeddings are stored in PostgreSQL. Self-hosted models run on cloud NVIDIA H100 and A100 GPUs; among them, speech input is transcribed with NVIDIA Parakeet TDT 0.6B v3 [13]. In production, and in every result in this paper, memories and queries are embedded with OpenAI `text-embedding-3-small` [14] at 384 dimensions. The embedding model is configurable, and each memory records which model embedded it, so vectors can be re-embedded if the model changes.

9 Limitations

Baselines are not re-run. All baseline numbers are published results. On LoCoMo they were produced with different answer models and judges from ours (GPT-5.6 Luna), and LLM-judged accuracy is sensitive to both. The gaps in Table 1 should be read as indicative. A controlled comparison, with all systems run under one answerer and judge, is future work. In Make0 Max, the reranker, answerer, and judge are the same model (GPT-5.6 Luna), which may favour Max over configurations whose context was selected by a different model.

Cross-AI benefits are not yet measured directly. LoCoMo and SWE-ContextBench evaluate a single agent retrieving from a single memory. The benefits in §6 of teaching once and propagating corrections follow from the design, but we have not measured them, for example as reduced repetition by users or fewer stale answers across assistants. We outline such an evaluation below.

Parameters were tuned on the evaluation benchmark. Because the parameters of §5 were tuned over repeated LoCoMo runs, the LoCoMo results may be optimistic. We did not hold out a LoCoMo split for tuning. SWE-ContextBench, which was not used for tuning, gives some evidence that the settings transfer, but it measures retrieval only, not answer accuracy, so it does not bound how optimistic the LoCoMo accuracy is. A cleaner protocol would tune on a subset of LoCoMo conversations and report the rest, and would report how sensitive accuracy is to the main settings.

The pair judge can be wrong. An LLM may call two compatible facts a contradiction, or two different facts a duplicate. The design limits the damage rather than preventing it: merged text must pass the check of Eq. (1), facts the user or an assistant stated deliberately cannot be retired by extraction alone, and every merge or contradiction archives the losing memory, so the user can restore it. We have not yet measured the judge's error rate; the false-overwrite controls in the planned correction track (§10) are designed to do so.

Exact parameter values are withheld. We describe every stage of the write and read paths and how scores are combined, but not the tuned values of their parameters. The results can therefore be checked against the hosted service, but the system cannot be rebuilt exactly from this paper alone.

Embedding size is not ablated. The choice of 384 dimensions (§3) is a design decision based on memory length, cost, and portability. We have not measured retrieval quality at 1,536 dimensions or with open-source encoders on our benchmarks.

Shared memory has risks. A shared graph lets one assistant's mistaken extraction reach every other assistant. Provenance, versioning, confidence thresholds, and separate write grants limit this but do not remove it. A malicious or prompt-injected assistant holding a write grant could insert false memories. Updates can be undone from the version history and deletions from the archive, but a newly added false memory has no earlier version to restore; the user must find and delete it.

Standard configuration is not best on every axis. Make0 Standard trails Memori and the full-context reference on overall LoCoMo accuracy, trails published results on multi-hop, open-

domain, and temporal questions, and uses more context per question than Memori, Mem0, or Zep.

10 Future Work: Measuring Cross-AI Memory

We plan two kinds of evidence for the cross-AI claims of §6.

Production statistics. Make0 records which assistant created each memory, which last wrote it, and who wrote each version, as well as daily read and write counts per assistant and brain. From these we will report, in aggregate and with small cells suppressed: (M1) the share of memories created by each assistant; (M2) the share of users whose brain holds memories from two or more assistants; (M3) the share of revised memories, and of corrections, written by two or more assistants; and (M4) the share of each assistant's reads made on brains holding memories another assistant wrote. M4 is measured per brain, not per memory; logging how many returned memories came from another assistant would make it exact.

Cross-Assistant LoCoMo. We will split each LoCoMo conversation's sessions across three simulated assistants that share one answer model, and ask every question through each assistant. Using LoCoMo's evidence annotations, each (question, assistant) pair is labelled *local*, *foreign*, or *mixed* by where its evidence was told. We compare isolated per-assistant memory, a shared brain, each assistant's own pasted history, and all history pasted, all under one judge. A correction track plants 200 updates where the old value is told to one assistant and the new value to another, plus 100 compatible controls, and measures stale answers and false overwrites. We will report accuracy by evidence location, tokens per question, and paired significance tests over three session assignments.

11 Conclusion

Make0 Brain treats long-term memory as a layer the user owns and every assistant borrows. One graph, outside any assistant, is served over MCP to ChatGPT, Claude, Cursor, and Make0's own agent, with separate read and write grants. Because many agents now write to the same memory, every memory and every version records its author, deletions become archives, and only the user can restore or permanently delete, so any change an agent makes can be undone. Underneath, a write path keeps the shared graph clean and a hybrid read path retrieves from it efficiently: 83.64% on LoCoMo in the standard configuration with 85% fewer context tokens than the full conversation, 92.0% in an experimental high-budget configuration, and the right prior task for 92.93% of held-out SWE-ContextBench Lite queries. What a user teaches one assistant is available to all of them, and the user stays in control of it.

References

[1] Anthropic. Model context protocol specification. `https://modelcontextprotocol.io`, 2024.

[2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A human generated MAchine reading COmprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[3] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[4] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[5] LangChain. LangMem: Long-term memory for LangGraph agents. `https://langchain-ai.github.io/langmem/`, 2025.

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[8] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. *arXiv preprint arXiv:2402.17753*, 2024.

[9] Make0. Make0 benchmarks: LoCoMo and SWE-ContextBench. `https://make0.ai/benchmarks/locomo`, 2026.

[10] Mem0. Introducing OpenMemory MCP. `https://mem0.ai/blog/introducing-openmemory-mcp`, 2025.

[11] Memori Labs. Memori LoCoMo benchmark results. `https://memorilabs.ai/docs/memori-cloud/benchmark/results/`, 2026.

[12] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*, 2019.

[13] NVIDIA. Parakeet TDT 0.6B v3. `https://huggingface.co/nvidia/parakeet-tdt-0.6b-v3`, 2025.

[14] OpenAI. New embedding models and API updates. `https://openai.com/index/new-embedding-models-and-api-updates/`, 2024.

[15] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[16] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

[17] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

[18] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[19] Supermemory. Supermemory MCP: Universal memory across LLMs. `https://supermemory.ai/docs/supermemory-mcp/introduction`, 2025.

[20] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv:2002.10957*, 2020.

[21] Jiayuan Zhu, Junde Wu, Minhao Hu, Shengda Zhu, Jiazhen Pan, Weixiang Shen, Yijun Yang, Fenglin Liu, Jianye Hao, Yueming Jin, Qirong Ho, and Min Xu. SWE Context Bench: A benchmark for context learning in coding. *arXiv preprint arXiv:2602.08316*, 2026.