READ THE ORDER, DON'T EXTRACT IT: DERIVING AGENT MEMORY STRUCTURE FROM THE BYTES

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents depend on memory that is organized and served in real time. Their stores differ in what they already record: logs number their lines, example banks delimit their records, and prose does neither. Current systems either fix one configuration for every store or pay a language model to rebuild structure at write time. We introduce MUSTER, which detects that structure before constructing any: two query-independent byte statistics, the share of lines that open with a serial marker and the share of text inside bounded records, decide the retrieval unit, the parent hierarchy, whether an order may be used, and how retrieval slots are shared across channels. The rule needs no labels, no per-store tuning and no generative model at write time, so a memory is ready in seconds as the store grows, and a query costs one ranking pass and one reader call. On MemoryAgentBench the same rule configures thirteen of fourteen stores in three seconds and ranks first among the retrieval and memory systems the benchmark reports at its own reader. On BEAM from 100K to 10M tokens it transfers without re-tuning: it matches the strongest model-built memory under our main reader while building each 10M-token memory in 36 seconds instead of 13 hours, and read together with a model-built memory it scores above both. Agent memory can thus keep pace with the conversation and sit beneath or alongside model-built memories.

1 INTRODUCTION

An agent that answers from memory inherits a store it did not design. The store arrives as bytes (a transcript, a list of assertions, a novel, a bank of labelled examples), and something must decide how to cut it, how to group the pieces, and what to do when two pieces disagree. A fixed chunk size treats every store alike. Building a hierarchy or an entity graph with a language model pays a write-time cost before asking whether the store already records the structure a reader needs.

MemoryAgentBench [1] shows how different stores are. Fact-consolidation stores number their assertions, in-context-learning stores separate their examples with blank lines, and narrative stores do neither. MUSTER treats these as properties it can measure (Section 3.1, Figure 1): serial detects numbered records that expose an update order, and recshare measures whether bounded records already make good retrieval units. The two statistics see no question, gold answer or dataset name and call no model; over fourteen stores they run in three seconds and configure thirteen, while LongMemEval keeps the sessions the benchmark delivers. The read path then combines dense, lexical and parent retrieval with one shared reranker, keeps a minimum number of slots for each channel, resolves conflicts by order as a read-time view, and assembles the selection into the reader's context.

Detect before you construct. The dominant way to handle heterogeneous memory is to build structure with a language model at write time: Mem0 [17] and Honcho [27] extract memories or user models at ingestion, and graph and tree builders [12, 11] spend a model call per document. Retrieving over the raw text can match such structuring [23, 24, 26], and EdgeMem [25] builds memory without a model in one fixed configuration. MUSTER adds the missing decision: it reads from the bytes whether the store already records the structure a reader needs, and then sets the unit, the hierarchy, the use of order and the channel policy from it. Slots read this way match an

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

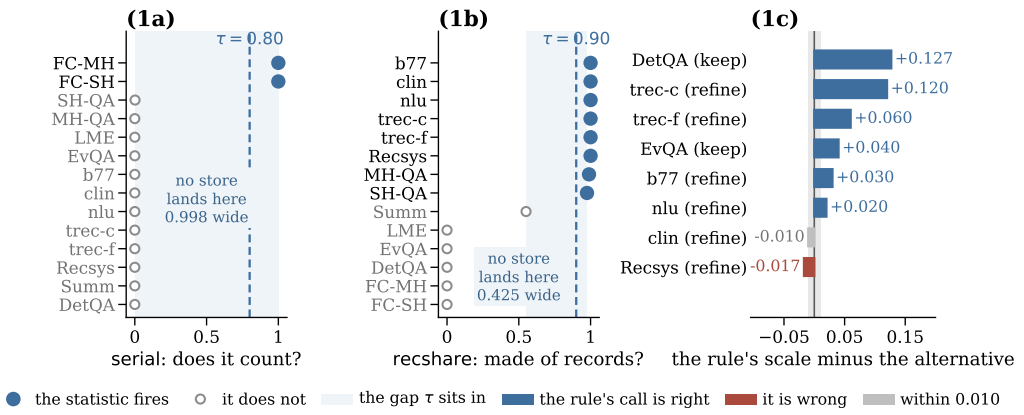


Figure 1: **(1a, 1b)** each statistic over the fourteen stores, filled where it fires, with τ inside a shaded interval no store occupies, 0.998 wide for serial and 0.425 for recshare. **(1c)** in the base configuration at $U = 10$, the scale the rule picks minus the alternative on the eight cells where both were run (Table 17): the rule's call is better on six, within 0.010 on one, and misses one (orange).

entity-aware extractor with no call, and a shuffled order costs more than no order (Appendix E). Adaptive Chunking [2] runs every candidate chunker and scores the outputs; MUSTER runs none. Deterministic conflict resolution [3] assumes the serial from the task format and extracts facts with a model on every query; MUSTER detects the serial, reads slots and order from the bytes, and scores 92.0 and 38.0 on the two fact-consolidation cells against 82.0 and 27.0 at the same reader.

Contributions. **(i)** Detect before construct: two byte statistics, computed in seconds, set the retrieval unit, the parent hierarchy, the use of order and the channel policy of a memory store, with no write-time model, no labels and no per-store tuning, and one reader call per query. **(ii)** First place among the retrieval and memory systems MemoryAgentBench reports at its GPT-4o-mini reader, on every competency, with 60.1 and 64.0 at GPT-4.1-mini and GPT-5-mini, with ablations that measure three of its components: the store-conditioned gate (+1.7), the record-preserving unit (+4.4 on MCC) and the read path over ranking alone (+3.6 points per cell). **(iii)** Transfer to BEAM from 100K to 10M tokens without re-tuning: the rule's fine units beat unrefined 4096-token chunks by 18.6 points at 100K, parity with the strongest model-built memory under our main reader at a fraction of its build cost (36 seconds against 13 hours and 10,429 calls per 10M-token conversation), a 7.8 to 15.3 point lead over a hosted memory service, and the best score at every scale when MUSTER reads a model-built memory alongside its own.

2 RELATED WORK

Structure built at write time. RAPTOR [11] clusters and summarises recursively. GraphRAG [12] and HippoRAG [13] construct entity graphs and communities, and MemoRAG [15] pairs a long-range model holding a global view of the store with the retriever it guides. Each spends a model per document before any query arrives, and each produces structure the store did not have. The benchmark numbers show the cost is not automatically repaid. On the suite we use, the strongest of them, HippoRAG-v2, ties a plain sparse retriever overall (41.6 against 41.5) and trails it by 8.7 on test-time learning (Table 1). Our position is not that constructed structure is useless but that it is the wrong first move when the store already carries structure, and that telling the two cases apart is cheap.

Structure chosen from document statistics. Adaptive Chunking [2] is the closest sibling. Mix-of-Granularity [9] and HiChunk [10] keep several granularities and route between them. All three choose among chunkers, while ours decides whether the store exposes an order to read. That is what the conflict-resolution competency turns on, what no chunk-quality metric can express, and what lets one statistic serve two decisions in Section 3.3.

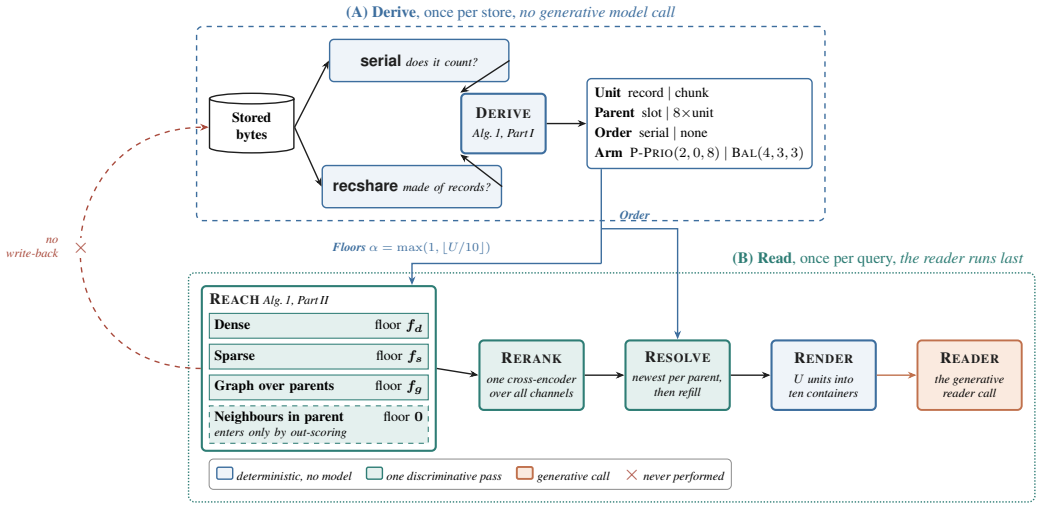


Figure 2: The system (Algorithm 1). Colour marks the class of computation: blue runs no model, teal is the single discriminative pass, orange the reader, the only generative call; the crossed arrow is the write-back the system never performs. **(A)** derives the representation from the bytes; **(B)** answers one query on the base path (refined branch: Section 3.4).

Deterministic supersession. A cluster of 2026 work attacks the same failure we do. StateFuse [5] keeps contradictions visible instead of collapsing them and reports a tie on answer accuracy, [3] applies `max(serial)` deterministically, [4] diagnoses and trains the memory-update gap, [6] revokes stale memories, [7] makes conflict a first-class memory primitive, MemTxn [30] places a transaction boundary around source-supported updates and resolves versions on FactConsolidation, [8] types resolution as a bitemporal operator algebra over the write path, and MemoryLACE [22] has a model write merge, supersession and contradiction relations between atomic memories at ingestion and exposes current, historical and conflicting evidence to the reader. They differ in what to do once a conflict is known, from deterministic selection given explicit metadata to learned updating and lifecycle revocation. Our question comes earlier: where the order comes from, and whether it is there to be read before anything is built. That is why this work detects whether a readable order exists before using it, and resolves as a read-time view. Appendix E holds the records fixed and changes only where the slots come from.

Agentic memory and long context. MemGPT [16], Mem0 [17], Zep [18], MIRIX [19] and Hindsight [29] keep a model in the loop as the session runs, consulting and editing memory as it goes, while ours is decided once per store and shared by every question against it. Long-context readers are the third answer: a 400K model reading the whole store outscores every retrieval system reported, and MUSTER passes it at the same reader (64.0 against 60.5, Section 5).

3 METHOD

We call a retrieval policy an *arm* and the benchmark's fourteen-cell results table the *board*.

3.1 BYTE STATISTICS

A retriever needs a unit, a parent and an order, and the only store content it may consult before the first query is the bytes themselves. So the design question is which properties of bytes decide those three, and two answers turn out to be enough. Whether an order exists is decided by whether the store counts: a leading integer on a line is the one surface form a writing process appends and a reader can trust without interpretation. Whether the store may be cut finer is decided by whether it is made of pieces already small enough to keep whole. The two statistics below are the smallest observables that answer those questions. Neither needs a query, a label, or a model. serial reads the bytes and nothing else; recshare reads the bytes against one reference scale, the chunk the benchmark would

otherwise have used, because the question it asks is comparative rather than absolute. It does not ask how long the records are but whether they already fit inside the cut you were going to make, and that is what lets one threshold license a finer cut on a record store and refuse it on a novel.

$$\textsf{serial} = \frac{\#\{\ell \in L : \ell \text{ matches } \langle \textit{integer} \rangle \,.\, \langle \textit{body} \rangle\}}{|L|}, \qquad L = \text{non-empty lines}, \tag{1}$$

$$\textsf{recshare} = \frac{\sum_{r \in R,\, |r| \le c} |r|}{\sum_{r \in R} |r|}, \qquad R = \text{blank-line records},\ |r| = \text{tokens in } r, \tag{2}$$

where c is the reference scale, the chunk size the benchmark prescribes for the cell and the one operating input the rule takes besides the bytes. The names say what each asks. serial, short for *serial fraction*, asks whether the store counts: a leading integer is the one mark a writing process appends as it goes, so a high value means the store recorded its own order and a low one means no order can be trusted. recshare, short for *record share*, asks whether the store is made of records rather than merely containing some, and asks it relative to the cut we were about to make, namely the share of tokens already in pieces smaller than c. High means the store can be cut finer without splitting a record; low means it is narrative and a finer cut would be arbitrary.

The token weighting in recshare is the whole of its behaviour. Counting records instead puts InfBench-Sum among the record stores on the strength of $1{,}513$ paragraph breaks, when one of those records holds 98.5% of its tokens, and weighting asks whether the store is made of records and separates the fourteen stores into eight at 0.9751 or above and six at 0.5504 or below.

3.2 Derivation rule

> If serial ≥ 0.80, the unit is the record with its serial stripped, the parent is the record's normalised slot, and the order is the serial. Otherwise the unit is the chunk size the benchmark prescribes, cut so a record is never split when recshare ≥ 0.90 and refined one documented step under the same condition (the record-preserving cut), the parent is eight times the unit, halved until at least ten parents remain, and no order is claimed.

This step is DERIVE in Figure 2. Run on two stores it is short enough to follow end to end. FactConsolidation's store arrives as lines of the form `1. Alice's employer is Zynga`, so serial reads 0.9999. The unit becomes one record with the serial stripped, the parent a slot key naming the record's entity and attribute, read by a regular expression (Appendix A), and the order the integer itself. Nothing is chunked, nothing is embedded to decide it, and the parent map is a real partition, so *resolve* has something to resolve by. DetectiveQA's store is a novel, where serial is 0 and recshare is 0, so the base unit is the benchmark's own 4096-token chunk, the parent is eight of those halved until ten remain, and no order is claimed at all, which switches supersession off rather than letting keep-newest delete by an arbitrary key. The two branches share no tuned parameter and neither consults a dataset name; what separates them is a regular expression over lines and a token count.

One cell is outside the rule. The benchmark's loader delivers LongMemEval as conversation rounds grouped into sessions, and we keep that segmentation: the rounds are the units and the sessions the parents. Both statistics are still computed for it, and neither fires, but the rule's unit and parent are not applied. Every other cell, thirteen of fourteen, is configured by the rule alone.

Three constants appear and none is ours to choose: the refinement step moves between the only two chunk sizes MemoryAgentBench itself uses, 512 and 4096, the parent multiplier of eight is the ratio between them, and the back-off floor of ten parents is the retrieval budget, since fewer than ten parents cannot give ten retrieved units a distinct parent each. A store whose unit stays coarse is refined fourfold at read time with four times the units, so the reader sees the same number of tokens (Algorithm 1). That is what lets Section 5 read as a test of the rule rather than a search over its settings.

3.3 Policy gate

The right floor triple is not the same everywhere: a graph-heavy reservation wins where the parent map carries meaning and a dense-heavy one wins where it does not. An earlier version chose between them with a content statistic that sampled slots and asked a model whether their competing

values were mutually exclusive. It works at $\$0.073$ for ten stores, and separates its classes, from the lowest firing store to the highest non-firing one, by 0.147.

It also fires on exactly the stores serial fires on. Using serial for both leaves the board unchanged while widening that separation to 0.998 and removing the cost, so the policy gate now reads the statistic that already chose the representation. A gate whose classes sit almost a unit apart needs no defence of where its threshold went. The content statistic is not discarded but demoted to evidence: a form test and a meaning test, computed at different levels of description, agree on all fourteen stores, which argues for the partition more strongly than either alone. It also removes the last write-time model call, and makes the cost a number: deriving both statistics over all fourteen stores takes three seconds and no generative call, the arm it replaces needs one per unit, $6{,}151$ across the twelve chunked stores alone, and the constructors in Table 1 spend one per document. The gate buys a write path whose cost does not scale with the store.

3.4 REACH: CHANNEL FLOORS

Retrieval runs three channels over the derived units, a dense embedding index, BM25, and a graph joining units that share an entity walked at the parent level, and one cross-encoder scores whatever they return. Pooling the three and ranking the pool is not the compromise it looks like. Where dense recall is strong its candidates take almost every slot, and the lexical or structural evidence that would have answered the harder questions never reaches the reader. *Reach* therefore gives each channel a *reserved floor*, a minimum number of selected units drawn from the channel's own ranking: the slots are first filled by the shared score, and a channel left short then trades the weakest selected unit outside it for its own next candidate (Algorithm 1, step 14). We write a policy as its floor triple (f_d, f_s, f_g) over the dense, sparse and graph channels.

At the base retrieval count, the two policies differ in how they allocate the slots. BALANCED RESERVATION sets $(f_d, f_s, f_g) = (4, 3, 3)$ and then admits a fourth channel, the immediate neighbours of each seed inside its own parent, at floor zero. A neighbour is never given a slot, and enters only by out-scoring every alternative on the shared cross-encoder. PARENT-PRIORITY inverts the emphasis at $(2, 0, 8)$, where two dense anchors represent the query and the parent graph spends the remaining budget on breadth, filled in that priority order. Where the derived parent map has semantic meaning, that breadth is useful, and otherwise the balanced arm avoids paying for it (Table 3).

When the store retains a coarse 4096-token protocol unit, the compensated refinement branch searches at 1024 tokens with four times as many selected units and then merges them into at most ten parent-aware reader containers. The nominal allowance is unchanged, $4096{\times}10 = 1024{\times}40 = 512{\times}80 = 40{,}960$ tokens, so the branch increases retrieval resolution before context assembly rather than buying more nominal reader context. The factor of four is fixed before any run. Full grouping details are in Appendix A.

3.5 RESOLVE: READ-TIME SUPERSESSION

Resolution is a context view, not a memory rewrite. We evaluate the *prune*, *annotate* and *label* views, and ship prune for the benchmark board. *Resolve* applies the derived order within each parent, keeps the largest-order unit and refills freed slots from the remaining candidates in rank order, without re-checking floors or parent identity (Appendix A), so the assembled context emphasizes distinct current evidence rather than repeated versions of the same slot. Where no explicit order was derived, serial resolution is off and the selection is shown as retrieved. The view analysis in Appendix F further shows that history exposure interacts with hop count, separating the store-side decision of whether a readable order exists from the query-side decision of how much history to show.

Resolution happens at read time and is discarded with the context, so the store is never rewritten and a wrong call never deletes evidence from later queries; write-time consolidation amortises its decision, and its error, over every later query.

4 SETUP

We evaluate all fourteen cells of MemoryAgentBench. The benchmark is an aggregate rather than one corpus, because its cells draw on separate public datasets, and the stores they build span 25 units

Algorithm 1 MUSTER. Part I reads the store once with no model; Part II answers a query with one ranking pass and one reader call. $\tau_{\text{num}} = 0.80$, $\tau_{\text{rec}} = 0.90$; P is the benchmark's chunk (512 or 4096; Table 7).

Input: store bytes S (never modified), query q. **Output:** the reader's context for q.

PART I, DERIVE *(once per store)*
serial ← share of lines opening with a serial marker; recshare ← share of text inside bounded records
if serial $\geq \tau_{\text{num}}$ **then**
    unit ← record, parent ← slot, order ← serial
else if recshare $\geq \tau_{\text{rec}}$ **then**
    unit ← record-preserving cut of $P/8$ tokens (at least 512)
else
    unit ← sentence-bounded cut of P tokens; if $P > 512$, refine to $P/4$ and show $4\times$ the units
end if
if not serial: parents ← $8\times$ the unit, halved until at least ten parents remain
channel floors ← parent-priority $(2, 0, 8)$ if serial $\geq \tau_{\text{num}}$, else balanced $(4, 3, 3)$
PART II, READ *(once per query)*
candidates ← dense ∪ lexical ∪ parent walk, each channel with a floor
select the top units after one reranking pass; lift any channel below its floor
if order **then**
    keep the newest unit of each slot (a view, not a write)
end if
group the units into containers and **return** the context to the reader

for a detective novel to 18,333 for an assertion log, a range of 733× that one threshold pair has to cross without per-dataset tuning (Table 6). The benchmark groups the cells into four competencies and we keep its abbreviations of accurate retrieval (AR), test-time learning (TTL), long-range understanding (LRU) and selective forgetting (SF), the last of which is the conflict-resolution family this paper turns on. Each cell keeps the benchmark's own metric, which is substring match except for LongMemEval and InfBench-Sum, which are judged by GPT-4o, and Recsys, which is recall@5. We use GPT-4o-mini, GPT-4.1-mini and GPT-5-mini. Each reader answers 1,671 questions, 300 for LongMemEval, 200 for Recsys, 71 for DetectiveQA and 100 for each remaining cell, so each full fourteen-cell, three-reader configuration rests on 5,013 answers, and twelve of the fourteen cells are scored again under a second protocol. Overall averages the four competency scores as [1] defines them, and read budgets follow Section 3.4.

We also rescore the same answers with Infini Memory's GPT-5 binary accuracy prompt on twelve cells, using the actual question, full reference answer and prediction [20]. Recall@5 and the existing GPT-4o-based Fluency×F1 summarization scores are retained. MAB and Infini-style scores are two evaluations of the same answers. Two retrieval arms are reported, the board's and the gate's; a third, ranking by cross-encoder alone, is an instrument (Appendix F). InfBench-Sum is generated rather than matched, so we measure what re-running costs. Two runs of one configuration give 91 of 100 questions a byte-identical context, none an identical summary, and judged F1 differing by 0.023 (Appendix H). No summarization difference below that is reported here as a result.

5 RESULTS

5.1 MEMORYAGENTBENCH

Read down its mean columns, Table 1 puts MUSTER first among the retrieval and memory systems on all four competencies and on the total, 57.0 against 41.6 for the strongest of them. At the same reader it also beats the whole store read in context, by 13.2 at GPT-4.1-mini (1M tokens) and 3.5 at GPT-5-mini (400K), and at GPT-4o-mini it is 8.2 above GPT-4o and 7.5 above Claude-3.7. Of the 15.4 over HippoRAG-v2, 8.9 comes from the SF competency and 6.5 from the other three.

Two competencies show the statistics acting. Conflict resolution is where serial fires and multi-hop is where it matters. The 38.0 there is the highest value in Table 1, above the 400K reader's 28.0, in a column where fifteen of the seventeen other systems are in single digits. The competency reads 65.0 against that reader's 53.0, and in-context learning is where recshare licenses a finer cut,

Table 1: MemoryAgentBench at a GPT-4o-mini reader except where the row says otherwise, with the other systems as [1] reports them, twelve of its seventeen shown. Deterministic supersession reports only the two conflict cells, and the last five columns are competency means as the benchmark defines them: MCC is the mean of Banking77, CLINC150, NLU, TREC-C and TREC-F, TTL the mean of MCC and Rec, and All the mean of the four competencies. Shading marks MUSTER, and bold marks where it leads the retrieval and memory systems. Long-context readers ingest the whole store.

	AR				TTL		LRU		SF		means				
Method	SH	MH	LME	EvQA	MCC	Rec	Summ	Det	FC-SH	FC-MH	AR	TTL	LRU	SF	**All**
long-context readers															
GPT-4o 128K	72.0	51.0	32.0	77.2	87.6	12.3	32.2	77.5	60.0	5.0	58.0	49.9	54.9	32.5	48.8
Claude-3.7 200K	77.0	53.0	34.0	74.6	89.4	18.3	52.5	71.8	43.0	2.0	59.6	53.9	62.1	22.5	49.5
GPT-4.1-mini 1M	83.0	66.0	55.7	82.6	75.6	16.7	41.9	56.3	36.0	5.0	71.8	46.1	49.1	20.5	46.9
GPT-5-mini 400K	85.0	71.0	63.3	78.2	84.0	13.2	56.3	76.1	78.0	28.0	74.4	48.6	66.2	53.0	60.5
retrieval and memory systems															
BM25	66.0	56.0	45.3	74.6	75.4	13.6	19.0	52.1	48.0	3.0	60.5	44.5	35.5	25.5	41.5
Text-Emb-3-L	54.0	44.0	50.3	70.0	72.4	16.2	18.2	56.3	28.0	4.0	54.6	44.3	37.2	16.0	38.0
Qwen3-Emb-4B	57.0	47.0	43.3	71.4	78.0	12.2	14.8	59.2	29.0	3.0	54.7	45.1	37.0	16.0	38.2
RAPTOR [11]	29.0	38.0	34.3	45.8	59.4	12.3	13.4	42.3	14.0	1.0	36.8	35.9	27.8	7.5	27.0
GraphRAG [12]	47.0	47.0	35.0	34.4	39.8	9.8	0.4	39.4	14.0	2.0	40.9	24.8	19.9	8.0	23.4
MemoRAG [15]	29.0	33.0	20.0	56.0	77.0	13.1	9.2	50.7	21.0	7.0	34.5	45.0	30.0	14.0	30.9
HippoRAG-v2 [14]	76.0	66.0	50.7	67.6	61.4	10.2	14.6	57.7	54.0	5.0	65.1	35.8	36.1	29.5	41.6
Mem0 [17]	25.0	32.0	36.0	37.5	32.4	10.0	4.8	36.6	18.0	2.0	32.6	21.2	20.7	10.0	21.1
Zep [18]	44.0	25.0	38.3	42.5	62.8	12.1	4.2	28.2	7.0	3.0	37.5	37.4	16.2	5.0	24.0
MemGPT [16]	41.0	38.0	32.0	26.2	67.6	14.0	2.5	42.3	28.0	3.0	34.3	40.8	22.4	15.5	28.2
MIRIX [19]	62.0	61.0	37.3	29.8	38.4	9.8	9.9	40.8	14.0	2.0	47.5	24.1	25.3	8.0	26.2
MIRIX 4.1-mini [19]	73.0	75.0	51.0	53.0	61.0	10.3	18.9	62.0	20.0	3.0	63.0	35.6	40.5	11.5	37.7
Det. supersession [3]	–	–	–	–	–	–	–	–	82.0	27.0	–	–	–	54.5	–
MUSTER (ours)	**92.0**	65.0	**56.0**	**80.0**	**81.8**	15.1	**20.7**	62.0	**92.0**	**38.0**	**73.3**	**48.4**	**41.3**	**65.0**	**57.0**

Table 2: Ablations of our configuration under the benchmark's scoring, and the same answers under Infini Memory's scoring. Three readers, all fourteen cells; shading marks the shipped MUSTER. The ablation rows were measured with one additional generative call per query, which the shipped system omits (Appendix B). Infini Memory's variants are as [20] reports them, at the GPT-5-mini backbone and GPT-5 judge that the lower block shares.

Configuration / system	4o-mini	4.1-mini	5-mini
benchmark scoring, ablations of our own configuration			
Base configuration ($U = 10$ throughout)	56.2	60.4	64.1
Global $U = 40$	55.5	60.0	65.4
MUSTER (ours)	57.0	60.1	64.0
Infini-style scoring, against the system that defines it			
Infini Memory-H [20] (GPT-5-mini)	–	–	61.3
Infini Memory-A [20] (GPT-5-mini)	–	–	64.7
MUSTER (ours)	59.5	62.7	**66.9**

worth 4.4 on the MCC aggregate and 1.3 on the competency once the Recsys loss is counted (both measured on the base configuration, Table 17). Long-range understanding is also first among the retrieval and memory systems, 20.7 on summarization and 62.0 on DetectiveQA, where MIRIX at the stronger GPT-4.1-mini reader also scores 62.0, and the 400K long-context reader remains higher on this competency. Both cells lack a serial, so refinement changes retrieval and assembly without enabling supersession; the resolution evidence is the two numbered-assertion stores, and the other twelve carry no serial.

Table 2 shows the same store-derived configuration across three reader strengths. MUSTER reaches 57.0, 60.1 and 64.0 under benchmark scoring. The base configuration is the same derived rule at $U = 10$ everywhere, before the compensated refinement of Section 3.4. MUSTER reaches that level while deriving the configuration automatically in three seconds, with no generative write-time call and one generative call per query. The store-conditioned gate adds 1.7 over either fixed policy (Table 3). Beyond the gate, the record-preserving unit adds 4.4 on MCC (Table 17) and the read path

Table 3: Channel policy at $U = 10$, GPT-4o-mini, views held at prune: the serial gate against each fixed policy.

Policy	AR	TTL	LRU	SF	Overall
BALANCED RESERVATION everywhere	70.3	47.3	42.3	58.0	54.5
PARENT-PRIORITY everywhere	65.2	47.0	40.9	65.0	54.5
Gate: serial ≥ 0.80	**70.3**	**47.3**	**42.3**	**65.0**	**56.2**

adds $+3.6$ points per cell over ranking the same candidates by the cross-encoder alone (Table 16); the rest of the margin over the strongest reported system comes from retrieving over the stored text itself with a hybrid retriever and one cross-encoder instead of over structure a model built. On fixed FactConsolidation selections (no refill, temperature 0), byte-read slots score 41 and 92 with no model call, an entity-aware GPT-4.1-mini extractor 43 and 93 with one call per question, and the recorded order carries the result: shuffling the same serials drops the scores to 33 and 59, below the 35 and 64 of showing no order at all (Appendix E). Which choices were fixed before any score was seen is stated in Appendix I.

The refinement result illustrates why retrieval resolution and reader context should be separated. EventQA begins from base scores of 74.0, 71.0 and 76.0 and reaches 80.0, 80.0 and 82.0 when the coarse store is searched at finer resolution and reassembled under the same nominal token allowance, so the improvement does not rely on a larger reader compensating for coarse retrieval. On stores that already admit a fine unit, MUSTER keeps that resolution instead of paying the refinement step. Appendix B reports the full cell-level breakdown and Appendix G the renderer/item-count controls.

With GPT-5-mini and Infini-style evaluation, MUSTER reaches 66.9 against the reported 64.7 of Infini Memory-A [20], with its largest gains in accurate retrieval (90.0 versus 81.2) and selective forgetting (68.5 versus 58.0), the two competencies store-conditioned retrieval and explicit-order resolution act on; Infini Memory-A is higher on long-range understanding (68.6 versus 60.6). Scoring details and the full cells are in Appendix B.

5.2 ABLATIONS

The serial gate takes the better policy on every competency: balanced floors on accurate retrieval and parent-priority floors on selective forgetting, for $+1.7$ Overall over either fixed policy (Table 3). Appendix G shows recshare choosing correctly in both directions (six of eight cells higher, one within 0.010) and exposing more parent-pure containers raising judged summarization by $+0.042$ ($p = 0.044$) at a fixed allowance.

5.3 TRANSFER TO BEAM

None of the design choices of Appendix I was revisited for BEAM [21]. BEAM prescribes neither a chunk size nor a block count; with $c = 4096$ the rule reads the store as record-shaped (recshare $= 0.986$, serial 0.18) and takes its documented step from 4096 to 512-token units under 4096-token parents (Section 3.2). We report $U = 20$ and 40 units per question (9.7K and 19.5K tokens). Every row uses one reader (GLM-5.3-Flash, maximum effort), our answer prompt C, which asks the reader to use the memory's order and state the dates it uses (Appendix C), and BEAM's official judge (GPT-4.1-mini).

Parity at a fraction of the cost. MUSTER matches Mem0 at every scale (pooled $+0.4$ at U=20, 95% CI $[-1.3, +2.0]$) and leads Honcho by 7.8 to 15.3 points, with no model at write time: per 10M-token conversation Mem0 makes 10,429 calls over 13 hours, while MUSTER builds in 36 seconds (Table 4). At U=40 MUSTER leads Mem0 on contradiction resolution ($+8.6$), information extraction ($+7.0$) and knowledge update ($+4.6$) at 1M, the categories where memory must be managed, and at 10M the lead on contradiction resolution grows to $+17.5$ and on information extraction to $+10.0$ (Table 5).

The fine unit transfers. At U=20 the rule's 512-token units score 74.9 at 100K, where two unrefined 4096-token chunks at a similar budget score 56.3 ($+18.6$; 95% CI $[15.1, 21.9]$); the rule's other branch and a boundary-blind 512-token cut stay within 0.7 (Table 14).

Table 4: BEAM with GLM-5.3-Flash, our prompt C and the official judge (ten-category macro); every row is our own run. U: 512-token units per question; reader context at 1M. Build cost for one 10M-token conversation (memory: text added); build time per conversation. Mem0: mean of two searches. Honcho is hosted: calls estimated from its open-source rules, input at least the tokens sent, memory not exposed (n/o). Last row: U=20 units, then Mem0's.

BEAM system	accuracy 100K	1M	10M	reader context	build cost, 10M conv. LM calls	LM input	memory	build time 100K/1M/10M
MUSTER (U=20)	74.9	72.0	64.2	9.7K	**0**	**0**	**0**	**1s/3s/36s**
MUSTER (U=40)	75.4	73.3	63.3	19.5K	**0**	**0**	**0**	**1s/3s/36s**
Mem0 2.1.0 [28]	75.4	72.3	62.4	8.2K	10,429	≈101M	1.01M	11m/77m/13h
Honcho [27]	67.1	60.7	48.9	20.7K	4.4–11.7K	≥10.8M	n/o	7m/28m/7.8h
MUSTER (U=20) + Mem0	**78.6**	**74.1**	**65.2**	19.5K	10,429	≈101M	1.01M	as Mem0

Table 5: BEAM by category at 1M and 10M (BEAM's ten categories; U: 512-token units per question; Mem0: mean of two searches). Bold: best at each scale.

BEAM system	Abst.	Contra.	Order	Extract	Instr.	Update	Multi-s.	Pref.	Summ.	Temp.	All
1M tokens											
MUSTER (U=20)	86.4	75.9	28.2	79.1	89.5	78.6	75.6	89.6	63.4	53.2	72.0
MUSTER (U=40)	83.2	**77.9**	**29.0**	**81.6**	90.8	**80.7**	76.1	89.6	**70.8**	**53.6**	**73.3**
Mem0 2.1.0	91.1	69.3	28.0	74.6	**94.2**	76.1	**77.0**	**90.7**	69.1	52.5	72.3
Honcho	**97.1**	69.8	22.8	55.5	84.4	67.1	54.3	84.5	34.7	36.2	60.7
10M tokens											
MUSTER (U=20)	90.0	80.0	25.5	**90.0**	**92.5**	75.0	**36.8**	**78.3**	61.5	12.5	**64.2**
MUSTER (U=40)	**95.0**	**82.5**	**28.1**	**90.0**	77.5	70.0	30.6	75.8	65.8	17.5	63.3
Mem0 2.1.0	77.5	65.0	26.1	80.0	**92.5**	**76.2**	33.1	74.4	**65.9**	**32.5**	62.4
Honcho	90.0	71.2	18.0	57.5	82.5	62.5	14.4	57.1	18.2	17.5	48.9

Composition. MUSTER's U=20 units followed by Mem0's memories, 19.5K tokens in all, score 78.6, 74.1 and 65.2, above either system at the same budget (+3.1, +0.8, +1.3), at Mem0's write-time cost.

Other readers and prompts. Under the open-weight Qwen3.5-9B, MUSTER at U=40 scores 67.3 and 64.2 at 100K and 1M against Mem0's 64.2 and 63.2. Under BEAM's direct-answer prompt A with GLM-5.3-Flash, MUSTER at U=40 scores 60.5, 57.1 and 49.6 against Mem0's 60.4, 58.0 and 49.6 and Honcho's 56.1, 49.9 and 40.8 (Table 13). Our Mem0 run is a strong baseline: Mem0's own published answers from its managed platform score 63.1 at 1M and 54.8 at 10M under the same judge, against 72.3 and 62.4 for our run.

6 CONCLUSION

LLM agents need memory that is organized and served as fast as their conversations grow, yet current systems fix one configuration or rebuild structure with a model at write time. MUSTER instead reads a store's structure from its bytes. One derive path configures thirteen of fourteen MemoryAgentBench stores in three seconds and ranks first at the benchmark's reader; on BEAM from 100K to 10M tokens the same rule matches the strongest model-built memory at our main reader while building its memory in seconds instead of hours, and read together with a model-built memory it scores above both. Because deriving needs no model, an agent can re-derive its memory as it writes, and the same reading extends to coding and tool-using agents whose logs carry their own structure. As personal AI agents accumulate months of conversations, logs and tool traces, MUSTER offers a practical path to memory that is ready in seconds, needs no model to maintain, and resolves updates and contradictions by the order the store records. Its advantage grows with the store: against the strongest model-built memory it moves from −0.5 at 100K to +1.8 at 10M, and on contradiction resolution from +8.6 at 1M to +17.5 at 10M.

AI USE STATEMENT

The authors conceived the research question, the MUSTER method (the two byte statistics, the derivation rule and the read path) and the experimental design, directed the implementation and every experiment, and verified the results. Under their direction, we used generative AI tools as assistants in four ways: to draft sections and polish prose, which the authors revised; to surface candidate related work, which the authors read before citing; to assist research execution by writing evaluation, aggregation and plotting code to the authors' specification and operating the experiment harness; and as a sounding board for design alternatives that the authors proposed and chose among. Every number, table cell and figure value was recomputed from the run artifacts and checked by the authors; no generative AI produced datasets, benchmark content or results. The authors take full responsibility for everything the paper states.

REPRODUCIBILITY STATEMENT

The two statistics are given in closed form in Section 3.1 and the rule that consumes them is stated in full in Section 3.2, and neither needs a query, a label or a model call, so a reader can recompute both from the benchmark's released stores. Appendix A lists the retriever components, the reader and judge model identifiers, and the retrieval parameters needed to reproduce every cell. Appendix B reports all fourteen cells for all three readers under both scoring protocols, so that the Base and MUSTER aggregates in Table 2 can be recomputed by hand, and the released harness makes no model call beyond the reader per query. The harness, the per-question BEAM answer and judgment files and the scoring entry point are included in the supplementary material, and the BEAM answer prompt is reproduced in full in Appendix C. Derive timings are on one Apple M3 Ultra; BEAM build timings on one RTX PRO 6000.

ETHICS STATEMENT

This work evaluates retrieval over a public benchmark of synthetic and public-domain documents. It involves no human subjects, no personal data, and no release of new data. The failure mode it reports, that an attribute-keyed extractor drops $2.7\times$ as many records as the byte-read slots, is a reliability concern for deployed memory systems and is reported so that it can be avoided.

REFERENCES

[1] Hu, Wang, and McAuley. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. ICLR 2026; arXiv:2507.05257.

[2] de Moura Júnior, Lelong, and Blangero. Adaptive Chunking: Optimizing Chunking-Method Selection for RAG. arXiv:2603.25333; LREC 2026.

[3] Reddy and Challaram. Reliable Post-Retrieval Assembly for Agent Memory: Separating Evidence Extraction from Policy Execution (v1: Don't Ask the LLM to Track Freshness). arXiv:2606.01435v2, 2026.

[4] Patel. Supersede: Diagnosing and Training the Memory-Update Gap in LLM Agents. arXiv:2606.27472, 2026.

[5] Volkov, Li, and Luo. StateFuse: Deterministic Conflict-Preserving Memory for Multi-Agent Systems. arXiv:2607.05844, 2026.

[6] Zhou, Ouyang, Zheng, and Xiang. TEPA: Revoking Stale Memories for Conflict-Robust Language Agents. arXiv:2608.07429, 2026.

[7] Zhou et al. LatticeMind: A Conflict-Aware Memory Primitive for Multi-Agent Systems. arXiv:2608.08236, 2026.

[8] Ziming Wang. TOKI: A Bitemporal Operator Algebra for Contradiction Resolution in LLM-Agent Persistent Memory. arXiv:2606.06240, 2026.

[9] Zhong, Liu, Cui, Zhang, and Qin. Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation. arXiv:2406.00456, 2024.

[10] Lu, Chen, Qiao, and Sun. HiChunk: Evaluating and Enhancing Retrieval-Augmented Generation with Hierarchical Chunking. arXiv:2509.11552, 2025.

[11] Sarthi et al. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. ICLR 2024.

[12] Edge et al. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv:2404.16130, 2024.

[13] Gutiérrez et al. HippoRAG: Neurobiologically Inspired Long-Term Memory for LLMs. NeurIPS 2024.

[14] Gutiérrez et al. From RAG to Memory: Non-Parametric Continual Learning for Large Language Models. ICML 2025; arXiv:2502.14802.

[15] Qian et al. MemoRAG: Moving towards Next-Gen RAG via Memory-Inspired Knowledge Discovery. arXiv:2409.05591, 2024.

[16] Packer et al. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560, 2023.

[17] Chhikara et al. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. arXiv:2504.19413, 2025.

[18] Rasmussen et al. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. arXiv:2501.13956, 2025.

[19] Wang and Chen. MIRIX: Multi-Agent Memory System for LLM-Based Agents. arXiv:2507.07957, 2025.

[20] Ji et al. Infini Memory: Maintainable Topic Documents for Long-Term LLM Agent Memory. arXiv:2606.10677, 2026.

[21] Tavakoli, Salemi, Ye, Abdalla, Zamani, and Mitchell. Beyond a Million Tokens: Benchmarking and Enhancing Long-Term Memory in LLMs. arXiv:2510.27246, 2025.

[22] Yacoubi, Schmidt, Petrovic, Frikha, Kirchhoff, and Knoll. MemoryLACE: Memory Lifecycle-Aware Consolidation and Evidence Retrieval. arXiv:2609.03201, 2026.

[23] Derehag, Calva, and Ghiurau. SmartSearch: How Ranking Beats Structure for Conversational Memory Retrieval. arXiv:2603.15599, 2026.

[24] An. Fidelity Before Structure: Verbatim Chunks Beat Lossy Artifact Extraction in Long-Conversation LLM Memory. arXiv:2601.00821, 2026.

[25] Cui, Cao, Wen, Yuan, Feng and Chen. EdgeMem: LLM-Free Agent Memory Construction and Retrieval via Evidence-Preserving Multi-Anchor Hypergraph. arXiv:2609.05553, 2026.

[26] Adler and Zehavi. Storage Is Not Memory: A Retrieval-Centered Architecture for Agent Recall. arXiv:2605.04897, 2026.

[27] Plastic Labs. Honcho, hosted service and open-source repository `https://github.com/plastic-labs/honcho` (ingestion runner at commit a1d689b), accessed September 2026.

[28] Mem0 open-source release v2.1.0. `https://github.com/mem0ai/mem0`, accessed September 2026.

[29] Latimer, Boschi, Neeser, Bartholomew, Srivastava, Wang, and Ramakrishnan. Hindsight is 20/20: Building Agent Memory that Retains, Recalls, and Reflects. arXiv:2512.12818, 2025.

[30] Cui, Tang, Yao, Meng, Ma, and Jia. MemTxn: A Transaction Boundary for Source-Supported Updates and Complete-State Recovery in Agent Memory. arXiv:2607.27834, 2026.

A READ-PATH AND EVALUATION DETAILS

Under the record branch of Section 3.2 the unit must be one record and no chunk size delivers that, since FactConsolidation's records span 7 to 30 tokens, so 16 splits the long ones and 32 glues the short ones. The branch emits atoms directly, and at 1.79 records per unit the parent map and the order are both meaningless.

Table 6: The fourteen stores under the base representation, before compensated refinement, so the unit counts of DetectiveQA, InfBench-Sum and EventQA are their coarse base segmentation. Units and parents are counted from the store, not from the benchmark's description of it. The range is 733× in units and 1,067× in parents; one threshold pair crosses it.

Cell	Source dataset	Units	Parents	n
DetectiveQA	DetectiveQA	25	12	71
InfBench-Sum	InfiniteBench	58	14	100
EventQA	EventQA	131	16	100
TREC-fine	TREC	202	25	100
CLINC150	CLINC150	206	25	100
TREC-coarse	TREC	207	25	100
NLU	NLU	208	26	100
banking77	banking77	210	26	100
SH-QA	RULER-style single-hop QA	398	49	100
LongMemEval	LongMemEval	719	89	300
MH-QA	RULER-style multi-hop QA	854	106	100
Recsys	ReDial	2,933	366	200
FC-SH	FactConsolidation	18,333	12,800	100
FC-MH	FactConsolidation	18,333	12,800	100

Table 7: Every constant Algorithm 1 refers to without naming. The two thresholds were fixed before any of the runs reported here and land in empty intervals of the observed distribution, 0.998 and 0.425 wide (Figure 1). Nothing in this table is tuned per cell; the rule reads the store and the table says what the reading maps onto. P is the protocol chunk size of the cell, 512 or 4096; c in Eq. 2 is the reference scale, P, and u the read unit.

Symbol	Value	What it decides	Where
Part I, derived once per store			
τ_{num}	0.80	serial numbering fires above this	§3.2
τ_{rec}	0.90	record-shaped rather than prose above this	§3.2
u	P or $P/8$	the read unit, from τ_{rec}	§3.2
$u_{\min}$	512	the finest unit the rule will cut to	§3.2
mult	$8, 4, 2$	parent layer, coarsest leaving ≥ 10 parents	§3.2
π	$(2, 0, 8)$ or $(4, 3, 3)$	channel floors, parent-priority or balanced	§3.4
U	10	items shown to the reader	§G.3
compensated refinement, when the store forced $u = P$			
u, U	$P/4,\ 40$	finer unit at the same token allowance	§3.4
α	$\max(1, \lfloor U/10 \rfloor)$	floors scale with the budget, $(4, 3, 3) \to (16, 12, 12)$	§3.4
Part II, per query			
R	10	containers the renderer merges down to	§3.4
K_1	200	first-stage candidates handed to the reranker	this section

The token allowance is the same on both branches by construction, $4096 \times 10 = 1024 \times 40 = 40{,}960$ tokens, so the refinement buys resolution and not context.

Two properties of that reservation carry the design. A floor is filled from the channel's *own* ranking rather than credited to any candidate that also sits in that channel's top-K. Under the looser reading a dense unit lying deep in BM25's ordering satisfies the sparse floor by overlap and the lexical best never enters. Floors are enforced after the global selection, largest floor first, the evicted unit is the lowest-scoring selected unit outside the channel whose removal keeps every other channel at its count, ties broken by unit identifier. Candidate selection deduplicates unit identifiers, and the graph uses parents, but the balanced floors do not guarantee distinct parents.

Finer search, assembled context. Small units can retrieve a precise fact while omitting context needed to interpret it. After the base rule, a non-serial store that retains a 4096-token unit is searched

at 1024 tokens with $U = 40$ selected units. Floors scale by $\alpha = \max(1, \lfloor U/10 \rfloor)$ to $(16, 12, 12)$, and the parent scale is recomputed by the same multiplier rule, giving 8192 tokens on the evaluated stores. This branch uses the three channels without the extra neighbour source and applies to EventQA, InfBench-Sum and DetectiveQA, and the other branches retain their base policies, including parent-priority on serial stores. LongMemEval uses its native rounds and sessions rather than the token chunker (Section 3.2).

On the refined branch the forty selected units are grouped into at most ten reader items by deterministic average-linkage merging. The affinity of two units is the number of non-hub entities they share plus, when they share a parent, a bonus one larger than the largest entity overlap in that selection; the initial size cap is $\lceil U/10 \rceil$, relaxed if no legal merge remains; containers are ordered by the best cross-encoder rank they contain and labelled `Memory i:`, each holding its units in stored order. Thus forty units need not form ten groups of exactly four. On the base branch the ten selected units are shown as they are, in stored order and without labels. The nominal content allowance remains $40 \cdot 1024 = 10 \cdot 4096 = 40{,}960$ tokens, while actual prompt length and execution cost can differ. Ten is the benchmark's `retrieve_num`, which its own prompt assembly turns into ten labelled containers for the sparse, dense and graph baselines alike. We keep it so the board compares, and Appendix G.3 prices it. This separates the resolution of retrieval from the context assembled for the reader.

The arm the gate replaces needs one generative call per unit, 6,151 across the twelve chunked stores alone, and the constructors in Table 1 spend one per document.

The dense encoder is BAAI/bge-m3 with maximum sequence length 448; the sparse channel is BM25; the cross-encoder is BAAI/bge-reranker-v2-m3. Both limits bound the text a 4096-token unit exposes to neural retrieval and scoring, so the refined branch is evaluated as a complete configuration, including its interaction with these limits, and not as a granularity effect alone. The refined runs set the first-stage parameter $K_1 = 200$ and a reranker input limit of 1024. Reader identifiers are `gpt-4o-mini-2024-07-18`, `gpt-4.1-mini-2025-04-14` and `gpt-5-mini`. The first two answer at temperature 0.7 with seed 42, the harness default; `gpt-5-mini` is a reasoning model and runs at its default sampling with no output cap, and its answer is scored in full. The additional binary accuracy judge returned `gpt-5-2025-08-07`. Reusing a judgment requires equality of the full question, reference, prediction and request settings. Cell identity is read from each result file's recorded question type and parquet row, never from its filename, and scoring runs through one entry point that owns the metric map, so a cell can neither be scored by the wrong metric nor filled from a run it did not come from.

Rules transcribed from the harness. The serial pattern is `^\s*(\d+)\.\s+(.+)$`. On a serial store the slot key is the record's first capitalised entity span, lower-cased, joined to the record body in which every capitalised span is masked to _; Appendix G reports three coarser keys. Entities everywhere are capitalised spans found by a regular expression, minus hubs, the entities that appear in more than 5% of units. The graph channel represents each parent by one unit, drops entities shared by more than 5% of parents, seeds the parents that hold a query entity with that entity's IDF, walks a bounded number of hops with weight decay 0.5 per hop, orders the reached parents by hop and then accumulated IDF with ties by stored order, offers the first 100, and scores their representatives with the shared cross-encoder; on a serial store the walk runs over slots. *Resolve* keeps, within the selection, the largest-order unit of each parent, drops the others and refills from the arm's remaining ranked candidates in order, without re-checking floors or parent identity; on a serial store each kept unit is shown as `[serial] [CURRENT] text` in serial order. The removed extraction view asked the reader, per query, "From the numbered memory passages below, extract facts ABOUT THE USER relevant to this question: $\langle q \rangle$. Output one per line exactly as: attribute :: value :: passage_number. Include superseded/older values too. Max 25 lines." over the selected units, each cut to 600 characters; the attribute string, lower-cased, is the slot.

B Complete cell results

For the Infini-style comparison we keep Recall@5 on Recsys and the GPT-4o Fluency×F1 score on InfBench-Sum rather than re-judging them, and Infini Memory's number is taken from its paper.

The shipped configuration is evaluated on the same answers under both protocols. Rec is Recall@5 and Summ is the existing Fluency×F1 score. The base and shipped columns compare complete configurations, so their differences are not single-factor estimates; the + extraction column differs from the shipped one only by the read-time extraction view, since removed. Like for like, both with the extraction view, the refinement path adds $+0.3$, $+0.6$ and $+0.5$ Overall to the base configuration. Global $U{=}40$ (Table 2) is higher at GPT-5-mini (65.4) but lower at the other two readers; the rule reads no reader property, so it is not shipped.

Table 8: All fourteen cells with GPT-4o-mini, 0–100 scale. Base, + extraction (the earlier build with the extraction view) and MUSTER (shipped) are MAB scores; Δ is MUSTER minus + extraction. The last column is MUSTER under Infini-style scoring, where Rec and Summ keep their MAB metrics.

Cell	n	Base	+ extraction	MUSTER	Δ	MUSTER Infini
SH-QA	100	90.00	90.00	92.00	$+2.00$	98.00
MH-QA	100	63.00	63.00	65.00	$+2.00$	80.00
LongMemEval	300	54.33	54.33	56.00	$+1.67$	62.67
EventQA	100	74.00	81.00	80.00	-1.00	84.00
Banking77	100	93.00	93.00	96.00	$+3.00$	96.00
TREC-C	100	71.00	71.00	70.00	-1.00	70.00
CLINC150	100	90.00	90.00	90.00	$+0.00$	90.00
NLU	100	90.00	90.00	88.00	-2.00	87.00
TREC-F	100	66.00	66.00	65.00	-1.00	65.00
Rec	200	12.67	12.67	15.08	$+2.41$	15.08
Summ	100	19.84	20.35	20.66	$+0.31$	20.66
DetectiveQA	71	64.79	63.38	61.97	-1.41	66.20
FC-SH	100	92.00	92.00	92.00	$+0.00$	92.00
FC-MH	100	38.00	38.00	38.00	$+0.00$	38.00
Overall		56.25	56.57	57.00	$+0.43$	59.48

Table 9: All fourteen cells with GPT-4.1-mini, 0–100 scale. Base, + extraction (the earlier build with the extraction view) and MUSTER (shipped) are MAB scores; Δ is MUSTER minus + extraction. The last column is MUSTER under Infini-style scoring, where Rec and Summ keep their MAB metrics.

Cell	n	Base	+ extraction	MUSTER	Δ	MUSTER Infini
SH-QA	100	93.00	93.00	91.00	-2.00	97.00
MH-QA	100	77.00	77.00	77.00	$+0.00$	86.00
LongMemEval	300	62.67	62.67	61.00	-1.67	68.33
EventQA	100	71.00	81.00	80.00	-1.00	87.00
Banking77	100	95.00	95.00	97.00	$+2.00$	96.00
TREC-C	100	73.00	73.00	73.00	$+0.00$	73.00
CLINC150	100	90.00	90.00	89.00	-1.00	89.00
NLU	100	87.00	87.00	85.00	-2.00	85.00
TREC-F	100	50.00	50.00	49.00	-1.00	47.00
Rec	200	16.92	16.92	16.92	$+0.00$	16.92
Summ	100	30.66	27.90	29.09	$+1.19$	29.09
DetectiveQA	71	70.42	73.24	67.61	-5.63	73.24
FC-SH	100	93.00	93.00	93.00	$+0.00$	93.00
FC-MH	100	41.00	41.00	41.00	$+0.00$	42.00
Overall		60.35	60.99	60.09	-0.90	62.68

Table 10: All fourteen cells with GPT-5-mini, 0–100 scale. Base, + extraction (the earlier build with the extraction view) and MUSTER (shipped) are MAB scores; Δ is MUSTER minus + extraction. The last column is MUSTER under Infini-style scoring, where Rec and Summ keep their MAB metrics.

Cell	n	Base	+ extraction	MUSTER	Δ	MUSTER Infini
SH-QA	100	92.00	92.00	92.00	+0.00	97.00
MH-QA	100	81.00	81.00	81.00	+0.00	97.00
LongMemEval	300	70.67	70.67	68.67	−2.00	78.00
EventQA	100	76.00	81.00	82.00	+1.00	88.00
Banking77	100	94.00	94.00	95.00	+1.00	95.00
TREC-C	100	84.00	84.00	84.00	+0.00	84.00
CLINC150	100	95.00	95.00	94.00	−1.00	94.00
NLU	100	85.00	85.00	89.00	+4.00	88.00
TREC-F	100	62.00	62.00	59.00	−3.00	58.00
Rec	200	13.83	13.83	13.67	−0.16	13.67
Summ	100	38.41	41.25	38.01	−3.24	38.01
DetectiveQA	71	78.87	77.46	76.06	−1.40	83.10
FC-SH	100	94.00	94.00	94.00	+0.00	92.00
FC-MH	100	44.00	44.00	44.00	+0.00	45.00
Overall		64.12	64.61	63.97	−0.64	66.95

C BEAM ACROSS SCALES: INSTRUMENT AND PROMPT

Instrument. Answer model GLM-5.3-Flash with maximum reasoning effort, called through one pinned provider; judge BEAM's official code, unmodified, with GPT-4.1-mini; score the benchmark's ten-category macro average, aggregated question → conversation → category; intervals from a paired bootstrap over whole conversations, 10,000 draws. Questions: 400 over 20 conversations at 100K, 700 over 35 at 1M, 200 over 10 at 10M. The rule is read once per scale (pooled recshare 0.986 at 100K, 0.993 at 1M and 10M). At 1M and 10M an answer left empty because reasoning used the whole allowance was re-asked once with a larger allowance (at most 47 of 700 rows).

Competitors. Mem0 is the open-source release 2.1.0 [28], run as a local server with the benchmark runner of its authors, returning the top 200 memories with their dates. Its ingestion and search differ from the system described in [17], which the Mem0 row of Table 1 reports as MemoryAgentBench measured it; the BEAM rows are results of release 2.1.0 under this configuration. Honcho is the hosted service [27], called through its session-context API with the ingestion runner of its repository, after its ingestion queue had drained; a hosted service is not version-pinned, so every request and response is included in the supplementary material. past.dev (`https://past.dev/benchmarks/beam`, accessed September 2026) is the hosted service at its lowest recall level; it scores 72.3 through its API under our instrument against 91.8 on its site. Each system received every message with its BEAM timestamp, and each returned context was passed to the same reader with the same prompt.

Table 11: Every BEAM setting we ran, same instrument. U is the number of 512-token units shown; adaptive groups the same units into one container per 4096-token parent (at least ten).

MUSTER setting	context	100K	1M	10M
$U = 20$	9.7K	73.7	72.7	63.1
$U = 20$, adaptive containers	9.7K	74.9	72.0	64.2
$U = 30$	14.6K	74.8	–	–
$U = 30$, adaptive containers	14.6K	75.8	73.3	61.3
$U = 40$	19.5K	76.0	73.9	60.7
$U = 40$, adaptive containers	19.5K	75.4	73.3	63.3

Answer prompt (system message). The user message is `QUESTION:` followed by the question and `RESPONSE:`; the retrieved context precedes it.

> You are the user's personal assistant. You have a memory: excerpts from your earlier conversations with this user, retrieved for the current question. The memory is a record of what was said; the current question decides what to do with it.

Sources
– Everything you state about the user - their projects, decisions, plans, preferences, dates, numbers, and what they told you - must come from the memory. Do not invent, embellish, or infer details the memory does not support.
– If the question is about the user's own history or work and the memory does not cover it, say so plainly and name what is missing, instead of guessing or answering with related material. Do not fill such a gap with general knowledge.
– If the question is a general request - an explanation, a calculation, a recommendation, a how-to - you may use your general knowledge to carry it out, shaped by what the memory says about this user.

Reading the record
– The memory has an order: messages carry serial numbers, and some carry a date and a session/turn index; larger numbers were said later. Use this order for anything about sequence, timing, or what is current.
– When a later statement updates or replaces an earlier one, the latest is current; bring up the earlier one only when the question is about the history or the change.
– When two statements cannot both be true and neither is framed as a change, do not silently pick one: say what each of them says and ask the user which is right.
– Combine excerpts from different parts of the record when the question needs it. When the answer involves dates, durations, counts, or order, state the specific dates or items you used and how you got the result.

Answering the user
– Before answering a request, check the memory for how this user has asked to be answered - format, level of detail, examples, things to include or leave out - and for preferences or constraints that bear on the request. Honor them even when the current question does not repeat them.
– Lead with the answer, and match its length to the question: a short factual question gets a direct one- or two-sentence answer; a request for a summary, explanation, plan, or comparison gets a complete, organized answer that covers each point with the specific details (names, versions, numbers, dates) from the memory.
– Speak to the user directly and naturally. Refer to the memory as your earlier conversations with them, not as excerpts or numbered items.

Build cost at every scale. Table 12 gives the build columns of Table 4 at all three scales. Mem0's calls and tokens come from its server's usage meter and log (one GPT-4o-mini call per ingestion chunk); its store and the source are counted in cl100k tokens from its vector store and the benchmark files. For Honcho we report the tokens we sent and calls estimated from its open-source rules; its model input and store are not observable.

Table 12: Building one memory, per conversation, at each scale (as run). Source: conversation tokens. Model input: tokens Mem0's extraction model read. Store: Mem0's stored memories. Time: wall-clock per conversation with the concurrency we used. † statistics, record units, one embedding pass (bge-m3 fp16 after an 11.5 s model load, or text-embedding-3-small with 8 parallel requests) and BM25, for conversation 0; no generative call. ¶ 1M tokens per call times the 10M call count. § Honcho calls: its open-source deriver rule (one call per user-message batch of at most 1,024 tokens) plus summaries every 20 and 60 messages, applied to the messages we sent; the lower value uses the 4,096-token batch of the cited commit.

		100K	1M	10M
	source tokens	132K	1.05M	11.0M
MUSTER	model calls / input tokens	0 / 0	0 / 0	0 / 0
	build time, one GPU†	0.8 s	3.2 s	35.7 s
	build time, embedding API†	1.9 s	3.6 s	23.5 s
Mem0 2.1.0	language-model calls	143	1,066	10,429
	model input (× source)	1.38M (10.4×)	10.3M (9.8×)	≈101M (9.2×)¶
	memories (tokens, % source)	456 (14.9K, 11.3%)	3,432 (102K, 9.7%)	32,490 (1.01M, 9.2%)
	top-200 read (% of store)	8.7K (58%)	8.2K (8.0%)	8.5K (0.8%)
	wall-clock	11 min	77 min	13.0 h
Honcho	tokens sent	130K	1.03M	10.8M
	model calls (estimated§)	54–155	428–1,190	4,441–11,700
	wall-clock (with queue drain)	7 min	28 min	7.8 h

Mem0 at matched budgets. Re-running Mem0's own search with a larger limit and cutting its list at the reader budgets of 20, 30 and 40 units (9.7K, 14.6K, 19.5K tokens) gives 76.2, 76.0 and 75.5 at 100K; its re-searched top-200 gives 75.6 (the saved search gave 75.2). MUSTER gives 74.9, 75.8 and 75.4 with adaptive containers and 76.0 with ten fixed containers at 40 units. Mem0 does not gain from more context at 100K, where its top-200 is already 58% of its store. At 10M the same budgets give Mem0 62.2, 62.9 and 63.9 against MUSTER's 64.2, 61.3 and 63.3. Mem0's search is not deterministic (a second search overlaps the first top-200 by 0.96, 0.86 and 0.61 at the three scales and scores 75.6, 72.7 and 61.5 against 75.2, 71.8 and 63.2); Table 4 reports the mean of the two.

The rule's other branch. Cutting BEAM-100K the way the rule cuts a store without records (1024-token refined units under 4096-token parents) at 10 and 14 units (10.0K and 14.0K tokens) scores 74.5 and 75.7, against 74.9 and 75.8 for the record units, and a sentence cut to 512 tokens that ignores record boundaries scores 75.6 and 75.8 at 20 and 30 units (date headers survive in both cuts: 202 and 206 of 400 contexts). On BEAM the loss of Table 14's 4096-token row therefore comes from the coarse unit, not from the boundaries: both branches of the rule transfer, and at 512 tokens where each cut falls matters by less than a point.

MUSTER with a model-built store. Appending Mem0's own search to MUSTER's 20 units up to 19.5K tokens (the last row of Table 4) scores 78.6, 74.1 and 65.2 at 100K, 1M and 10M, above MUSTER at 40 units (75.4, 73.3, 63.3) and Mem0 given 19.5K (75.5, 73.0, 63.9) at the same budget; it keeps Mem0's write-time cost. Merging the two into one store (the conversation followed by every Mem0 memory) and letting MUSTER select from it scores 77.0 at 40 units but 73.0 at 20 units at 100K: at a small budget the short memories take slots from the records, so a fixed share for each source works better than one pool.

Mem0's published answers under our judge. Mem0's published BEAM answers from its managed platform (GPT-5 answerer, top-200; the vendor reports 64.1 and 48.6 under its own GPT-5 judge) score 63.1 at 1M and 54.8 at 10M under BEAM's official judge with GPT-4.1-mini. Our open-source run scores 72.3 and 62.4 (mean of two searches).

D OTHER READERS AND BEAM'S OWN ANSWER PROMPT

Table 13 repeats Table 4 with BEAM's answer prompt (A; Listing 44 of [21], a direct answer without explanation) and with the open-weight Qwen3.5-9B, on the same retrieved contexts. Our prompt raises every system, and MUSTER stays within reach of Mem0 under every reader.

Table 13: BEAM under GLM-5.3-Flash with our prompt (C) and BEAM's (A), and Qwen3.5-9B with ours. Judge: the official code with GPT-4.1-mini. Each cell lists 100K / 1M / 10M; Mem0 under GLM with C is the mean of two searches.

Reader, prompt	MUSTER (U=20)	MUSTER (U=40)	Mem0	Honcho
GLM-5.3-Flash, C	74.9 / 72.0 / 64.2	75.4 / 73.3 / 63.3	75.4 / 72.3 / 62.4	67.1 / 60.7 / 48.9
GLM-5.3-Flash, A	56.3 / 56.0 / 49.3	60.5 / 57.1 / 49.6	60.4 / 58.0 / 49.6	56.1 / 49.9 / 40.8
Qwen3.5-9B, C	64.5 / 62.4 / 52.3	67.3 / 64.2 / 51.6	64.2 / 63.2 / 51.8	– / – / –

Table 14: BEAM-100K unit ablation under the Table 4 instrument: the rule's record units against the rule's other branch, a boundary-blind cut at the same size, and unrefined 4096-token chunks. Context in tokens.

unit	≈10K context	≈15K context
record units, 512 tokens (the rule's call)	74.9 (9.7K)	75.8 (14.6K)
the rule's no-record branch, 1024 refined	74.5 (10.0K)	75.7 (14.0K)
sentence cut, 512 tokens, boundary-blind	75.6 (9.9K)	75.8 (14.9K)
unrefined 4096-token chunks	56.3 (8.1K)	61.3 (16.2K)

Unrefined 4096-token chunks. The last row of Table 14 cuts BEAM-100K into sentence-bounded 4096-token chunks, the reference scale without the refinement the rule applies to stores without records (that branch scores within 0.4 of the record units; Appendix C), with parents eight chunks wide and floors scaled to the budget (2, 1, 1 for four chunks; 1, 1, 0 for two). Two chunks (8.1K tokens) score 56.3, 18.6 below 20 units at 9.7K (95% CI $[+15.1, +21.9]$, 20 of 20 conversations); four chunks (16.2K) score 61.3, 14.5 below 30 units at 14.6K ($[+11.2, +17.8]$, 19 of 20). The loss concentrates in information extraction (-37), knowledge update (-35) and temporal reasoning (-42).

E WHERE THE SLOTS COME FROM

On the two FactConsolidation cells, each question's selection (about ten records, each with its recorded serial) is fixed, and only the source of the supersession slots changes. A record is dropped when one of its facts has a newer value for the same slot in another record; the kept records are shown in ascending serial with no refill, and GPT-4o-mini reads them at temperature 0. The byte-read slot is the harness key of Appendix A. The extractors see the same records: the extraction-view prompt of Appendix A, keyed by attribute, and an entity-aware prompt, keyed by entity and attribute: "From the numbered passages below, extract every fact relevant to this question: $\langle q \rangle$. Output one per line exactly as: entity :: attribute :: value :: passage_number. The entity is the subject the fact is about. Include superseded/older values too. Max 25 lines." Without the refill of the shipped read path and at temperature 0, the byte-read row scores 41 on FC-MH where the board's refilled, temperature-0.7 leg scores 38.0.

Table 15: Slot source on fixed selections, 100 questions per cell. Dropped: records dropped per question. Model calls: extraction calls per question.

Slot source	model calls	dropped	FC-MH	FC-SH
Read from the bytes (MUSTER)	0	0.89	41	92
Extraction-view prompt, GPT-4o-mini	1	2.40	37	90
Entity-aware prompt, GPT-4o-mini	1	1.34	42	91
Entity-aware prompt, GPT-4.1-mini	1	0.65	43	93
byte-read slots, order changed				
Recorded serials shuffled among the records	0	0.89	33	59
No order: all records, no serials, random order	0	0	35	64

Reading the slots from the bytes reaches the accuracy of an entity-aware extractor without a model call. The order is what the slots are read against: permuting the same serial values among each question's records (one seed per question) lowers FC-SH from 92 to 59 (37 questions worse, 4 better) and FC-MH from 41 to 33 (10 against 2), below showing every record with no order at all. The attribute-keyed prompt drops 2.7× as many records, because facts about different entities share an attribute string.

F THE VIEW AXIS, AND THE READ PATH AGAINST A RANKING-ONLY CONTROL

Table 16: The base read path against ranking the same candidates with the cross-encoder alone; mean $+0.036$ over thirteen cells. InfBench-Sum has no scored ranking-only run and is excluded.

Cell	rank	full	Δ	Cell	rank	full	Δ	Cell	rank	full	Δ
FC-MH	.140	.380	+.240	SH-QA	.880	.900	+.020	Recsys	.130	.127	−.003
EvQA	.610	.740	+.130	nlu	.880	.900	+.020	LME	.550	.543	−.007
trec-c	.660	.710	+.050	DetQA	.634	.648	+.014	b77	.940	.930	−.010
MH-QA	.600	.630	+.030	FC-SH	.910	.920	+.010	clin	.910	.900	−.010
								trec-f	.680	.660	−.020

The view axis splits by hop count. The three views are produced in one run. *Prune* keeps the newest unit per slot, refills, and renders in ascending serial; *annotate* keeps every unit, in descending serial, and prefixes the newest per slot with [CURRENT]; *label* is annotate with the superseded units prefixed [SUPERSEDED], so annotate and label share unit set, order and budget and both keep the history that removal discards. Each unit carries its serial.

The label recovers about a third of what annotation loses on single-hop, 0.690 to 0.770 against removal's 0.910 in the same run, and *beats* removal on multi-hop, 0.400 against 0.380. That split is the mechanism: a single-hop reader has no use for the superseded value and is only endangered by it, while a multi-hop reader can use the history as evidence for its chain. Adopting the label on both cells would cost the board 1.6, so it does not ship. We report it because it puts the view choice on the hop axis rather than the store axis, where no byte statistic can follow it, and that boundary between representation and assembly is where we would look next.

Table 17: The base refinement decision at fixed $U = 10$ on eight cells, before compensated context assembly. Bold is the higher score, which is also the score the rule selects on every row but Recsys, the one call it gets wrong; CLINC150 differs by 0.010 and is left unmarked.

Cell	recshare	rule says	refined 512	protocol 4096	rule correct?
banking77	1.0000	refine	**0.930**	0.900	yes
trec-coarse	1.0000	refine	**0.710**	0.590	yes
nlu	1.0000	refine	**0.900**	0.880	yes
trec-fine	1.0000	refine	**0.660**	0.600	yes
CLINC150	1.0000	refine	0.900	0.910	-0.010
EventQA	0.0000	keep	0.700	**0.740**	yes
DetectiveQA	0.0000	keep	0.521	**0.648**	yes
Recsys	1.0000	refine	0.127	**0.144**	*no*

G FURTHER RESULTS

The slot rule that groups records into parents turns out not to matter. Four keys, from a copula-only one to the entity-plus-skeleton key the harness ships (Appendix A), which is the registry's key derived from the bytes instead of looked up by dataset name, all sit between 0.360 and 0.390 on FC-MH, the crudest highest. An extractor keyed by attribute alone loses 4 points on FC-MH in the fixed-selection setting (Appendix E).

One cell is budget-bound rather than method-bound, and it shows why an unreported budget matters. InfBench-Sum reads $40{,}348$ tokens at the protocol's ten slots, and doubling the slots reads $79{,}628$ and moves the judged score from 0.388 to 0.477, 63 questions improving against 36. That is $+1.12$ board points bought with twice the benchmark's budget, so it stays out of Table 1 and is reported here as an ablation. Every configuration in Table 2 holds the protocol ceiling instead, refining to 1024 and taking forty units reads $40{,}147$ tokens where ten protocol chunks read $40{,}348$ on the same cell. Nor does any of it turn on how the retrieved blocks are laid out. Rendering every cell of the global $U{=}40$ configuration a second time, by a rule that partitions the same selection into the same ten containers differently, moves the three totals by $+0.38$, -0.20 and $+0.71$ points and splits the fourteen datasets six to six with two exact ties.

G.1 ONE STATISTIC, OPPOSITE CALLS

A statistic that only ever calls one way cannot be checked, so Table 17 in Appendix F reports both calls. recshare licenses a finer cut on six of the eight cells and refuses it on two. Of the six it wins four, is within 0.010 on one and loses one, Recsys, most likely because dialogue turns are short enough to fire the statistic while the preference a question needs spans several turns; we have not tested this. Paired over questions it separates on trec-coarse, $+0.120$ on 18 flips against 6 ($p = 0.023$).

On the two cells where it refuses, keeping the protocol chunk is right by a wide margin at the fixed count the base rule uses: at $U = 10$ finer units cost 0.040 on EventQA and 0.127 on DetectiveQA, an eightfold smaller allowance. The fixed-allowance test is the compensated branch, where EventQA

gains 6.0, 9.0 and 6.0. One statistic therefore produces opposite decisions and is correct in both directions at fixed count.

G.2 What the read path is worth, and where

The read path should justify itself separately from the statistics that configure it. Table 16 in Appendix F ablates it against the same candidates ranked by the shared cross-encoder alone, with no reserved floors, no parent and no order, on the thirteen cells where that instrument has a scored run. The mean is $+0.036$ and two cells hold most of it: FC-MH's $+0.240$ comes from the serial branch, where the parent map is a real partition and the order is read, and EventQA's $+0.130$ is what floors and the parent channels add together, since that cell has no serial. Floors pay where the parent map provides useful candidates, order pays only where a serial is available, and where the map is degenerate a slot reserved for a channel with nothing to contribute displaces a candidate the ranker preferred. A system unable to tell the two cases apart pays that everywhere to collect the $+0.240$, while ours reads which case it is from the bytes.

Table 3 (Section 5.2) measures the same distinction from the policy side. A fixed policy must choose which population to serve, which is why both constants land at 54.5 while trading 5.1 AR points for 7.0 SF points, with smaller TTL/LRU changes. The gate finds no better single policy, since neither dominates, and it removes the need to choose one globally.

G.3 How many items the reader should see

The grouping step of Section 3.4 stops at ten containers because `retrieve_num` is ten, a constant that answers how many units to *fetch* and was never asked how many containers a reader should be *shown*. The two differ here, because the selection touches 16.8 parent segments at $U{=}40$ and 20.0 at $U{=}80$, so 41% of ten containers mix parents where the matched count leaves every container parent-pure.

On InfBench-Sum at a fixed allowance, two contrasts that change the *structure* of the containers while holding the count at ten move the score by $+0.009$ ($p = 0.920$) and $+0.021$ ($p = 0.266$), and the first of them is the widest structural contrast available, two partitions sharing no question's layout, and removing the containers altogether moves it by $+0.005$ ($p = 1.000$). None of the three exceeds the resolution (at most 0.91 of it). Raising the count instead, from ten to the parent number, moves it by $+\mathbf{0.042}$ ($p = 0.044$, $n{=}100$, 60–39–1), 1.84 times the resolution, and doubling the budget with the count matched moves it by $+0.045$ ($p = 0.002$). Within this sweep, container count is the only measured factor whose matched contrast exceeds the re-run resolution, and layout-only contrasts do not.

H Re-run resolution

Resolution. Two runs of one configuration (1024-token units, $U{=}80$, count at the parent number), GPT-5-mini reader, the benchmark's GPT-4o judge at its own temperature 0.1. Identity is the hash of the whole assembled prompt, not a prefix: 91 of 100 match. On those 91 no summary is byte-identical and mean length differs by 0.45% (7,745 against 7,709 characters), so the runs differ in prose, not in length. Recall 0.4147 against 0.3953; precision 0.7439 against 0.7488; fluency 1.000 against 1.000, no question scored 0 in either run. Judged F1 0.4993 against 0.4764, 37–51–3, $p = 0.165$, per-question $\mathrm{sd} = 0.152$. Eighty-eight of the 91 receive a different score. Reader and judge variance both contribute and this pair does not separate them.

I Which choices were fixed before any score, and which were not

Fixed before any run reported here. The two statistics (Equations 1 and 2), the thresholds $\tau_{\mathrm{num}} = 0.80$ and $\tau_{\mathrm{rec}} = 0.90$, the rule of Section 3.2, and the refinement factor of four were written down before the runs in this paper.

Chosen during development on MemoryAgentBench, whose questions are also the reported ones. The two floor triples, the prune view over annotate and label (Appendix F), the shipped

renderer (Appendix G), $K_1 = 200$, and the removal of the extraction view (Tables 8–10). The MUSTER rows of Tables 1 and 2 are therefore development-set numbers, while the other systems were not tuned on these questions. The alternatives we measured bound what the selection can have bought: the label view would have cost 1.6, the second renderer moves a reader's Overall by at most 0.71, and the extraction view by at most 0.9, each far below the 15.4-point margin of Table 1.

Not revisited on BEAM. None of the choices above was changed for BEAM (Section 5.3). Four inputs are set for BEAM: $c = 4096$, because BEAM prescribes no chunk; the unit budget U, reported at 20, 30 and 40, with one container per 4096-token parent (at least ten) because BEAM prescribes no block count; floors scale with U as $(4, 3, 3) \cdot U/10$; and the answer prompt of Appendix C, written for MUSTER's contexts on BEAM's 500K development conversations, which none of the reported scales share. Table 11 lists every BEAM setting we ran.