MIXTURE OF SUB-MEMORIES: SELECTIVE UPDATES FOR CONTINUAL KV-CACHE COMPACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models can acquire knowledge from long contexts, but this approach becomes increasingly impractical in continual settings, where context accumulates over time: prefill cost grows quadratically with sequence length, and the KV cache grows linearly. Cartridges address this by distilling a long context into a small, fixed-size trainable KV cache. In deployment, however, knowledge is not static: new corpora arrive continually, and a memory with a fixed budget must be able to absorb each one while retaining what it already holds. We introduce two complementary approaches that extend training-free KV compaction updates built on attention matching that limit what each update overwrites: (1) a *modular update*, which routes each incoming corpus to one of several sub-memories under a likelihood gate and read one sub-memory per question, and (2) a *monolithic update*, which keeps a single memory but rewrites only a small, attention-selected subset of its slots. On LongHealth and QuALITY with two instruction-tuned models Llama-3.2-3B and Qwen3-4B, the modular update remains above the no-memory baseline at 16,384 slots and is at least as accurate as a text memory of equal size. Its gains on old phases shrinks with every update, and a controlled comparison attributes the loss to repeated recompaction without the source text rather than routing errors or memory size. The monolithic update improves over phases with Llama-3.2-3B and, at 8,192 slots, approaches in-context accuracy on LongHealth and reaches it on QuALITY, but collapses with Qwen3-4B. Our results show that a fixed-size KV memory can survive continual updates when writes are confined, and identify repeated recompaction as the main source of forgetting.

1 INTRODUCTION

Large language models (LLMs) are increasingly asked to reason and answer over long contexts, such as large document corpora, yet doing so is both expensive and unreliable: serving cost grows with the size of key-value (KV) cache (Kwon et al., 2023), and answer quality degrades when relevant information sits in the middle of a long context (Liu et al., 2024). One promising solution is to replace the raw context with a *compressed memory*: a small, trainable KV cache optimized to reproduce the behavior of the full context at a fraction of its size. Cartridges (Eyuboglu et al., 2026) used this idea, distilling a long document into a compact set of key and value vectors through *self-study*, a synthetic question-answering objective that encourages memory to generalize beyond exact memory recall. However training a cartridge, often requires several GPU-hours per context. Attention matching (Zweiger et al., 2026) removes this cost: it selects a subset of the original keys and fits their values using least squares so that the attention over the compacted cache reproduces the attention over the full context, reaching the Cartridges-level compaction and accuracy in minutes rather than hours. Both methods compact to a fixed cache, however real deployments are rarely static: a financial assistant must understand each year's new filings, all without re-encoding everything seen before. A compressed memory with fixed budget must therefore be able to adapt *continually*, *acquiring* each new corpus while *retaining* what it already encodes. The usual stability–plasticity trade-off in continual learning (Lopez-Paz & Ranzato, 2017), is transferred here on a memory whose budget cannot grow. Deciding which of its limited slots should absorb new information and which should be left untouched is a question that existing compressed-memory methods leave largely unaddressed.

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.

In this paper, we study continual KV compaction: documents arrive in phases, each document is discarded once it has been compacted, and a memory of fixed size must answer questions about every documents seen so far. We use attention matching as the compaction method and propose two different ways to limit what each update can overwrite. The first, *modular update*, divides budget into different sub-memories, routes each document to the sub-memory that most increase the model's likelihood of that document, or opens a new one, and answers each question from these sub-memories. The second, *monolithic update* keeps a single memory but let each update rewrite only a small subset of its slots, selected by the attention score and discounted by how often each slots have been written. We evaluated both methods with Llama-3.2-3B and Qwen3-4B on five different long-document datasets, each split into five phases that arrive in sequence. To isolate the effect of restricting writes, we compare against a single KV memory of same slot that is re-compacted in full at every update. Additionally, we also compare compress memory with text baseline, including: no memory, full text in context, concatenated per-document summary, and a recursively rewritten memory. Our results show that confining writes is what keeps a fixed-size memory usable. Specifically, recompacting a single memory at every phase collapses within five phases with both models, whereas the modular update stays above the no-memory baseline at 16,384 slots and is at least as accurate as a text memory of equal size. Its gain on old phases shrinks with every update, and a controlled comparison attributes the loss to repeated recompaction without the source text rather than to routing errors or memory size: the first phase's partition, recompacted four times, is 8.8 to 43.8 points less accurate than the same phase compacted once to the same size. The monolithic update reach in-context accuracy with Llama-3.2-3B at 8,192 slots but collapses with Qwen3-4B. Our contributions are: (1) a continual KV compaction setting in which corpora arrive in phases, are discarded once compacted, and a fixed-size memory is scored on every phase seen so far, (2) two training-free updates built on attention matching that confine what each update can overwrite and (3) evidence that partitioning can prevents the collapse caused by full recompaction, and that repeated recompaction without the source text is the main source of forgetting in a compacted memory.

2 Related Work

Context compression and Compressed memory for LLMs : Long context makes LLMs to become less efficient and less effective. Serving cost increases with the KV cache, which grows linearly (Kwon et al., 2023), and the quality of recall and reasoning degrades as the context lengthens (Liu et al., 2024; Hsieh et al., 2024). As such, both problems motivate the same solution: giving the model a smaller representation of the context to read. Token-space methods shorten the prompt itself by pruning low-information tokens(Jiang et al., 2023; Li et al., 2023). Soft-prompts method train the model to encode the context into a few learned vectors (Mu et al., 2023; Ge et al., 2024; Chevalier et al., 2023). Latent-space methods compress the KV cache itself, either by training a compact cache end to end (Eyuboglu et al., 2026) or by constructing one in closed form so that it reproduces the attention of full context (Zweiger et al., 2026). Both of these methods compress a fixed context once, but how a compressed memory should absorb new corpora without overwriting what it already holds still remain open.

Sequential updates to a bounded memory : A bounded memory that is updated sequentially must decide what to keep, what to let go, and where to place new information as well. Recurrent models make these decisions with input and and forget gates over a hidden state (Hochreiter & Schmidhuber, 1997; Cho et al., 2014), state-space models with selection mechanisms over their state (Mehta et al., 2023; Gu & Dao, 2023; Yang et al., 2025), and Transformer variants with a compressive memory updated at every segment (Rae et al., 2020; Munkhdalai et al., 2024). These methods train the architecture, whereas we attach a fixed-size memory to a frozen pretrained model. The closest to our setting is the recent work that applies latent compaction repeatedly along a single trajectory, either by re-running attention matching (Zweiger et al., 2026; Liu et al., 2026) or by training a compactor once so that each compaction is a single forward pass (O'Neill et al., 2026). Our work measure it in a different form, in which the memory must be able to retain many different corpora and is scored on every earlier corpus after every update, and we adapt two ideas to limit what each update can overwrites. Mixture of experts (Shazeer et al., 2017; Dai et al., 2024) routes each input to one of several expert sub-networks. Specifically, our gated update routes each document to one of several sub-memories, using a likelihood gate in place of a learned router. Sparse memory finetuning (Lin

et al., 2025) updates only the slots of a memory layer that new data activates strongly, our sparse method updates only the KV slots that new data attends to strongly.

3 METHODOLOGY

In the continual learning setting we study, documents arrive in phases $t = 1, 2, \ldots$ and are compacted in sequence (Appendix D). We write $\mathcal{D}_t$ for the documents of phase t, which the gated update and the KV baselines compact as one text. A frozen language model p_θ is augmented with a compressed KV memory of at most B slots, which after phase t must support questions about every document observed so far. Each slot holds the key and value of one token at every layer and every key-value head. The documents $\mathcal{D}_t$ are discarded after phase t, so the memory is the only record of earlier phases. We propose two methods, the gated update (Section 3.1) and the free-form sparse continual compaction (Section 3.2).

Both proposed methods and the two KV baselines (Section 3.3) use the same compaction operation, attention matching (Zweiger et al., 2026). Prefilling the text to compact produces a teacher cache (K_T, V_T). For the reference queries Q defined below, the output of the teacher cache is $Y_T = \mathrm{softmax}(QK_T^\top/\sqrt{d})\,V_T$, with d the head dimension. Within each layer and each key-value head, the keys with the largest root mean square σ_j of their attention weights across Q are retained. Attention matching also fits a bias on the attention logit of each retained key, which we set to zero. Each method specifies the RoPE positions assigned to the retained keys. With A the attention from the reference queries to the retained keys and V^{old} the values the retained slots held before, the values V of the retained slots are obtained from

$$V^* = \arg\min_V \left\|AV - Y_T\right\|_F^2 \;+\; \lambda\left\|V - V^{\mathrm{old}}\right\|_F^2, \tag{1}$$

in which the second term biases each value toward the value previously stored in the same slot, and thus toward the existing memory. The sparse method (Section 3.2) sets $\lambda = 0.01$, whereas the gated update sets $\lambda = 0$.

The reference queries Q come from self-study (Eyuboglu et al., 2026), which has two steps. First, the model reads a text and writes conversations about the text. Each conversation is then prefilled with a KV cache as prefix, and the post-RoPE query vectors of the conversation tokens form Q. The notation below names the text and the cache separately, since the two can differ.

Notation. $\ell(X \mid P) \triangleq |X|^{-1} \sum_i \log p_\theta(X^{(i)} \mid P, X^{(<i)})$ denotes the average log-likelihood per token of the input X, conditioned on the prefix P. $\ell(X) \triangleq \ell(X \mid \emptyset)$ denotes the same quantity with an empty prefix. We write $\mathrm{KV}(D)$ for the key-value cache obtained by prefilling a text D, and $\mathrm{KV}_P(D)$ for the cache of D prefilled with P as prefix. $\textsc{Compact}(C, Q, s)$ denotes the compaction above, applied to a cache C with reference queries Q and a target of s slots. $\textsc{SelfStudy}(P, X)$ returns the query vectors of conversations that the model writes with the text or cache X as context, recorded with the cache P as prefix. $\textsc{SelfStudy}(P)$ abbreviates $\textsc{SelfStudy}(P, P)$ and applies when no document text is available.

3.1 MODULAR UPDATE

The gated update uses a likelihood-based gate to assign the documents of each phase to one of several partitions that share the slot budget.

Memory. The memory is a list of partitions $P_1, \ldots, P_m$, where partition P_j is a compacted KV cache of $|P_j|$ slots covering n_j document tokens and $\sum_j |P_j| \le B$. Retained keys keep the RoPE positions of their source text. Algorithm 1 in Appendix A gives the complete procedures.

Write. For every partition the gate computes the likelihood gain $g_j = \ell(\mathcal{D}_t \mid P_j) - \ell(\mathcal{D}_t)$ in nats per token (line 2). When the largest gain exceeds the threshold $\tau = 0.05$, $\mathcal{D}_t$ is merged into the partition P_w with the largest gain (line 3). Otherwise, the gate allocates a new partition P_w for $\mathcal{D}_t$.

Before each compaction, rebalancing recomputes the slot targets s_j (line 9), which are proportional to n_j, at least 16 slots each and sum to B. Allocating a new partition compacts $\mathcal{D}_t$ alone to s_w, whereas merging prefills $\mathcal{D}_t$ through P_w and compacts the union to s_w (line 10). Rebalancing then recompacts every other partition with a decreased slot target, using $\textsc{SelfStudy}(P_j)$ as query source (line 12).

Query sources. Gated update follows the self-study configuration of Zweiger et al. (2026), with four prompts about the text and a fifth conversation that asks for the text verbatim (Appendix B.1). When the gate merges $\mathcal{D}_t$ into P_w, the reference queries combine $\textsc{SelfStudy}(C, \mathcal{D}_t)$ with $\textsc{SelfStudy}(P_w)$, recorded before $\mathcal{D}_t$ is prefilled, so that the compaction also fits what P_w already holds (line 8). We also consider a variant that takes all merge queries from $\mathcal{D}_t$ (Appendix B.1).

Values. Gated update and the KV recursive baseline (Section 3.3) compact the layers in order, following the on-policy compaction of Zweiger et al. (2026). When $\mathcal{D}_t$ is written, the queries for each layer are recorded again with the layers below already compacted. The value fit of Equation 1 thus uses the queries that each layer produces when the compacted memory is read.

Read. To select a partition, the router scores each question q under every partition by the total log-probability of the question tokens (line 14). The answer is generated with the highest-scoring partition as prefix (line 15).

3.2 Monolithic update

The second proposed method maintains a single memory and, instead of updating by partitions, restricts each write to a small subset of slots; the remaining slots preserve the representation of earlier documents.

Memory. The memory is a single compacted cache of B slots. At phase 1 each document is compacted alone; at every later phase each document is prefilled through the current memory and the union is compacted. The query sources match the two variants of Section 3.1, using either the document conversations alone (*KV recursive doc*) or pooling them with memory-side conversations (*KV recursive doc+mem*). Every question is answered under the whole memory, so KV recursive is the gated update with a single partition. which COMPACT reduces memory to B slots, called K_M and V_M. From phase 2 onward, each arriving document D_i of that phase is appended into the B-slot memory. which COMPACT reduces memory to B slots, representing the new K_M and V_M.

Write. A compaction to B slots allows the current document to overwrite the representation of earlier phases. With the intuition to avoid forgetting knowledge from previous stage, we selectively updates to only the top-important slots regarding to the current stage context. Hence, each write after phase 1 modifies only $|S| = t$ of the B slots per layer. Each layer selects its writable set $S \subseteq \{1, \ldots, B\}$ by ranking slots with an attention score a_j computed from the reference queries with the current, before-update memory.

Values. For continual compaction, we define a deviation penalty and set the trust-region weight to $\lambda_\Delta = 0.01$. Empirically, this regularization improves continual performance relative to an unregularized write. Intuitively, penalizing large changes to the previously stored values encourages each update to incorporate new information without unnecessarily perturbing the existing memory, which is consistent with the goal of reducing forgetting across successive phases.

Comparison. Both methods write into a fixed slot budget, apply COMPACT with self-study reference queries, and obtain values from Equation 1. The gated update partitions the budget, routes each document to one partition, answers each question from the highest-scoring partitions, sets $\lambda = 0$, and gives each retained key its source-text position. Maintaining a single memory, the sparse method restricts each write to the slots that the attention score selects, answers every question from the memory, sets $\lambda = 0.01$, and gives each retained key its slot index.

3.3 Baselines

We compare against two KV baselines that isolate parts of the gated update, three baselines that keep text in the prompt, and a model without memory. **KV recursive** maintains a single cache of B slots, prefills each new phase through the cache, and recompacts the union, making it equivalent to the gated update with a single partition. **KV concat** compacts each phase independently into 4,096 slots and appends the cache, and thus equals the gated update with one partition per phase, no rebalancing and all partitions read together. **In-context** places all documents seen so far in the prompt and, once the model's context window is full, shortens the longest documents first. Thus, the prompt for the in-context baseline is truncated to the context window limit. **Text concat** summarizes each document using a share of a fixed token budget per phase proportional to the document length and places all

summaries in the prompt. **Text recursive** rewrites a single text memory of at most B tokens after every phase using the current memory and summaries of the new documents, with one entry per document. **No memory** gives the model only the question and the answer options.

4 EXPERIMENTS

4.1 DATA

In our setup, we used five datasets each divided into five phases. A phase is a corpus of documents that arrive as a unit: the memory ingests phase 1, then phase 2, and so on, and is scored on every phase after every update. Table 4.1 summarizes the phase composition and question counts for each dataset, and appendix D describes each dataset.

Table 1: Dataset statistics. Each dataset is split into five phases that arrive in sequence.

		Tokens		Questions		
Dataset	Documents per phase	Per phase	Total	Per phase	Total	Answer format
LongHealth	4 patient records	47.2k	236,124	80.0	400	5-way MC
FinQA	13–15 companies	47.1k	235,635	147.4	737	Numeric
QuALITY	7 stories	54.9k	274,675	124.4	622	4-way MC
QASPER	16 papers, one topic	92.1k	460,595	60.8	304	Free-form
TechQA	98–100 technotes	151.9k	759,611	122.0	610	Free-form

We evaluate both proposed methods on the five-phase document sequences with Qwen3-4B and Llama-3.2-3B. Section 4.2 describes the evaluation setup. For the gated update, we first test partitioning, routing and rebalancing on the five-document sequences at 1,024 and 16,384 slots (Section 4.3), then examine merging on sequences of 16 to 24 single-record documents (Appendix B.7), and compare with text and in-context baselines (Appendix B.8). For the sparse method, we sweep the writable capacity and the usage penalty (Section 4.4).

4.2 EXPERIMENTAL SETUP

In the main text, we evaluate gated update on LongHealth and QuALITY, the two datasets with multiple-choice questions. Appendix B.10 reports FinQA, on which every method, the in-context baseline included, reaches less than 10% accuracy, and QASPER and TechQA, which have free-form answers. We split the documents of each dataset into five phases, with four patient records per phase on LongHealth and seven stories per phase on QuALITY. The two resulting sequences, one per dataset, are the five-phase sequences. We run Qwen3-4B[1] and Llama-3.2-3B[2], at memory budgets of 1,024 and 16,384 slots. For the sparse method, we measure log-perplexity on FinQA, QASPER, QuALITY and TechQA at $B = 512$ slots (Section 4.4).

To measure what the memory retains, we ask the model, after each phase, the multiple-choice questions about every phase seen so far. The router selects a partition from the question text alone, so we add the story title to each QuALITY question, as LongHealth questions already name the patient (Appendix B.2). Every method, the no-memory baseline included, receives the same questions, and the model answers by greedy decoding with at most 512 new tokens. We report three quantities per method. *Acc* is the final accuracy on each phase after the last phase, averaged over the five phases. *Fgt* is forgetting for each of phases 1 to 4, the highest accuracy that phase reached after any earlier update minus its accuracy after phase 5, averaged over the four phases, a negative value means the phase is answered better at the end than at its peak. *Unp* is the fraction of replies without a parasable answer, which count as wrong in Acc. To separate memories that prevent usable answers from memories that hold little information, we define collapse. We call a method collapsed when at least 90% of the replies after phase 5 give no parsable answer or at least half are repetition loops, strings of symbols or empty. As every table entry comes from a single run, we report 95% bootstrap intervals over questions for the difference in final accuracy between two methods (Appendix B.2).

[1] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

[2] https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

4.3 PARTITIONED MEMORY

The idea behind gated update is that a new phase, compacted alone from the text into a new partition, leaves the information of earlier phases in their own partitions. On the five-phase sequences, no likelihood gain exceeds τ, so the gate opens a new partition at every phase and never merges (Appendix B.4). After phase t, the memory thus holds one partition per phase, and at each new phase, rebalancing recompacts every older partition to a smaller slot target. We therefore test partitioning, rebalancing and routing on the five-phase sequences and test merging separately in Appendix B.7. Without continual updates, routing each question to a separate per-record cache is more accurate than reading four caches together, and at 16,384 slots it comes close to placing the records in the prompt (Appendix B.3); reading one partition per question is therefore the read procedure throughout. On the five-phase sequences, we then compare gated update with KV recursive, which holds all phases in a single cache (Section 4.3.1). Finally, we ask whether routing errors (Section 4.3.2) or recompaction (Section 4.3.3) limits the accuracy on old phases.

4.3.1 PARTITIONING AVOIDS COLLAPSE

With continual updates, we run gated update and KV recursive on the five-phase sequences at both budgets. KV recursive is gated update with a single partition of B slots, so that every new phase is prefilled through the one compacted cache and all phases are recompacted together (Section 3.3). Comparing gated update with KV recursive thus isolates the effect of partitioning. The no-memory baseline, in which the model sees only the question and the answer options, shows how much the memory adds. Table 2 also lists the text and in-context baselines, which we discuss in Appendix B.8.

Within five phases, KV recursive collapses or becomes less accurate than the chance level, except with Qwen3-4B on QuALITY at 1,024 slots (Appendix B.8). In contrast, at least 75% of the replies of gated update give a parsable answer (Figure 1; Table 2).

Table 2: Final results after phase 5. Acc, Fgt and Unp are defined in Section 4.2, all in %. Size is in slots for KV memories and tokens for text memories; for the monolithic update, t is the number of slots rewritten per update. Appendix Tables 8 and 9 give 95% intervals.

		LongHealth						QuALITY					
		Qwen3-4B			Llama-3.2-3B			Qwen3-4B			Llama-3.2-3B		
Method	Size	Acc	Fgt	Unp	Acc	Fgt	Unp	Acc	Fgt	Unp	Acc	Fgt	Unp
Fixed size													
Gated update	1,024	28.0	4.4	4	22.3	4.7	25	32.5	−0.6	24	32.3	1.2	2
Gated update	16,384	**33.5**	18.1	7	32.0	6.9	10	**42.3**	8.1	7	**36.2**	5.4	6
Monolithic update	512, t=32	9.0	2.8	74	29.1	−1.2	0	9.5	0.2	78	30.8	0.5	3
Monolithic update	512, t=64	8.1	7.5	76	28.6	0.3	0	11.1	4.2	74	31.3	0.8	3
Monolithic update	512, t=128	8.4	5.6	73	28.3	0.9	0	10.6	−1.8	73	31.9	−0.5	4
Monolithic update	8,192, t=512	6.5	0.0	71	33.6	2.8	0	10.4	−1.0	73	35.2	0.7	11
Monolithic update	8,192, t=1024	6.6	−0.3	70	**33.8**	1.2	0	9.1	−1.2	75	35.5	1.3	11
Monolithic update	8,192, t=2048	6.4	0.6	72	32.8	−0.3	0	9.6	0.0	75	34.7	0.9	11
KV recursive	1,024	0.0	29.4	100	1.0	27.8	97	35.4	3.4	1	4.3	20.6	87
KV recursive	16,384	0.3	19.7	99	0.0	16.3	100	0.3	18.7	99	0.0	15.8	100
Text recursive	1,024	23.8	10.0	11	11.3	9.1	64	**36.3**	4.1	0	13.2	14.3	60
Text recursive	16,384	**35.8**	34.1	6	**12.8**	18.8	59	34.1	18.1	0	**24.0**	11.9	36
Growing													
KV concat	20,480	18.5	29.4	20	28.8	7.8	3	7.7	23.7	74	33.9	2.6	0
Text concat	5,120	34.5	1.9	23	34.0	−1.3	9	41.8	1.4	4	32.0	3.8	5
Text concat	20,480	**47.0**	−0.6	22	**38.8**	−1.6	7	**47.8**	4.4	4	**34.1**	1.4	4
References													
In-context	sequence	**60.8**	22.5	5	**38.5**	16.3	4	**45.1**	18.7	1	**34.1**	8.4	1
No memory	0	30.5		12	25.5		20	41.6		0	31.1		2

At 16,384 slots, gated update is more accurate than the no-memory baseline after every phase. Averaged over both datasets and models, the gain over the no-memory baseline decreases from 10.4% after phase 2 to 3.8% after phase 5 (Appendix B.4). At 1,024 slots, gated update is at most 1.3%

more accurate than the no-memory baseline, with an interval that includes 0 (Appendix Table 8). We test in Sections 4.3.2 and 4.3.3 whether routing errors or a loss of information in the partitions cause the gain to decrease.

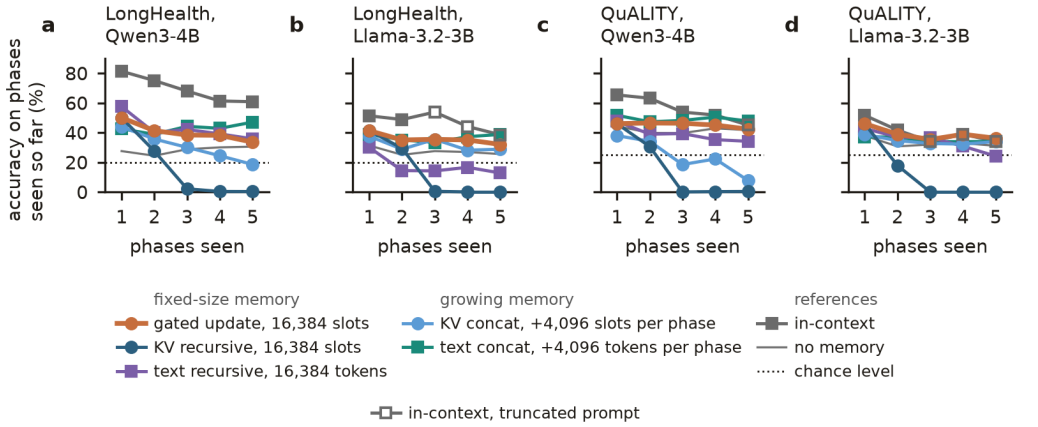


Figure 1: Mean accuracy on the phases seen so far, with hollow markers for a truncated in-context prompt. Table 8 gives 95% intervals.

4.3.2 Routing errors cost little final accuracy

The gain over the no-memory baseline could decrease for two reasons. Either the router selects the wrong partition, or the correct partition has lost the information that the question asks about. To separate the two causes, we reread every memory of Section 4.3.1 after phase 5 with oracle routing and compare with the likelihood router. Oracle routing selects the partition of the question's phase and thus removes routing errors. We also give the model all five partitions concatenated as in KV concat, to test whether reading one partition per question matters.

Oracle routing changes final accuracy by at most 4.1% at either budget (Appendix B.6), so routing errors cost little final accuracy. Reading one partition per question nevertheless remains necessary, as in the test without continual updates (Appendix B.3). With all five partitions read at once, final accuracy decreases by up to 42.1% with Qwen3-4B and changes by at most 9.3% with Llama-3.2-3B (Appendix B.6).

As oracle routing leaves final accuracy almost unchanged, the partitions limit accuracy, for different reasons at the two budgets. At 1,024 slots, the router already assigns at least 80% of the questions of each phase to the partition of the phase. Even under oracle routing, however, the memory after phase 5 is at most 2.6% more accurate than the no-memory baseline. Partitions of about 210 slots, a fifth of the budget, are thus too small to retain four patient records or seven stories (Appendix B.5).

At 16,384 slots, the router assigns only 0.0% to 55.6% of the questions about phase 1 to the partition of phase 1 (Appendix B.5). Oracle routing nevertheless raises final accuracy only on QuALITY with Qwen3-4B, which suggests that the partitions of old phases no longer hold the information that the questions ask about. We test in Section 4.3.3 whether recompaction removes the information.

4.3.3 Accuracy on old phases decreases with the number of recompactions

Rebalancing recompacts every older partition whenever the gate opens a new partition, so by the end of phase 5, the partition of phase i has been recompacted $5 - i$ times. As documents are discarded after their phase, each recompaction starts from the current contents of a partition. Any information that a recompaction removes is therefore lost for the rest of the sequence.

To measure the loss at 16,384 slots, we compare the accuracy on each phase at two points. Right after phase i, the partition of phase i has been compacted once from the text. After phase 5, we read the memory with oracle routing, which removes routing errors (Section 4.3.2). The recompaction loss of phase i is the accuracy on phase i at the first point minus the accuracy at the second point. Across phases, the loss grows with the number of recompactions on LongHealth with both models and on

QuALITY with Llama-3.2-3B, and reaches 12.5% to 48.8% on phase 1 (Figure 2a). With Qwen3-4B on QuALITY, the loss is at most 4.0% on every phase. After four recompactions, the partition of phase 1 is less accurate on phase 1 than the no-memory baseline (Figure 2b).

Since recompaction also reduces the partition of phase 1 to about 3,300 slots, we compact phase 1 once from the text to the same size. The resulting cache is 8.8% to 43.8% more accurate on phase 1 than the recompacted partition (Appendix B.6), so the smaller size does not explain the loss. Such a cache is even about as accurate as the 16,384-slot memory right after phase 1. As every recompaction in gated update also lacks the text of the documents, our experiments do not separate the number of recompactions from the absence of the text. Fewer recompactions reduce the loss from either cause, which makes the number of recompactions per partition a direct target for future methods.

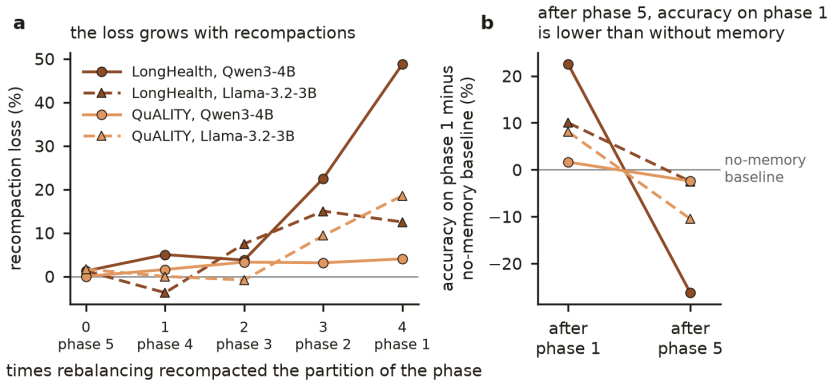


Figure 2: At 16,384 slots, the recompaction loss grows with the number of recompactions, except with Qwen3-4B on QuALITY. (a) Recompaction loss of each phase (Section 4.3.3). (b) Accuracy gain on phase 1 over the no-memory baseline, measured right after phases 1 and 5.

4.4 Monolithic memory

Gated update limits interference by assigning documents to separate partitions. We also investigate a different form of selectivity: maintaining one shared KV memory of (B) slots while allowing each subsequent rewrite to only (t) slots per layer (Section 3.2). This experiment tests whether restricting writes within a shared cache can perform similar to when no partitioning the memory. The results vary substantially by model, so we treat the monolithic update approach as an exploratory alternative rather than a general replacement for gated update, and leave the investigation for model collapse for later studies.

4.4.1 Scaling

Increasing the memory size from 512 to 8,192 slots, a 16× increase, changes the accuracy averaged over update ratios by 1–6% in each of the four model–dataset settings (Appendix Figure 13). Compared to the partitioned memory, where oracle routing gains 16–21% from 1,024 to 8,192 slots (Appendix B.3), scaling in this monolithic memory doesn't give clear improvement and still affects by model collapse. Therefore, we spend the next parts investigating on these two memory sizes instead of scaling more.

4.4.2 Selective update

We test on budget size of 512 and 8192. For 512, we selectively choose a subset of 128, 64, 32 to update; and for 8192, we do 2048, 1024, 512 - both follow the update ratios of 1:4, 1:8, 1:16.

After phase 5, the monolithic update is comparable to in-context accuracy with Llama but collapses with Qwen (Table 2). At (B=8192), Llama reaches 32.8–33.8% on LongHealth and 34.7–35.5% on QuALITY across the three settings. Its highest QuALITY result, 35.5% with (t=1,024), even surpasses truncated in-context result by 1.4 percentage points. On the other hands for Qwen, at (B=8192), its accuracy is 6.4–6.6% on LongHealth and 9.1–10.4% on QuALITY, compared with

30.5% and 41.6% without memory. Across both budgets, 70–78% of its replies have no parsable answer, as it frequently refuses to know the answers such as ”There is no mention of Anne or any scenario involving pregnancy in the provided story” Across the Fgt metrics, no update ratio consistently outperforms the others in knowledge retention. Besides the outliers in Qwen budget 512 on Longhealth, Fgt is smaller than 4.2 and occasionally between -2 and 3.

With a more fine-grained display for information. Monolithic updates tend to increase over phases while the long ICL tends to decrease, either because of long-context or truncation. For Longheatlh, Budget 8192 is able to reach the performance of ICL as we progress into the final stage. Or for Quality, it is able to match ICL performance from stage 3 and forward.

With a more fine-grained display for information Figure 3. For Llama, shared-memory accuracy generally increases over phases, while in-context accuracy declines. On LongHealth, the gap narrows until the two are close after phase 5; on QuALITY, shared memory reaches the in-context range around phase 3 and remains there. The smaller 512-slot memory shows a weaker and less consistent progression, particularly on LongHealth. On the other hand, Qwen already collapses at phase 1. It generally improves after the first phase, but soon plateaus far below its in-context performance. Across these trajectories, no update ratio consistently outperforms the others in knowledge acquisition.

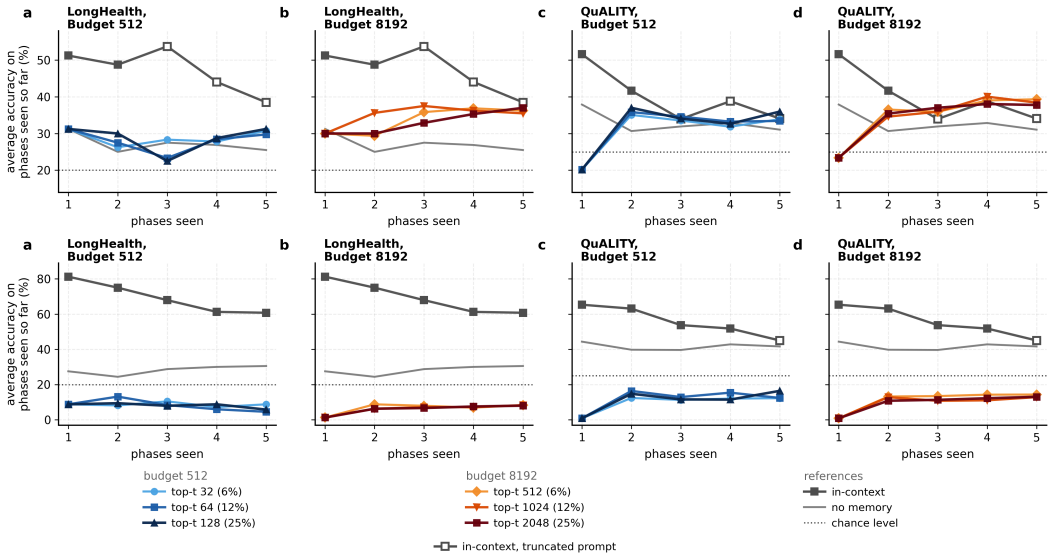


Figure 3: Mean accuracy on the phases seen so far; the top row shows Llama-3.2-3B, the bottom row shows Qwen3-4B, and hollow markers indicate truncated in-context prompts. At 8,192 slots, selective sparse updates approach in-context accuracy with Llama-3.2-3B as phases arrive, whereas Qwen3-4B remains below the no-memory baseline at both budgets.

5 Conclusion

In this paper, we studied continual KV-cache compaction under a fixed budget, where new documents must be incorporated after earlier source documents have been discarded. We examined two ways to limit interference: gated updates to separate sub-memories and selective writes to a single shared cache. At 16,384 slots, gated update avoids the severe failure of fully recursive KV compaction across both models and datasets, although its advantage over no memory narrows over stages. With Llama-3.2-3B, the shared-cache method at 8,192 slots generally gains accuracy as documents arrive and finishes close to the 16,384-slot gated results on both datasets. However, shared-cache accuracy collapses since stage 1, and most replies have no parsable answer. Gated partitioning therefore shows more consistent performance in our tested settings, while selective writes do not yet provide reliable results across models. Reducing information loss when older partitions are rebalanced without their source documents and understanding the Qwen failure are important directions for future work.

AI USAGE DISCLOSURE

We used AI assistants to help write, debug code and adapt open-source implementations for our experiments. We also used AI assistants to polish the paper writing. All technical decisions, experimental designs, and analysis were made by the authors.

REFERENCES

Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo JWL. Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. Longhealth: A question answering benchmark with long clinical documents, 2024. URL https://arxiv.org/abs/2401.14490.

Vittorio Castelli, Rishav Chakravarti, Saswati Dana, Anthony Ferritto, Radu Florian, Martin Franz, Dinesh Garg, Dinesh Khandelwal, Scott McCarley, Michael McCawley, Mohamed Nasr, Lin Pan, Cezar Pendus, John Pitrelli, Saurabh Pujar, Salim Roukos, Andrzej Sakrajda, Avi Sil, Rosario Uceda-Sosa, Todd Ward, and Rong Zhang. The TechQA dataset. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1269–1278, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.117. URL https://aclanthology.org/2020.acl-main.117/.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. Finqa: A dataset of numerical reasoning over financial data, 2022. URL https://arxiv.org/abs/2109.00122.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi (eds.), *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL https://aclanthology.org/W14-4012/.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers, 2021. URL https://arxiv.org/abs/2105.03011.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Y Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, volume 2026, pp. 42642–42687, 2026.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uREj4ZuGJE.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6342–6353, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.391. URL https://aclanthology.org/2023.emnlp-main.391/.

Jessy Lin, Luke Zettlemoyer, Gargi Ghosh, Wen-Tau Yih, Aram Markosyan, Vincent-Pierre Berges, and Barlas Oğuz. Continual learning via sparse memory finetuning. *arXiv preprint arXiv:2510.15103*, 2025.

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. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Yujian Liu, Jiabao Ji, Li An, Rohit Jain, Gungor Polatkan, Siyu Zhu, and Shiyu Chang. Practical online KV cache compaction for LLM agents: An empirical study, 2026. URL https://arxiv.org/abs/2608.00902.

David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f87522788a2be2d171666752f97ddebb-Paper.pdf.

Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=5MkYIYCbva.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=2DtxPCL3T5.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024. URL https://arxiv.org/abs/2404.07143.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized KV cache compaction in a single forward pass, 2026. URL https://arxiv.org/abs/2606.07878.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. Quality: Question answering with long input texts, yes!, 2022. URL https://arxiv.org/abs/2112.08608.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=t01SKTj8pP.

A ALGORITHMS

Algorithm 1 gives the write and read procedures of the gated update (Section 3.1), and Algorithm 2 the free-form sparse continual compaction (Section 3.2).

Algorithm 1: Gated update

Input: budget B (slots); threshold τ; model p_θ

State: partitions $\mathcal{P}=\{P_1,\dots,P_m\}$, each a compacted KV cache of $|P_j|$ slots from n_j tokens; $\sum_j |P_j| \le B$; initially $m=0$

Write (documents $\mathcal{D}_t$ of phase t)

- $g_j \leftarrow \ell(\mathcal{D}_t \mid P_j) - \ell(\mathcal{D}_t)$ **for** $j=1,\dots,m$ ▷ *gain in log-likelihood from P_j*
- **if** $m=0$ **or** $\max_j g_j \le \tau$ **then** ▷ *open a partition for $\mathcal{D}_t$*
  - $m \leftarrow m+1;\ w \leftarrow m;\ n_w \leftarrow |\mathcal{D}_t|;\ C \leftarrow \mathrm{KV}(\mathcal{D}_t);\ Q \leftarrow \textsc{SelfStudy}(C, \mathcal{D}_t)$
- **else** ▷ *merge $\mathcal{D}_t$ into P_w*
  - $w \leftarrow \arg\max_j g_j$
  - $Q_{\mathrm{mem}} \leftarrow \textsc{SelfStudy}(P_w)$ ▷ *recorded before $\mathcal{D}_t$ is prefilled*
  - $n_w \leftarrow n_w + |\mathcal{D}_t|;\ C \leftarrow P_w \cup \mathrm{KV}_{P_w}(\mathcal{D}_t);\ Q \leftarrow \textsc{SelfStudy}(C, \mathcal{D}_t) \cup Q_{\mathrm{mem}}$ ▷ *each source provides half of Q*
- $s_j \leftarrow \max\big(16,\ \mathrm{round}(B\, n_j / \textstyle\sum_i n_i)\big)$ ▷ $\sum_j s_j = B$
- $P_w \leftarrow \textsc{Compact}(C,\ Q,\ s_w)$
- **for** $j \neq w$ **with** $|P_j| > s_j$ **do** ▷ *partitions that lost slots*
  - $P_j \leftarrow \textsc{Compact}\big(P_j,\ \textsc{SelfStudy}(P_j),\ s_j\big)$

Read (question q)

- $r_j \leftarrow \sum_i \log p_\theta(q^{(i)} \mid P_j, q^{(<i)})$ **for** $j=1,\dots,m$ ▷ *over the tokens of q*
- **return** answer from p_θ with the partition of largest r_j as prefix

Algorithm 2: Continual KV Compaction

Input: memory budget B; continual write budget t; stages $\mathcal{D}_1, \ldots, \mathcal{D}_N$; model p_θ; weight λ_Δ
State: single compacted KV memory M of B slots; initially $M \leftarrow \emptyset$

1 **Write** (stage n)
2 **if** $n = 1$ **then** ▷ *initialize memory M*
3 $C \leftarrow \bigcup_{D \in \mathcal{D}_1} \mathrm{KV}(D)$; ▷ *prefill each doc separately and append their KVs*
4 $Q \leftarrow \textsc{SelfStudy}(\mathcal{D}_1)$
5 $M \leftarrow \textsc{Compact}(C, Q, B, \lambda_\Delta)$;
6 **else**
7 **for** $D \in \mathcal{D}_s$ **do** ▷ *continually update M*
8 $Q \leftarrow \textsc{SelfStudy}(M, D)$
9 $C \leftarrow M \cup \mathrm{KV}_M(D)$ ▷ *form the continual teacher*
10 $S \leftarrow \textsc{SelectSlots}(M, Q, t)$ ▷ *select top t slots*
11 $M \leftarrow \textsc{Compact}(C, Q, S, \lambda_\Delta)$ ▷ *compact but update to only slots S*
12 **Read** (question q)
13 **return** answer from p_θ with M as prefix

B Additional results for the gated update

B.1 Self-study queries

The four self-study prompts ask for a summary, a fact aggregation and a JSON rendering of the text, and for three questions that the model writes and then answers. A fifth conversation asks for the text verbatim, with the text itself as the answer. When only a cache is available, as in $\textsc{SelfStudy}(P_j)$, the model also generates the answer of the verbatim conversation. Conversations are sampled at temperature 0.7 with top-p 0.8 and top-k 20. Each call of $\textsc{SelfStudy}$ retains at most 50,000 query vectors per key-value head. At a merge, the queries of $\textsc{SelfStudy}(C, \mathcal{D}_t)$ and of $\textsc{SelfStudy}(P_w)$ share the 50,000 vectors of each key-value head equally. A set of fewer than 25,000 vectors contributes every vector in the set.

In the main text, gated update uses both sources at every merge, a configuration that we call gated update doc+mem. We also run a variant, gated update doc, that takes every merge query from $\mathcal{D}_t$ and sets $Q_{\mathrm{mem}} = \emptyset$ at line 7 of Algorithm 1. On the five-phase sequences, no merge occurs, so the two variants follow the same procedure and differ only in the conversations that self-study samples. Their final accuracies differ by at most 1.7% (Table 3), which indicates how much final accuracy varies with the sampled conversations alone.

B.2 Evaluation details

Adding the story title to each QuALITY question raises routing accuracy, the share of the questions of a phase that the router assigns to the partition holding the phase. After five phases at 1,024 slots with Qwen3-4B, routing accuracy per phase is 54.9% to 78.7% without the title, against 100% with the title.

Table 3: Without merges, gated update doc is within 1.7% of gated update doc+mem at both budgets. Final accuracy in % on the five-phase sequences.

		LongHealth		QuALITY	
Budget	Queries	Qwen3-4B	Llama-3.2-3B	Qwen3-4B	Llama-3.2-3B
1,024	doc+mem	28.0	22.3	32.5	32.3
	doc	29.3	21.5	33.0	32.7
16,384	doc+mem	33.5	32.0	42.3	36.2
	doc	33.5	33.0	40.5	37.8

We report final accuracy as the primary metric, since forgetting measures only the decrease from the highest accuracy on each phase. On LongHealth with Qwen3-4B at 16,384 slots, the forgetting of gated update and KV recursive is 18.1% and 19.7%, although their final accuracies are 33.5% and 0.3%. The in-context baseline reaches the highest accuracy, 60.8%, with a forgetting of 22.5%, above the 18.1% of gated update.

We compute 95% bootstrap intervals for the difference in final accuracy between two methods by resampling the questions of each phase with replacement 2,000 times. The two methods are matched by question id, and the intervals reflect only the sampling of questions.

B.3 Partitioning without continual updates

To test partitioning without continual updates, we compact each patient record of the first LongHealth phase once from the text into a separate cache of $B/4$ slots. With B from 1,024 to 16,384 slots, we ask the 80 questions about the four records. We compare oracle routing, which gives the model only the cache of the question's record, with reading all four caches together at consecutive or at overlapping positions. In addition, we compact all four records together into a shared cache of B slots. Placing the four records in the prompt gives the in-context accuracy.

With Qwen3-4B, oracle routing is more accurate than reading the four caches together at every budget and more accurate than the shared cache from 2,048 slots on. At 16,384 slots, oracle routing nearly reaches in-context accuracy (Figure 4; Appendix B.3). With Llama-3.2-3B, oracle routing is the most accurate read at 16,384 slots and exceeds in-context accuracy there.

Compaction fits each per-record cache only to reference queries about the same record. Reading several caches together therefore mixes unconstrained attention outputs into every answer. A shared cache avoids the mixing but fits one cache to queries about all four records. At 1,024 slots, with 256 slots per record, the shared cache is at least as accurate as oracle routing with both models. Gated update likewise reads separate partitions one at a time at the level of phases, where KV recursive holds a single cache. Within a phase, however, a partition of gated update holds all records of the phase, like the shared cache.

At 16,384 slots, oracle routing is the most accurate way to read the four caches with both models (Table 4). With Qwen3-4B, the 78.8% of oracle routing is close to the in-context accuracy of 80.0%. Reading the four caches together places them either at consecutive positions, concatenated as in KV concat, or at overlapping positions, where every cache starts at position 0. Both layouts are less accurate than oracle routing at every budget with Qwen3-4B. At 1,024 slots, oracle routing and the shared cache differ in accuracy by at most 2.5% with both models.

Table 4: Oracle routing is the most accurate read of the per-record caches at 16,384 slots with both models. Accuracy in % of Figure 4 by total budget B in slots.

		Total budget B				
Model	Read	1,024	2,048	4,096	8,192	16,384
Qwen3-4B	Oracle routing	41.3	55.0	56.3	62.5	78.8
	One shared cache	42.5	41.3	43.8	47.5	50.0
	Four caches, concatenated	27.5	25.0	30.0	40.0	58.8
	Four caches, overlapping positions	21.3	22.5	27.5	35.0	46.3
	Records in the prompt			80.0		
	No memory			27.5		
Llama-3.2-3B	Oracle routing	33.8	41.3	43.8	50.0	58.8
	One shared cache	36.3	37.5	40.0	33.8	41.3
	Four caches, concatenated	37.5	41.3	45.0	45.0	45.0
	Four caches, overlapping positions	40.0	36.3	40.0	41.3	47.5
	Records in the prompt			52.5		
	No memory			31.3		

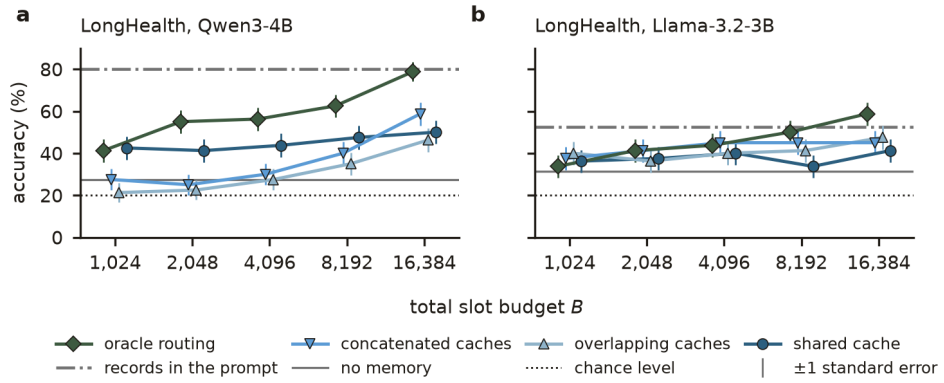


Figure 4: At 16,384 slots, oracle routing to one of four per-record caches comes close to or exceeds in-context accuracy. Accuracy on the 80 questions about the four patient records of the first LongHealth phase, without continual updates. Overlapping positions start every cache at position 0, and error bars show one standard error.

B.4 Five-phase sequences

On the five-phase sequences, the largest likelihood gain, 0.048 nats per token on QuALITY with Qwen3-4B at 1,024 slots in phase 5, is below $\tau = 0.05$. The gate therefore opens a new partition at every phase, and rebalancing reduces the size of all older partitions each time. On LongHealth with Qwen3-4B at 16,384 slots, four recompactions reduce the partition of phase 1 from 16,409 slots after phase 1 to 3,433 slots after phase 5.

After phase 5 at 16,384 slots, gated update is 6.5% $[1.0, 11.8]$ more accurate than the no-memory baseline on LongHealth and 5.1% $[1.0, 9.3]$ on QuALITY with Llama-3.2-3B (Table 8). With Qwen3-4B, the gains of 3.0% $[-2.5, 8.5]$ on LongHealth and 0.7% $[-3.9, 5.1]$ on QuALITY have intervals that include 0. Across phases, the gain is smallest on phase 1, on which gated update is less accurate than the no-memory baseline on both datasets with both models (Figure 6). The partition of phase 1 is recompacted four times, more often than any other partition (Section 4.3.3).

At 1,024 slots, the accuracy of gated update on the fifth phase of QuALITY with Qwen3-4B is 13.1% (Figure 7). Of the replies about the phase, 67% reach the 512-token limit without a parsable answer, almost all by repeating the name of the first story of the phase. The other replies give a parsable answer, and 40.0% of them are correct.

B.5 Routing

At 1,024 slots, routing accuracy is at least 80% on every phase (Figure 9). On the other datasets with Qwen3-4B at 1,024 slots, routing accuracy per phase is 97.9% to 100% on QASPER, 45.8% to 79.4% on FinQA and 3.4% to 47.4% on TechQA.

At 16,384 slots, routing accuracy on the oldest phase after five phases decreases to 0.0% on LongHealth and 12.1% on QuALITY with Qwen3-4B, and to 20.0% and 55.6% with Llama-3.2-3B. The router assigns 98.5% of the misrouted questions about phases 1 to 4 to a newer partition, against about 87% under random misrouting. Figure 8 shows the same trend with budget after two phases, in runs at budgets from 1,024 to 16,384 slots. On LongHealth with Qwen3-4B, routing accuracy on the older of the two phases decreases from 91.2% at 1,024 slots to 10.0% at 16,384 slots.

At 16,384 slots, the gain over the no-memory baseline depends on the routing outcome. Pooled over both datasets and models, gated update is 6.5% $[3.5, 9.5]$ more accurate than the no-memory baseline on correctly routed questions and 4.5% $[0.0, 9.0]$ less accurate on misrouted questions.

Replies stating that the memory holds no information on the named patient or story indicate why accuracy remains low at 1,024 slots. With Qwen3-4B at 1,024 slots, such replies make up 72% of the replies to correctly routed questions on LongHealth and 78% on QuALITY, against 5% and 1%

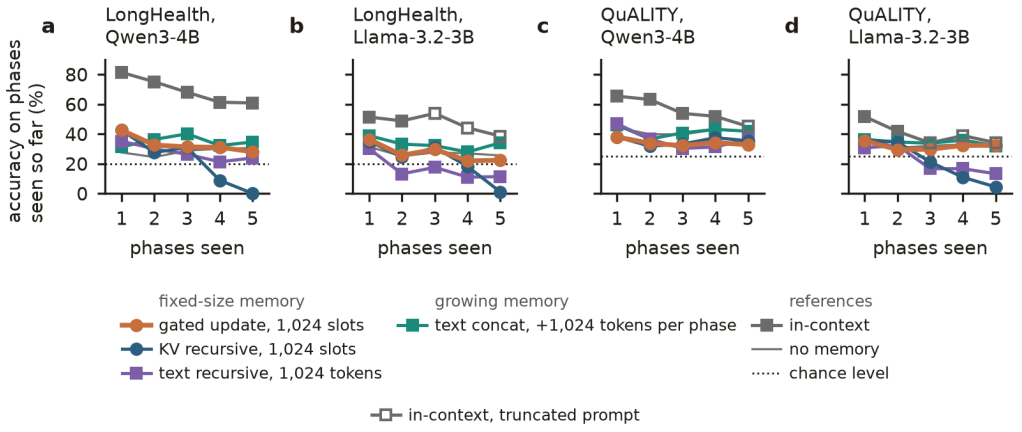


Figure 5: At 1,024 slots, gated update is about as accurate as the no-memory baseline, and KV recursive reaches less than 5% accuracy, except with Qwen3-4B on QuALITY. Layout as in Figure 1.

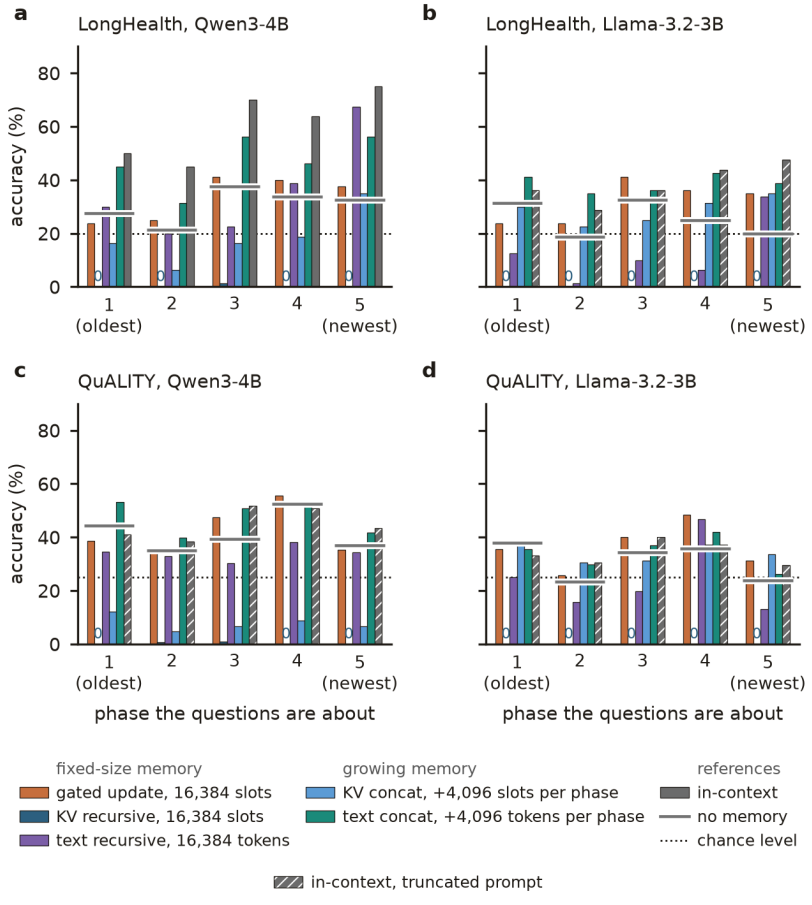


Figure 6: At 16,384 slots, gated update is less accurate than the no-memory baseline on phase 1, the phase with the most recompactions. Accuracy on each phase after phase 5, with grey segments for the no-memory baseline.

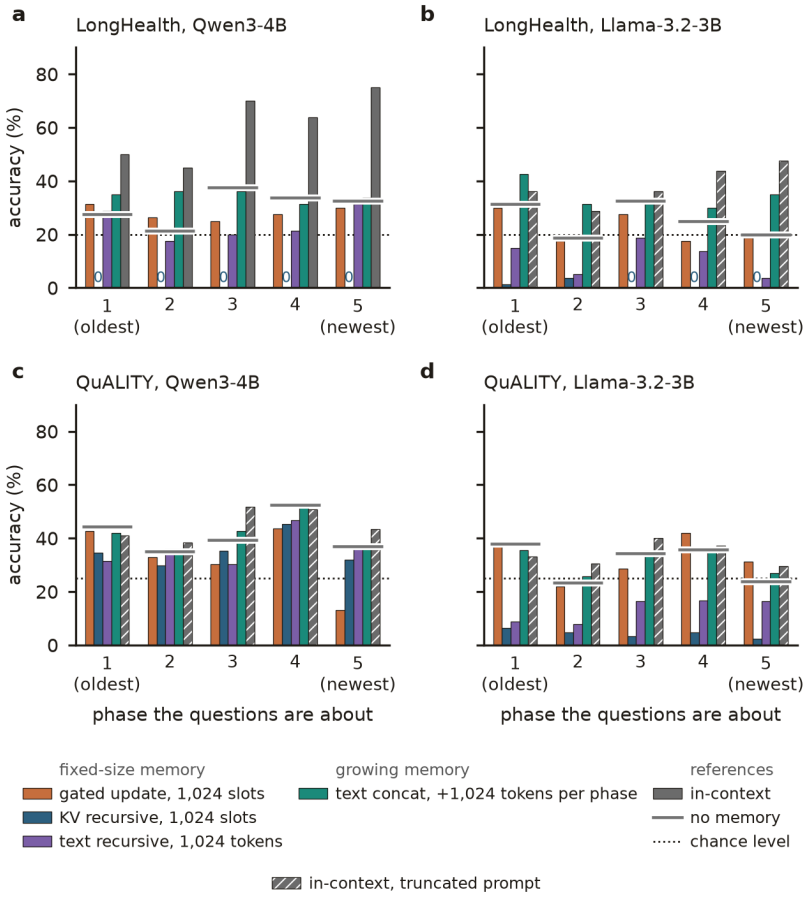


Figure 7: At 1,024 slots, the largest deficit against the no-memory baseline is on phase 5 of QuALITY with Qwen3-4B, where 67% of the replies give no parsable answer. Layout as in Figure 6.

without memory. At 16,384 slots, the share decreases to 25% and 21% for correctly routed questions and rises to 93% and 92% for misrouted ones. Almost all such replies to misrouted questions nevertheless end with an answer option. The chosen options are correct in 20.7% and 31.4% of the cases, close to the chance levels of 20% and 25%. With Llama-3.2-3B, such replies make up 3.2% to 44.2% of the replies with memory and 20.5% to 34.0% without memory, so they do not indicate what the memory holds.

B.6 Oracle routing and recompaction

Oracle routing, which uses the partition of the question's phase as prefix, changes final accuracy by at most 4.1% at both budgets (Table 5). We read each saved final memory with the likelihood router, with oracle routing and with all partitions at once, using the decoding parameters of the original runs. The only change with a 95% interval that excludes 0 is on QuALITY with Qwen3-4B at 16,384 slots, where oracle routing is 4.1% $[1.2, 6.8]$ more accurate than the router. On the questions that the router misroutes there, oracle routing is 12.6% $[4.3, 20.8]$ more accurate than reading the partition that the router selects.

At 1,024 slots, oracle routing changes final accuracy by at most 1.0%, as the router already assigns at least 80% of the questions of each phase to the correct partition. Under oracle routing, the final memory is within 2.6% of the no-memory baseline, except on QuALITY with Qwen3-4B, where the final memory is 9.1% $[4.9, 13.4]$ less accurate than the no-memory baseline.

With Qwen3-4B, reading all five partitions at once, concatenated as in KV concat, lowers accuracy on both datasets at both budgets. At 16,384 slots, the decrease is 33.5% $[29.0, 38.3]$ on LongHealth and 42.1% $[38.3, 45.9]$ on QuALITY, and at least 98% of the replies give no parsable answer. By comparison, KV concat, with the same layout over caches compacted once, reaches 18.5% accuracy on LongHealth at 16,384 slots. At 1,024 slots, accuracy decreases by 11.3% $[6.8, 15.5]$ on LongHealth and 16.9% $[12.8, 20.9]$ on QuALITY. With Llama-3.2-3B, reading all partitions at once lowers accuracy by 9.3% $[3.3, 14.5]$ on LongHealth at 16,384 slots. On QuALITY at both budgets and on LongHealth at 1,024 slots, the change with Llama-3.2-3B ranges from -3.0% to $+0.4\%$, and every interval includes 0.

The recompaction loss is largest on phase 1, which rebalancing recompacts four times, except with Llama-3.2-3B on LongHealth, where the loss on phase 2 is 2.5% larger (Figure 2a). For phase 1, the 95% interval of the loss lies above 0 on LongHealth with Qwen3-4B and on QuALITY with Llama-3.2-3B, and includes 0 otherwise. Right after phase 1, the accuracy on phase 1 is 1.6% to 22.5% higher than the accuracy of the no-memory baseline, and after phase 5, 2.4% to 26.3% lower. On LongHealth with Qwen3-4B, 95.0% of the replies about phase 1 give no parsable answer under oracle routing, against 1.3% under the likelihood router. Under the likelihood router, most questions about phase 1 are assigned to newer partitions. After phase 5, the partition of phase 1 holds 3,433 slots and the partition of phase 5 holds 3,451, so the two differ mainly in how often each was recompacted.

To test whether the final size explains the loss, we compact phase 1 once from the text to the final size of the oldest partition. The compaction is the first write of Algorithm 1 with a smaller budget. On phase 1, the once-compacted cache is more accurate than the oldest partition by 8.8% to 43.8% (Table 5). The 95% interval of the difference lies above 0 except with Llama-3.2-3B on LongHealth. Repeated recompaction without the text of the documents therefore causes the loss. Against the memory of 16,384 slots right after phase 1, the once-compacted cache differs on phase 1 by -5.6% to $+9.7\%$, and every 95% interval includes 0. The largest difference is on QuALITY with Qwen3-4B, at $+9.7\%$ $[0.0, 19.4]$.

B.7 Merging across documents

The merging sequence alternates LongHealth patient records 1 to 12 with 12 of the 35 QuALITY stories of the five-phase sequences, one document per phase, beginning with patient record 1. We use the first 8, the first 16 or all 24 documents. With Qwen3-4B, $B = 1{,}024$ and $\tau = 0.05$, the gate opens partitions at documents 1, 2 and 4 of the 16-document sequence and merges the remaining documents (Figure 11). At every merge, likelihood gains toward a partition of the same dataset range from 0.055 to 0.100 nats per token and rise with partition size. By comparison, likelihood gains toward a partition of the other dataset range from 0.006 to 0.064.

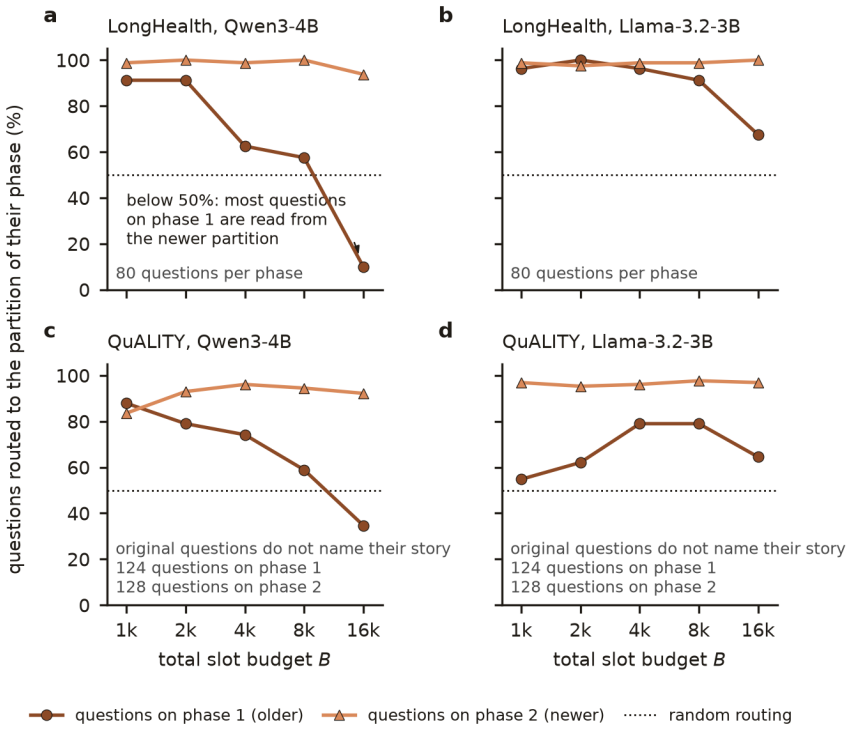


Figure 8: After two phases, fewer questions about the older phase are routed to the partition of the older phase at 16,384 than at 1,024 slots, except with Llama-3.2-3B on QuALITY. Routing accuracy against the budget B, with QuALITY questions that omit the story title.

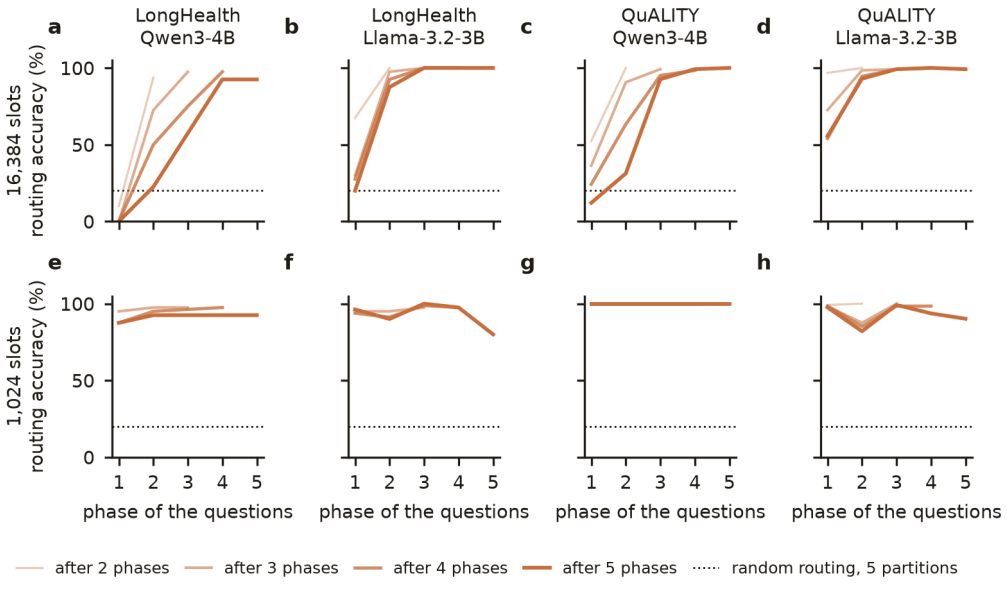


Figure 9: At 1,024 slots, at least 80% of the questions of each phase are routed to the partition of their phase, whereas at 16,384 slots routing accuracy on the oldest phases decreases as phases accumulate. Routing accuracy of gated update after each phase.

Table 5: At 16,384 slots, oracle routing changes final accuracy by at most 4.1%, and phase 1 compacted once to the final size of the partition of phase 1 is more accurate on phase 1 than the recompacted partition. Values in %, with 95% bootstrap intervals for the differences.

	LongHealth		QuALITY	
	Qwen3-4B	Llama-3.2-3B	Qwen3-4B	Llama-3.2-3B
Final accuracy				
Likelihood router	33.5	32.5	42.3	35.7
Oracle routing	31.8	31.0	46.4	34.7
Oracle minus router	-1.8 $[-5.3, +1.8]$	-1.5 $[-4.5, +1.3]$	$+4.1$ $[+1.2, +6.8]$	-1.0 $[-2.1, 0.0]$
Phase 1, oldest partition under oracle routing				
Accuracy	1.3	28.8	41.9	27.4
Unparsed replies	95.0	17.5	16.1	21.8
Phase 1 compacted once to the same size				
Size in slots	3,433	3,455	3,276	3,274
Accuracy	45.0	37.5	55.6	40.3
Unparsed replies	10.0	3.8	2.4	1.6
Compacted once minus oldest	$+43.8$ $[+32.5, +56.3]$	$+8.8$ $[-6.3, +23.8]$	$+13.7$ $[+4.0, +24.2]$	$+12.9$ $[+2.4, +23.4]$

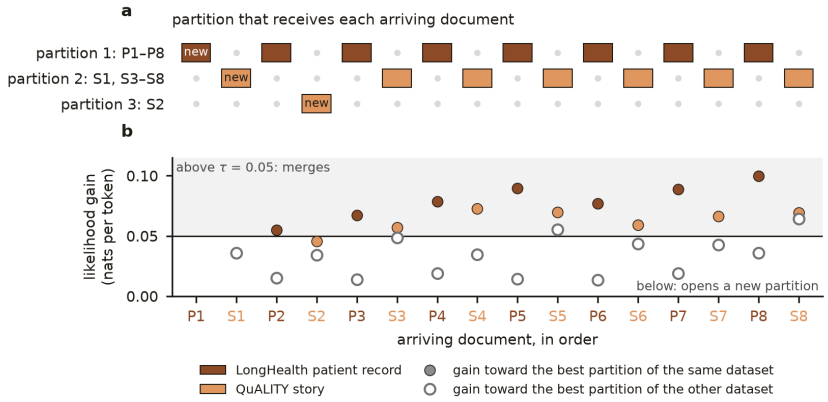


Figure 10: With Qwen3-4B, $B = 1{,}024$ and $\tau = 0.05$, the gate merges only documents of the same dataset, without dataset labels. (a) Partition that receives each arriving document, with Pi and Si for patient record i and story i. (b) Largest likelihood gain toward a partition of the same dataset, as filled markers, and of the other dataset, as hollow markers.

On the first eight documents with Qwen3-4B, we vary τ from 0.02 to 1.0 (Table 6). Accuracy is highest without merges, at 47.0% to 51.0% with eight partitions, against 37.7% to 44.4% with one to three partitions. After 16 documents, accuracy under $\tau = 0.05$ is 9.2% $[2.9, 15.0]$ lower than under $\tau = 1.0$, by 3.7% on the LongHealth questions and 15.6% on the QuALITY questions. The router then assigns 83.7% of the questions to the correct partition under $\tau = 0.05$, against 93.8% under $\tau = 1.0$.

With 24 documents, the gate forms 24 partitions under $\tau = 1.0$ and 5 under $\tau = 0.05$, and the router assigns 89.8% and 76.6% of the questions to the correct partition. Accuracy under $\tau = 1.0$ is 0.4% $[-4.1, +5.4]$ higher than under $\tau = 0.05$, and accuracy under both thresholds is higher than the accuracy of the no-memory baseline, by 6.7% $[1.9, 11.7]$ and 6.3% $[2.0, 10.6]$. Separate partitions hold fewer slots as documents accumulate, down to the 16-slot minimum of line 9, which a memory of 1,024 slots reaches at 64 documents.

With Llama-3.2-3B on the first eight documents, the gate merges no document under $\tau = 0.05$, as the largest likelihood gain is 0.049. The router assigns 94% to 100% of the questions about each document to the correct partition. Under $\tau = 1.0$, gated update is 2.0% more accurate than the no-memory baseline with Llama-3.2-3B, at 35.1%, and 15.2% more accurate with Qwen3-4B.

Table 6: The gate merging no document gives the highest accuracy. Accuracy in % after the first eight documents with Qwen3-4B at $B = 1{,}024$, by merge threshold τ.

τ	Partitions	Merges	Accuracy (%)
0.02	1	7	37.7
0.03	1	7	41.1
0.04	2	6	44.4
0.05	3	5	42.4
0.06	8	0	47.0
0.07	8	0	51.0
1.0	8	0	50.3

Gated update without merges is the most accurate memory of 1,024 slots on the eight-document sequence with both models (Table 7). KV recursive reaches 43.7% with Qwen3-4B but collapses with Llama-3.2-3B. With 128 slots per document, KV concat is less accurate than the no-memory baseline with Qwen3-4B, at 27.2% against 35.1%. Since the 75,010-token sequence fits the context window, the in-context baseline places every document in the prompt and reaches 70.2% with Qwen3-4B and 45.0% with Llama-3.2-3B.

Since the gate never merges on the five-phase sequences, we test merging on a sequence of single documents. The sequence alternates LongHealth patient records 1 to 12 with 12 QuALITY stories, one document per phase (Appendix B.7). Each new document thus has earlier documents of the same dataset to merge with. We expect merging to matter most when separate partitions are small and therefore use $B = 1{,}024$ slots, with $\tau = 0.05$ as in all our runs. With Qwen3-4B, we run the first 16 and all 24 documents, and with Llama-3.2-3B the first eight. After the last document of each run, we measure accuracy on the questions about all documents of the run.

With Qwen3-4B, every merge joins documents of the same dataset, although the gate receives no dataset labels (Figure 11). The patient records share one partition, and the stories two partitions. At every merge, the partition with the largest likelihood gain holds documents of the same dataset, so τ controls how many partitions the gate forms per dataset. With Llama-3.2-3B, no likelihood gain exceeds τ, and the gate opens a new partition for every document. A threshold that groups documents with one model thus need not group them with a different model.

To measure what merging costs, we also run the sequence with $\tau = 1.0$, a threshold that no likelihood gain reaches, so that every document has a separate partition. Rebalancing gives each document about the same share of slots under both thresholds, and the two runs therefore differ mainly in whether documents share a partition. After 16 documents, gated update with $\tau = 1.0$ is 9.2% more accurate than with $\tau = 0.05$. The router then assigns 93.8% of the questions to the partition of their document, against 83.7% with $\tau = 0.05$. After 24 documents, the difference decreases to 0.4%, and both runs are more accurate than the no-memory baseline (Appendix B.7).

With separate partitions of about 42 slots per document, gated update is 6.7% more accurate than the no-memory baseline on all 24 documents. The five-phase memories of 1,024 slots hold about 30 to 52 slots per document and are at most 2.6% more accurate even under oracle routing (Section 4.3.2). A partition thus appears to retain a single document better than several documents, an advantage that

Table 7: Gated update without merges is the most accurate memory of 1,024 slots on the first eight documents, and KV recursive collapses with Llama-3.2-3B. Acc and Unp as in Table 2, in %.

	Qwen3-4B		Llama-3.2-3B	
Method	Acc	Unp	Acc	Unp
In-context	70.2	6.6	45.0	1.3
Gated update (τ=1.0)	50.3	9.9	35.1	4.0
KV recursive	43.7	4.0	0.7	95.4
No memory	35.1	8.6	33.1	5.3
KV concat	27.2	7.9	33.8	2.6

merging gives up. Beyond 64 documents, however, separate partitions no longer fit in 1,024 slots at the 16-slot minimum of rebalancing, so a longer sequence requires merging.

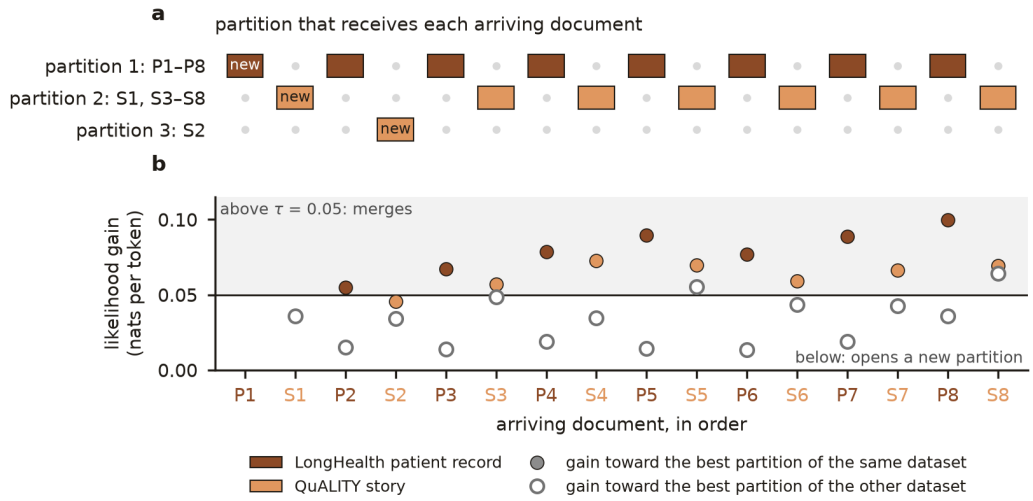


Figure 11: With Qwen3-4B, $B = 1{,}024$ and $\tau = 0.05$, the gate merges only documents of the same dataset, without dataset labels. (a) Partition that receives each arriving document, with Pi and Si for patient record i and story i. (b) Largest likelihood gain toward a partition of the same dataset, as filled markers, and of the other dataset, as hollow markers.

Comparison with text and in-context baselines. At equal size, gated update is more accurate than text recursive except on LongHealth with Qwen3-4B, whereas text memories that grow with each phase and the in-context prompt are about as accurate or more accurate over five phases (Table 2; Appendix B.8). The exception is QuALITY with Llama-3.2-3B, where gated update matches the in-context baseline with about an eighth of the memory. The advantage of a fixed size would therefore appear only on longer sequences, on which rebalancing also recompacts old partitions more often, so the recompaction loss could offset the saving in memory.

B.8 Comparison with the baselines

To judge whether a KV memory is competitive with memories of text, we compare gated update after phase 5 of the five-phase sequences with the text and in-context baselines (Table 2). A slot holds the keys and values of one token, as much memory as one prompt token, so we compare sizes in slots and in tokens directly. After phase 5, gated update holds 16,384 slots and text recursive at most 16,384 tokens. Text concat grows by 1,024 or 4,096 tokens per phase to 5,120 or 20,480 tokens. The in-context baseline places all documents in the prompt and truncates the prompt to the context window of the model.

At the same budget, gated update is more accurate than text recursive except on LongHealth with Qwen3-4B (Appendix B.8). Text concat with 1,024 tokens per phase stores less than gated update and is about as accurate, at most 2% more accurate on LongHealth and less accurate on QuALITY. For each answer, however, gated update gives the model less to read, one partition of about 3,300 slots against about 6,000 prompt tokens with text concat (Appendix B.9). Text concat with 4,096 tokens per phase and the in-context baseline store more than gated update and are more accurate, by up to 13.5% and 27.3% on LongHealth with Qwen3-4B. The exception is QuALITY with Llama-3.2-3B, where gated update is at least as accurate as every text baseline and the in-context baseline. Gated update thus matches the in-context baseline with about an eighth of the memory, as the in-context prompt there holds about 130,000 tokens.

Overall, gated update is more accurate than a text memory of equal size, whereas text memories that grow with each phase are about as accurate or more accurate over five phases. The advantage of a fixed size would therefore appear only on longer sequences, on which a growing memory of

similar accuracy outgrows gated update. On such sequences, however, rebalancing also recompacts the older partitions of gated update more often. Since the recompaction loss grows with the number of recompactions (Section 4.3.3), the accuracy of gated update on old phases could decrease further and offset the saving in memory.

Table 8 gives the difference in final accuracy between gated update and every baseline at both budgets, and Figure 12 plots accuracy against memory size for every method. At 16,384 slots, text concat with 4,096 tokens per phase is more accurate than gated update except with Llama-3.2-3B on QuALITY, by up to 13.5% $[8.0, 19.3]$ on LongHealth with Qwen3-4B. Gated update is more accurate than KV concat at 16,384 slots, by 2.3% to 15.0% where KV concat does not collapse.

At 16,384 slots, the difference between gated update and text concat with 1,024 tokens per phase ranges from -2.0% on LongHealth to $+4.2\%$ on QuALITY, both with Llama-3.2-3B. Text concat with 1,024 tokens per phase is more accurate than gated update at 1,024 slots, except with Llama-3.2-3B on QuALITY.

KV concat with Qwen3-4B on QuALITY and text recursive with Llama-3.2-3B lose accuracy mainly through replies without a parsable answer. With Llama-3.2-3B, 36% to 64% of the text recursive replies lack a parsable answer. Of such replies, 96% to 99% state that the memory holds no information on the named patient or story. On the replies with a parsable answer, text recursive reaches 31% to 38% accuracy, above the chance level. The gains of gated update over text recursive with Llama-3.2-3B, from 11.0% to 19.3%, therefore come from questions for which the text recursive replies give no parsable answer. After phase 5, the text recursive memory holds 4,368 and 3,624 of 16,384 tokens on LongHealth and QuALITY, so most of the budget remains unused. KV concat with Qwen3-4B on QuALITY produces repetition loops or strings of symbols in 88% of the replies and no parsable answer in 74%.

KV recursive does not collapse on QuALITY at 1,024 slots. With Qwen3-4B, KV recursive reaches 35.4% accuracy with 1% unparsed replies, 6.2% less than the no-memory baseline and 2.8% $[-1.1, 7.0]$ more than gated update. For Llama-3.2-3B, 87% of the KV recursive replies give no parsable answer, below the collapse threshold of 90%.

B.9 Memory cost

At 16,384 slots, the model decodes each answer with a single partition of about 3,300 slots as prefix. The partition is about 20% of the stored memory and about 16% of the 20,700 slots that the model attends to with KV concat. After phase 5, text concat places about 6,000 tokens in the prompt with 1,024 tokens per phase and 14,000 to 21,000 tokens with 4,096 tokens per phase. One slot holds the keys and values of one token at every layer and key-value head, which occupies 144 KiB with Qwen3-4B and 112 KiB with Llama-3.2-3B in bf16. At 1,024 slots, a partition holds about 210 slots, or 29.5 MiB with Qwen3-4B.

The stored memory of gated update exceeds B by at most 25 slots with Qwen3-4B and 46 slots with Llama-3.2-3B. Compacting a new phase preserves the chat-template prefix, which accounts for the excess, whereas each recompaction returns the affected partition to the exact target size. KV concat instead adds about 4,130 slots per phase, and the in-context prompt adds 44,000 to 55,000 tokens per phase.

The router scores each question under every partition, which requires one forward pass of the question per partition over the whole memory. Writing a phase requires scoring the new documents with every partition as prefix, compacting the documents and recompacting every partition that lost slots. As a result, the time to write a phase grows with the number of partitions.

On QuALITY, the Llama-3.2-3B tokenizer splits the five phases into 274,277 tokens, which gated update holds in 16,430 slots. The resulting memory is 2.1% $[-1.9, 6.2]$ more accurate than the in-context baseline, which truncates the prompt to 130,050 tokens.

B.10 Other datasets

On FinQA, accuracy is low for every method, with the in-context baseline at 9.5% with Qwen3-4B and 5.3% with Llama-3.2-3B. The KV memories, gated update included, reach 0.4% to 2.6% accuracy, close to the chance level. We estimate the chance level at 0.7% to 1.5% by scoring each reply against

Table 8: With Llama-3.2-3B, gated update is more accurate than KV recursive and text recursive at both budgets. Final accuracy of gated update minus the final accuracy of each baseline in %, with 95% bootstrap intervals, and † marks a collapsed baseline.

	LongHealth		QuALITY	
Baseline	Qwen3-4B	Llama-3.2-3B	Qwen3-4B	Llama-3.2-3B
Gated update at 16,384 slots				
No memory	+3.0 [−2.5, +8.5]	+6.5 [+1.0, +11.8]	+0.7 [−3.9, +5.1]	+5.1 [+1.0, +9.3]
KV recursive	+33.3† [+28.8, +38.0]	+32.0† [+27.3, +36.5]	+42.0† [+38.0, +45.7]	+36.2† [+32.4, +39.9]
KV concat	+15.0 [+10.3, +19.5]	+3.3 [−2.5, +8.8]	+34.6† [+30.4, +38.4]	+2.3 [−1.8, +6.3]
Text recursive	−2.3 [−7.5, +3.3]	+19.3 [+14.3, +24.0]	+8.2 [+3.8, +12.4]	+12.2 [+8.0, +16.5]
Text concat, 1,024	−1.0 [−6.8, +4.8]	−2.0 [−7.8, +3.3]	+0.5 [−4.2, +5.1]	+4.2 [0.0, +8.5]
Text concat, 4,096	−13.5 [−19.3, −8.0]	−6.8 [−12.3, −1.0]	−5.5 [−10.0, −1.1]	+2.1 [−2.3, +6.6]
In-context	−27.3 [−32.8, −21.3]	−6.5 [−12.3, −1.0]	−2.8 [−7.4, +1.5]	+2.1 [−1.9, +6.2]
Gated update at 1,024 slots				
No memory	−2.5 [−7.8, +2.5]	−3.3 [−8.0, +1.8]	−9.1 [−13.4, −4.9]	+1.3 [−2.1, +5.0]
KV recursive	+28.0† [+23.8, +32.3]	+21.3† [+17.0, +25.5]	−2.8 [−7.0, +1.1]	+28.0 [+24.0, +32.0]
KV concat	+9.5 [+4.5, +14.3]	−6.5 [−12.0, −1.0]	+24.8† [+20.8, +28.7]	−1.6 [−5.6, +2.6]
Text recursive	+4.3 [−0.8, +8.8]	+11.0 [+6.5, +15.8]	−3.8 [−8.0, +0.1]	+19.1 [+15.3, +23.0]
Text concat, 1,024	−6.5 [−12.0, −1.0]	−11.8 [−17.3, −6.3]	−9.3 [−13.8, −4.9]	+0.3 [−4.1, +4.5]
Text concat, 4,096	−19.0 [−24.8, −13.0]	−16.5 [−21.8, −11.0]	−15.2 [−19.9, −10.6]	−1.7 [−6.1, +2.6]
In-context	−32.8 [−38.3, −27.3]	−16.3 [−22.0, −10.8]	−12.5 [−17.0, −8.3]	−1.8 [−6.3, +2.3]

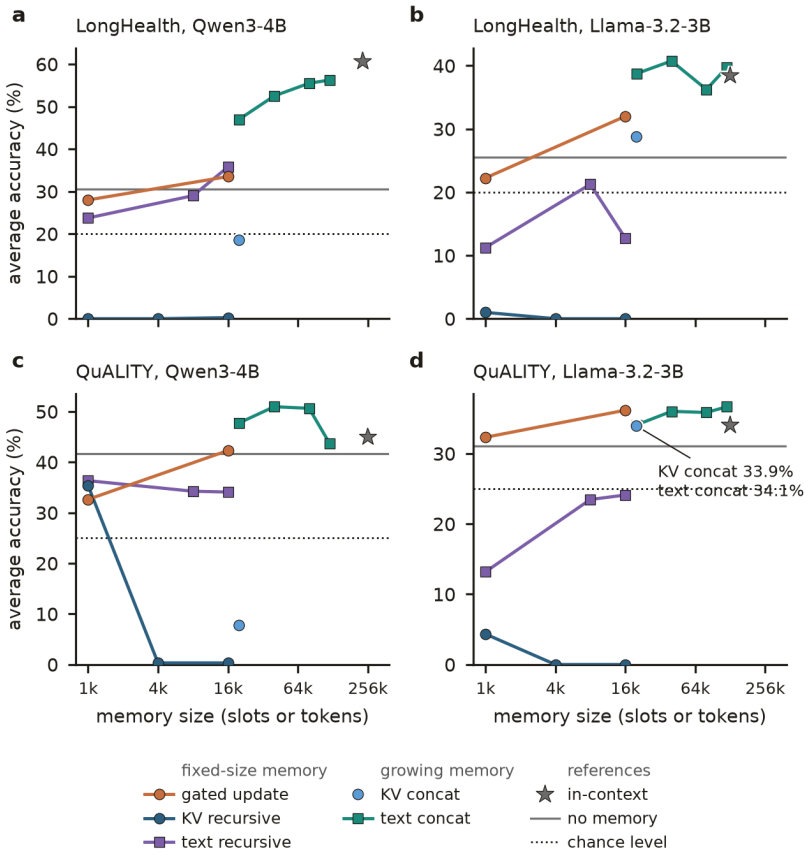


Figure 12: At 16,384 slots, gated update is the most accurate fixed-size memory except on LongHealth with Qwen3-4B. Final accuracy after phase 5 against memory size, in slots or tokens on a log scale. KV recursive points on QuALITY at 4,096 slots use questions that omit the story title.

the answer to another question of the same phase. As on LongHealth and QuALITY, the gate opens a new partition at every phase on FinQA, with both models and at both budgets.

On QASPER and TechQA, we measure the log-perplexity of the free-form reference answers, for which lower values are better. With Qwen3-4B at 1,024 slots, gated update lowers the log-perplexity on every phase, overall from 4.25 without memory to 3.51 on QASPER and from 2.92 to 2.38 on TechQA. On QASPER at 1,024 slots, gated update with Llama-3.2-3B reaches 3.02, close to the 3.04 without memory. At 16,384 slots, the log-perplexity of gated update on QASPER is 3.62 with Qwen3-4B and 2.83 with Llama-3.2-3B, against 4.25 and 3.04 without memory.

The text baselines reach lower log-perplexity than gated update with either model. Text concat at 8,192 tokens per phase reaches 3.33 on QASPER and 2.00 on TechQA with Qwen3-4B and 2.73 and 2.11 with Llama-3.2-3B. The in-context baseline reaches the lowest values, 2.82 and 1.10 with Qwen3-4B and 2.53 and 1.83 with Llama-3.2-3B. KV recursive at 1,024 slots with Qwen3-4B raises the log-perplexity to 9.91 on QASPER and 19.62 on TechQA, against 4.25 and 2.92 without memory.

C Additional Results for the Monolithic Update

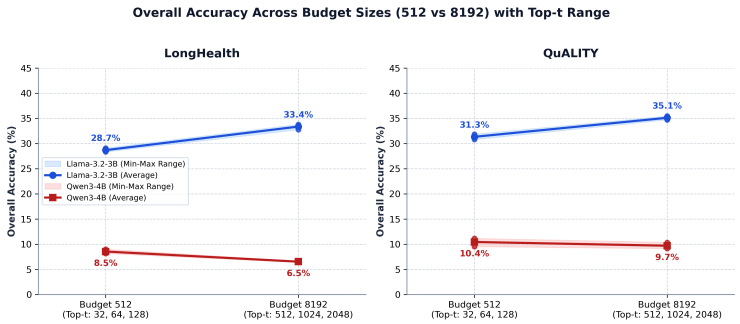


Figure 13: Overall performance report of budget size 512 and 8192. The dark color line represents the average performance across top-t values. The lighter color range gives the min and max average performance among the top-t runs.

Table 9: Final stage accuracy (%) after completing all continual learning phases on LongHealth and QuALITY, with 95% bootstrap confidence intervals.

	LongHealth		QuALITY	
Configuration	Qwen3-4B	Llama-3.2-3B	Qwen3-4B	Llama-3.2-3B
Budget 8,192 slots				
Top-t 512 (6%)	8.3 [5.8, 11.0]	36.3 [31.8, 41.0]	14.3 [11.6, 17.0]	39.4 [35.7, 43.4]
Top-t 1024 (12%)	8.3 [5.8, 11.0]	35.5 [30.8, 40.3]	12.9 [10.3, 15.6]	38.4 [34.6, 42.3]
Top-t 2048 (25%)	8.0 [5.5, 10.8]	37.0 [32.5, 41.8]	13.0 [10.5, 15.8]	37.8 [33.9, 41.6]
Budget 512 slots				
Top-t 32 (6%)	8.7 [6.0, 11.5]	30.8 [26.2, 35.5]	12.2 [9.6, 14.6]	33.9 [30.2, 37.6]
Top-t 64 (12%)	4.5 [2.5, 6.5]	29.8 [25.5, 34.2]	12.4 [9.8, 15.0]	33.4 [29.7, 37.0]
Top-t 128 (25%)	5.8 [3.5, 8.2]	31.2 [26.8, 35.8]	16.4 [13.7, 19.1]	36.0 [32.3, 39.7]

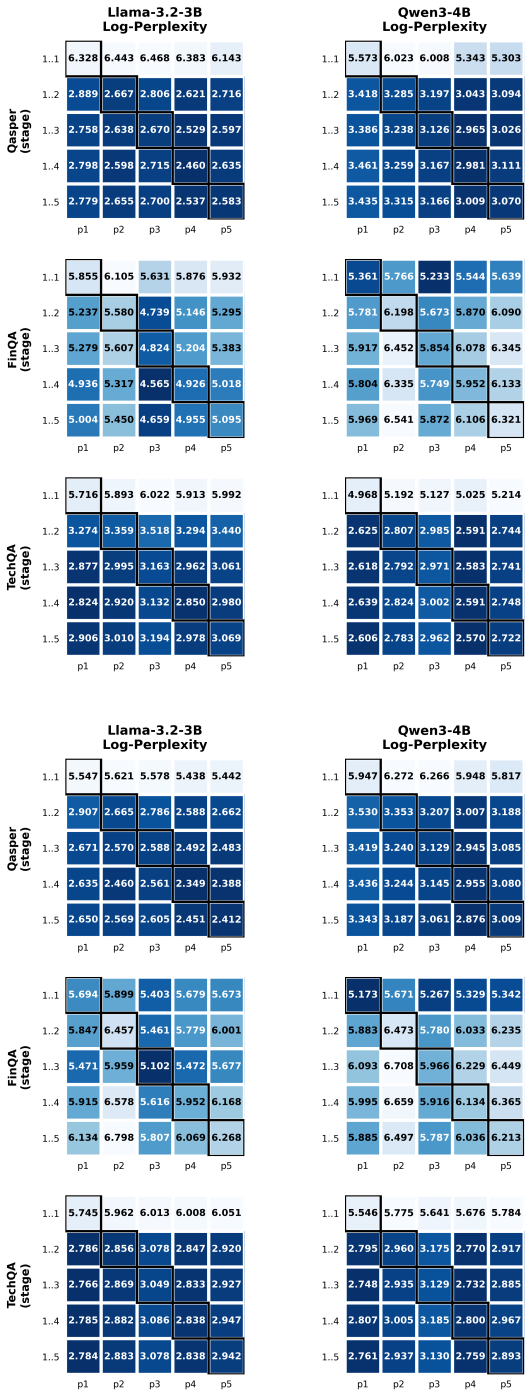


Figure 14: Performance of the monolithic memory update for Qasper, FinQA, and TechQA, recorded in Logppl against the ground truth (Top) Budget size 512 (Bottom) Budget size 8192.

D DATASETS

LongHealth (Adams et al., 2024) : consists of 20 fictional patient records with 5-way multiple-choice questions answered by option text. We assign four patients to each phase in ID order, so each phase is heterogeneous in diagnosis and phases are separated by patient identity.

FinQA (Chen et al., 2022) : pairs pages of SEC filings with numeric questions that require arithmetic over the table and ship an executable program, so answers are checkable without a judge. We fix the fiscal year (2017, the largest inventory: 71 companies, 241 pages, 737 questions) and vary the company across phases, keeping every page of a company in one phase; 13–15 companies make up a phase.

QuALITY (Pang et al., 2022) : provides long narratives of the 1950s-60s science fiction with 4-way questions written to require the whole plot rather than a lookup. We keep the 35 longest articles and place seven in each phase.

QASPER (Dasigi et al., 2021): pairs NLP papers with questions written by readers who saw only the tittle and abstract, answers are free-form and each grounded in evidence from the paper. We group papers by topic and assign one topic to each phase: machine translation, question answering, sentiment analysis, knowledge graphs, and speech recognition, with 16 longest papers of each topics. Phases are therefore separated by topic, and each phase is homogeneous in subject, the opposite of the LongHealth split.

TechQA (Castelli et al., 2020) :pairs IBM support technotes with real user questions from support forums; answers are free-form prose. We use the 610 answerable questions and their 496 source technotes, $\sim$99 per phase.