MEMTOKEN: SCALING QUERY-AGNOSTIC CONTEXT COMPRESSION ACROSS TASKS AND CONTEXT LENGTHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long contexts in real-world applications make the inference of Large Language Models (LLMs) expensive since attention prefill and key–value cache costs grow with sequence length, which motivates research on context compression. Soft token compression aims to encode a long context into a short sequence of continuous vectors that can be directly input into the LLM and replace the original context. Existing query-agnostic soft compressors are typically trained on limited data distributions and relatively short contexts, leaving generalization to unseen data and longer sources unresolved. To address these limitations, we introduce MemToken, a query-agnostic soft token compressor that condenses groups of context representations into memory tokens consumed by a frozen LLM. To improve distributional generalization, we construct a broad-coverage training set that combines web-text reconstruction with synthetic supervised samples across diverse downstream tasks. To scale source length, we use a short-context reconstruction warmup, interleave short and long contexts, and fine-tune on longer task contexts. Across four unseen long-context benchmarks with different LLMs, MemToken scales the compressed context length to 128K and significantly outperforms the state-of-the-art methods.

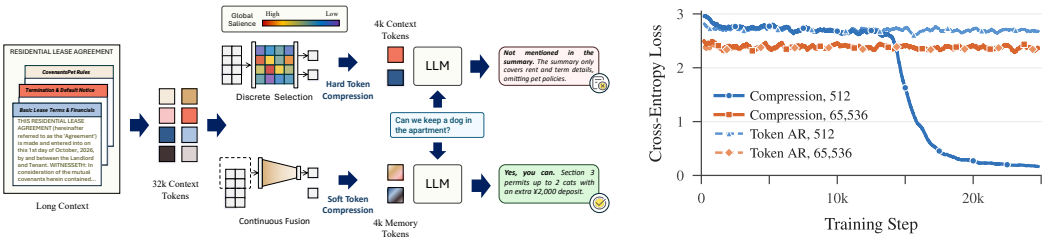
1 INTRODUCTION

Large Language Models (LLMs) are increasingly expected to reason over long documents (Bai et al., 2024; Li et al., 2023a), code repositories (Ding et al., 2023), and interaction histories (Maharana et al., 2024). Yet extending the context window does not make these inputs cheap to use: full-attention prefill grows quadratically with sequence length (Vaswani et al., 2017), while the key–value (KV) cache grows linearly and must be accessed during generation (Pope et al., 2023). The limited capacity and bandwidth of GPU and other AI-accelerator memory therefore constrain both the length of context that can be served and the cost of repeatedly consulting it. Human memory suggests a different approach: retain a compact representation of past information and recover what is relevant when needed, instead of rereading the entire history. For LLMs, this motivates compressing a long context into a much shorter representation that remains useful for subsequent tasks.

Existing context-compression methods can be broadly divided into *hard* and *soft* compression, as shown in Figure 1a. Hard methods shorten the input by selecting, deleting, or rewriting discrete text tokens (Li et al., 2023b; Jiang et al., 2023; Pan et al., 2024; Xu et al., 2024). They are interpretable and easy to attach to an existing model, but they may remove details needed by future queries. Soft methods instead learn continuous *compressed tokens* that encode the source context in the model's representation space (Mu et al., 2023; Chevalier et al., 2023; Ge et al., 2024; Cheng et al., 2024; Li et al., 2024b; Pilchen et al., 2025). This potentially allows information to be distributed across a compact set of vectors rather than retained only as verbatim tokens. We focus on query-agnostic soft compression, in which a context is compressed once before future queries are known. For such compressed tokens to be broadly useful, they must generalize across *unseen data distributions* and scale to *longer source contexts*.

The first challenge is out-of-distribution (OOD) generalization. Existing soft-compression models are typically trained and evaluated inside a single data pipeline: training and test samples come

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.



(a) (b)

Figure 1: (a) Hard compression summarizes or rewrites discrete text tokens, so partial information is removed before the future query is known and is unavailable to the LLM, while soft compression encodes the source into continuous compressed tokens that contain full information. (b) Cross-entropy loss during training with different context lengths: the compression loss is plotted against a plain token-autoregression reference that receives no compressed tokens, so the gap between them measures the contribution of the compressed tokens.

from one corpus and one question-construction procedure, so the reported scores mainly show that the compressed tokens support inputs similar to those seen during training (Mu et al., 2023; Ge et al., 2024; Li et al., 2024b). ICAE, for instance, is fine-tuned and tested on the training and test splits of one set of context–prompt–response samples generated from a single corpus (Ge et al., 2024). Held-out samples of this kind do not establish that the same compressed tokens remain effective under shifts in domain, document structure, or query type. The gap matters most for query-agnostic compression: the compressor cannot know which parts of an unseen document a later query will require, so any bias toward the training distribution is fixed in the compressed tokens before the query arrives.

The second challenge is scaling the *source context length* during training. Prior soft-compression training concentrates on inputs of hundreds to a few thousand tokens (Mu et al., 2023; Ge et al., 2024; Li et al., 2024b). Supporting a larger positional window or reporting long-context inference does not, by itself, show that the compressor has learned from comparably long training samples. Simply increasing the source length also changes the optimization problem. Under teacher-forced reconstruction, the model tends to ignore the compressed tokens and degenerate into plain autoregression over the output tokens, because the compressed tokens occupy what is in effect a new modality that carries no meaning to the decoder at initialization. Figure 1b shows the cross-entropy loss of Qwen3.5-2B trained under plain token autoregression (AR) and under compression, at two source lengths. At 512 tokens the compression loss falls well below the AR loss as training proceeds, which indicates that the model has learned to read the compressed tokens. At 65,536 tokens the two losses stay close for the entire run, showing that the model does not learn to use the compressed tokens effectively and instead relies on autoregressive prediction.

To address the above challenges, we introduce MemToken, a query-agnostic soft token compressor that can generalize to various downstream tasks and scale the compression context length to 128K. To address the OOD generalization challenge, we release the first training dataset for LLM context compression: we build a comprehensive training set assembled for coverage rather than from a single question-construction pipeline. The pretraining set consists of 2.43 million web-text samples with contexts spanning 128 to 131,069 tokens for a total of 80B tokens. The supervised fine-tuning (SFT) set contains 446,093 samples spanning various long-context tasks, such as document question answering, long-context summarization, long-horizon coding, and information retrieval, for a total of 7B tokens. To address the context-length scaling challenge, we use a short-context warmup followed by length expansion across training stages. Reconstruction pretraining begins on short contexts to align the compressed-token representations with the frozen LLM's input embeddings, then mixes short and long contexts. SFT extends training to longer source contexts and teaches the same compressed representation to support diverse queries. The two components are complementary: broad data expose the compressor to varied information, while the length-scaling recipe prepares it to use that information across longer sources. Across four long-context inference benchmarks, multiple LLM backbones, and compression ratios, MemToken significantly outperforms state-of-the-art query-agnostic compression methods and generalizes to OOD tasks and 128K-token contexts.

The contributions of this paper can be summarized as follows:

- We introduce MemToken, a query-agnostic soft context-compression method aimed at generalizing across both data distributions and source-context lengths.
- We construct the first training dataset for generalized context compression, consisting of a pretraining set with 80B tokens and a supervised fine-tuning set of 7B tokens, spanning various downstream tasks and context lengths.
- We propose a novel context compressor architecture and a two-stage training strategy that combines a short-context reconstruction warmup and mixed-length pretraining with supervised fine-tuning on longer contexts.
- In experiments, we provide a comprehensive comparison among alternative designs for context compression and show that MemToken significantly outperforms the state-of-the-art methods on 4 long-context inference benchmarks.

2 Related Work

Long-context inference. Long-context benchmarks ask a model to answer over long documents, code repositories, or multi-session dialogue (Bai et al., 2024; Li et al., 2023a; Shaham et al., 2023; Chen et al., 2026; Ding et al., 2023). Serving such inputs in full is costly because every query pays for the whole context again: prefill attention grows quadratically with its length and the key–value cache grows linearly (Vaswani et al., 2017; Pope et al., 2023). Cache-eviction methods reduce the memory of this cache after prefill (Zhang et al., 2023; Li et al., 2024a; Kim et al., 2025). Human readers take a different route: they do not reread a book to answer each question about it, but consult a condensed memory of what they have read. Context compression brings this economy to LLMs by encoding a context once into a much shorter representation that later queries reuse.

Hard token compression. Hard compression shortens the input in text space by selecting, deleting, or rewriting discrete tokens (Figure 1a). Selective Context (Li et al., 2023b) removes lexical units with low self-information, LLMLingua (Jiang et al., 2023) prunes tokens by a small model's perplexity under a budget controller, LLMLingua-2 (Pan et al., 2024) casts pruning as token classification distilled from GPT-4 labels, and RECOMP (Xu et al., 2024) rewrites retrieved passages into extractive or abstractive summaries. Query-aware variants (Jiang et al., 2024) also condition on the question. However, hard token compression methods inevitably discard partial information in the original context, losing semantics that may be utilized in future inference.

Soft token compression. For lossless semantic compression, soft token compression instead encodes the context into continuous vectors in the decoder's representation space. Gist tokens (Mu et al., 2023) compress a prompt into a few attended positions, AutoCompressors (Chevalier et al., 2023) carry summary vectors across segments, ICAE (Ge et al., 2024) and 500xCompressor (Li et al., 2024b) train a LoRA-adapted encoder whose memory slots a frozen decoder reads, xRAG (Cheng et al., 2024) projects a retrieval embedding into one token, and ARC-Encoder (Pilchen et al., 2025) adapts a separate encoder to one or several frozen decoders. C3 (Liu & Qiu, 2025) and DeepSeek-OCR (Wei et al., 2025) reach high ratios by reconstruction through a small language model and through rendered images. Most are trained within one corpus or question-generation pipeline and on sources of at most a few thousand tokens (Mu et al., 2023; Ge et al., 2024; Li et al., 2024b), the two axes along which MemToken broadens its training. PISCO (Louis et al., 2025) trains the reader together with the compressor. MemToken keeps the reader frozen as ICAE and ARC-Encoder do, and differs in that the producer is the decoder itself with a mid-stack gate, compression is one global pass rather than independent windows, and training runs reconstruction and then question answering. Holding training fixed, mean pooling has been reported to match or beat learned compression operators (Petrov et al., 2025; Feldman & Artzi, 2025). However, existing soft compression methods suffer from overfitting to specific tasks (e.g., merely context reconstruction for C3) and short compression lengths (typically hundreds or several thousand tokens). In this work, we propose MemToken, which generalizes to various downstream tasks and scales compression length to 128K.

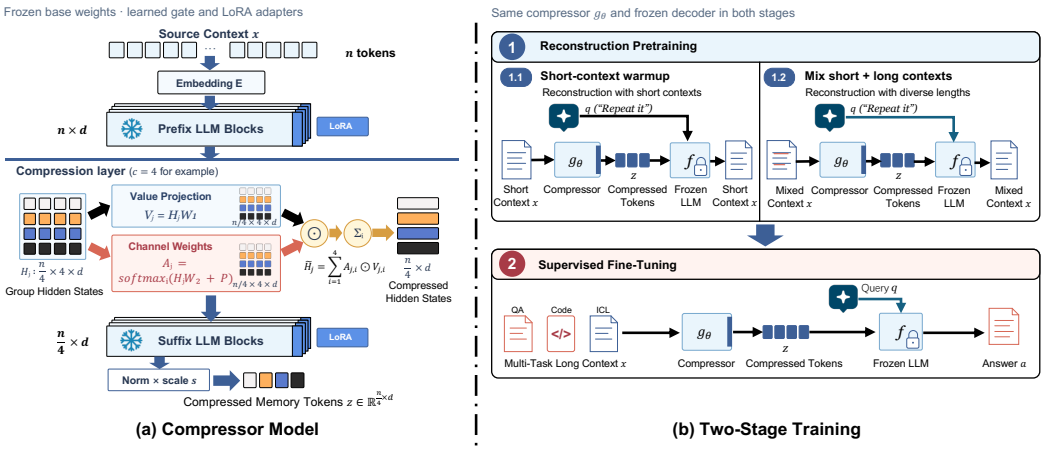


Figure 2: Overview of MemToken. (a) The compressor uses LoRA-adapted decoder blocks and a learned gate to turn each group of c source states into one compressed token ($c = 4$ shown). (b) Reconstruction begins with short contexts before mixing context lengths, followed by supervised fine-tuning with the same compressor and frozen decoder.

3 Task Definition

We define the task of *query-agnostic soft context compression* for a frozen LLM. Let f be a frozen LLM with token embedding function E and embedding dimension d, and let $x = (x_1, \ldots, x_n)$ be a context of n tokens. Given a compression ratio $c > 1$, a compressor g_θ produces

$$z = g_\theta(x) \in \mathbb{R}^{m \times d}, \qquad m = \lceil n/c \rceil. \tag{1}$$

The rows of z are *compressed tokens* that replace the context embeddings $E(x)$ as input to f, whose parameters and architecture remain unchanged. Compression is independent of the query, allowing z to be computed once and reused across queries.

The objective of the compression is to preserve the next-token distribution of f under context compression:

$$p_f\big(\cdot \mid [z;\, E(u)]\big) \approx p_f\big(\cdot \mid [E(x);\, E(u)]\big) \qquad \text{for every token sequence } u. \tag{2}$$

u denotes a query followed by any generated prefix, so the criterion applies at every decoding step.

4 Method

Figure 2 illustrates the framework of MemToken.

4.1 Compressor Model

We initialize the compressor from the frozen decoder f, insert a compression layer after the ℓ-th block, and train only its LoRA adapters, gate parameters, and output scale. For the context x and token embeddings $E(x)$ defined in Section 3, the first ℓ adapted decoder blocks produce hidden states $H = F^{\theta}_{1:\ell}(E(x)) \in \mathbb{R}^{n \times d}$. We partition H into $m = \lceil n/c \rceil$ consecutive groups of at most c states, padding only the final group if needed. For each group $H_j \in \mathbb{R}^{c \times d}$, the compression layer computes content vectors and channel-wise aggregation weights, and produces one state:

$$\begin{aligned} V_j &= H_j W_1 \in \mathbb{R}^{c \times d}, & A_j &= \mathrm{softmax}_i\big(H_j W_2 + P\big) \in \mathbb{R}^{c \times d}, \\ \widetilde{H}_j &= \sum_{i=1}^{c} A_{j,i} \odot V_{j,i} \in \mathbb{R}^{d}, & \widetilde{H} &= [\widetilde{H}_1; \ldots; \widetilde{H}_m] \in \mathbb{R}^{m \times d}. \end{aligned} \tag{3}$$

where $W_1, W_2 \in \mathbb{R}^{d \times d}$ are learned projections, $P \in \mathbb{R}^{c \times d}$ is a learned positional bias within each group, and $\odot$ denotes element-wise multiplication. The softmax is taken over the group index i

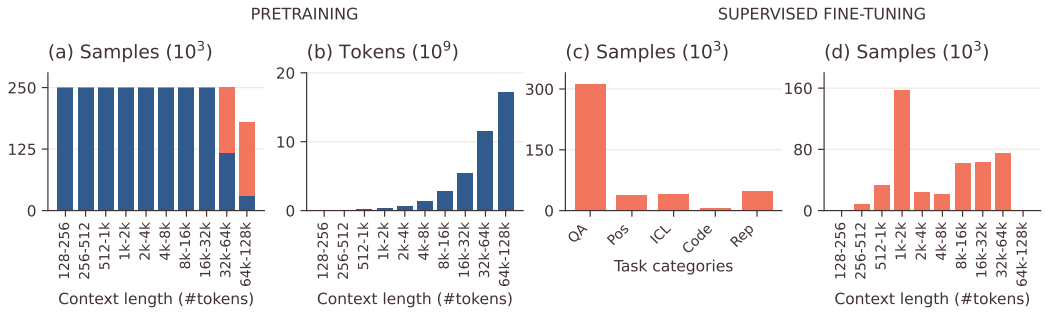


Figure 3: Distributions of our constructed pretraining and supervised fine-tuning (SFT) sets. (a) Pretraining samples by context length, separating original Ultra-FineWeb records (blue) from concatenated contexts (coral). (b) Source tokens by context length. (c) SFT samples by task category: question answering (QA), positional lookup (Pos), in-context induction (ICL), code understanding (Code), and reconstruction replay (Rep). (d) SFT samples by context length.

independently for each channel, so each output channel can assign different weights to the c input states. The compressor outputs $z = g_\theta(x) = s\,\mathrm{Norm}(F^\theta_{\ell+1:L}(\widetilde{H}))$, where the trainable scalar s is initialized to match the expected magnitude of f's input embeddings. This global scale calibration reduces the mismatch with the LLM's input-embedding distribution to accelerate training, while s remains learnable. The rows of z are compressed memory tokens supplied to the frozen LLM.

4.2 Training Dataset

Reconstructing the compressed context is a common task in training the token compressor. However, compressors trained only on reconstruction cannot transfer to general tasks (Liu & Qiu, 2025; Wei et al., 2025). Therefore, we construct two complementary datasets for context compression: a web-text pretraining corpus of 80B tokens and a synthetic supervised fine-tuning set covering various downstream tasks. The former is for learning lossless information compression and the latter for task generalization, and Figure 3 shows their distributions.

4.2.1 Pretraining Set

We build the pretraining set from Ultra-FineWeb (Wang et al., 2025), a web corpus of about one trillion tokens obtained by filtering open web crawls with a lightweight fastText quality classifier. Web pages are mostly short, whereas the compressor must retain contexts of every length. Therefore, we stratify records by token count into ten doubling bins, where bin $k \in \{1, \ldots, 10\}$ holds samples of 2^{k+6} to $2^{k+7} - 1$ tokens, from 128–255 in the first bin to 65,536–131,071 in the last, and cap each bin at 250,000 samples. Records outside this range are discarded rather than truncated, so every context is a complete document. Original pages longer than 32,768 tokens are scarce, so we complete the two longest bins with samples synthesized by joining randomly drawn shorter records with newlines until a length sampled within the target bin is reached, using each source record at most once and never using it as a stand-alone context. Because Ultra-FineWeb is already quality-filtered, our cleaning is limited to deduplication by record identifier and to removing literal `<think>` and `</think>` markers, which the LLM chat template treats as reasoning delimiters and would use to truncate the reconstruction target. Each training sample pairs a context with one of 29 paraphrased repetition instructions to alleviate overfitting and uses the context itself as the target.

The final set contains 2,430,000 distinct samples and 39.83 billion tokens, with 250,000 samples in each of the first nine bins and 180,000 in the longest. Nearly all of the 282,029 concatenated samples fall in the two longest bins, where they make up 53.3% and 82.6% of the samples. A validation set of 100,000 samples, 10,000 per bin, is drawn from the same sampling pool and shares no Ultra-FineWeb record with the training set, either stand-alone or inside a concatenation. Since the target repeats the input context, the rendered samples total about 79.7 billion tokens.

4.2.2 Supervised Fine-Tuning (SFT) Set

The SFT set combines numbered Wikipedia passages from the HotpotQA training split (Yang et al., 2018), whole Python files from `codeparrot/codeparrot-clean`, synthetic demonstration

blocks, and reconstruction replay. For document question answering, each context contains ten titled passages, with evidence positions assigned round-robin. GLM-5.2 generates single-passage and two-hop questions, paraphrases original questions, and supplies supporting quotes. Mechanical checks reject unsupported quotes and answer leakage, after which GLM-5.2 validates the remaining questions. It also rejects two-hop questions answerable from a single supporting quote. For Wikipedia questions, the frozen Qwen3.5-2B answers from the full context, and its wording replaces the target only when its token-F1 against the original answer is at least 0.6. The resulting targets are shared across backbones. Whole-sentence questions use the same evidence but require a verbatim sentence rather than a short answer.

Positional lookup uses passage headers as labels, in-context induction uses fresh mappings in demonstration blocks, and code answers come from the files' abstract syntax trees. GLM-5.2 phrases the code questions and screens them for answer leakage and dependence on the file. Replay samples ask the LLM to reproduce web-text contexts from the pretraining source. Before augmentation, we remove benchmark evidence paragraphs and overlapping code and screen generated questions against evaluation questions. We extend prose contexts with shuffled distractor passages to evidence fractions of 10%, 5%, or 2%.

The final set contains 446,093 samples across 19 task types: question answering accounts for 69.9%, positional lookup for 8.8%, in-context induction for 9.0%, code understanding for 1.3%, and reconstruction replay for 11.0% (Table 4). The set totals 6.94 billion tokens in training.

4.3 TWO-STAGE TRAINING

We propose a two-stage training strategy for the LLM context compressor, consisting of reconstruction pretraining and supervised fine-tuning.

Training objective. Both stages optimize the compressor parameters θ while keeping the LLM f frozen. For a context x, a user query q, and a target response $y = (y_1, \ldots, y_T)$, we first compute $z = g_\theta(x) \in \mathbb{R}^{m \times d}$ without access to q. Under teacher forcing, the LLM predicts each target token from z, the query q, and the preceding gold response tokens. Writing its vocabulary logits at target position t as $u_t \in \mathbb{R}^{|\mathcal{V}|}$, where $\mathcal{V}$ is the token vocabulary, the per-sample loss is

$$\ell_{ce}(x, q, y) = -\frac{1}{T} \sum_{t=1}^{T} \log \left[\mathrm{softmax}(u_t)\right]_{y_t}, \qquad u_t = f([z; E(q); E(y_{<t})]). \tag{4}$$

Here f returns the next-token logits, and the query embeddings include the chat-template headers.

Stage 1: reconstruction pretraining. For each web-text context x, we sample a reconstruction instruction q_{rep} and set the response text to x itself. Stage 1 therefore minimizes

$$\mathcal{L}_{\mathrm{pre}}(\theta) = \mathbb{E}_{x, q_{\mathrm{rep}}} \left[\ell_{ce}(x, q_{\mathrm{rep}}, x)\right], \tag{5}$$

with response delimiters included as specified above. The LLM receives the source context through z and reconstructs it autoregressively, learning lossless semantic compression of long contexts.

Context length warmup. As Figure 1b shows, direct training on long contexts leaves the optimization close to a plain autoregressive baseline, while the compressed tokens remain unlearned and are ignored by the LLM. To address this, we propose a simple yet effective strategy to warm up the context length. We begin pretraining on short contexts to establish semantic-lossless compressed tokens adapted to the frozen LLM. After the compressor learns to align the compressed-token space with the LLM input-embedding space (indicated by a cliff in the training loss), we interleave short and long contexts from the eligible length bins in roughly equal proportions, without replacement.

Stage 2: supervised fine-tuning. Pretraining enables the compressor to condense semantics into compact tokens. However, the trained compressor is overfitted to the reconstruction task and cannot be applied to other tasks yet. Therefore, we apply an SFT stage to build general instruction-following ability, with the question q and answer a from the SFT set:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = \mathbb{E}_{(x,q,a) \sim \mathcal{D}_{\mathrm{SFT}}} \left[\ell_{ce}(x, q, a)\right], \tag{6}$$

where $\mathcal{D}_{\mathrm{SFT}}$ is the set described in Section 4.2, including its reconstruction replay samples. This stage teaches the compressed representation to support diverse queries and extends the context-length coverage beyond reconstruction pretraining.

Table 1: Evaluation results for query-agnostic context compression on 4 benchmarks: score and, in parentheses, normalised R. Bold marks the best method per column and ratio. †Ratio on KV-cache bytes. ‡Trained baseline.

	Qwen3.5-9B				Qwen3.5-2B			
Method	LongBench	LooGLE	CCEval	FRAMES	LongBench	LooGLE	CCEval	FRAMES
No context	0.204 (0.00)	0.172 (0.00)	0.035 (0.00)	0.111 (0.00)	0.130 (0.00)	0.114 (0.00)	0.005 (0.00)	0.047 (0.00)
Full text	0.609 (1.00)	0.706 (1.00)	0.242 (1.00)	0.509 (1.00)	0.455 (1.00)	0.590 (1.00)	0.044 (1.00)	0.352 (1.00)
4× compression								
First+last	0.372 (0.41)	0.256 (0.16)	0.103 (0.33)	0.338 (0.57)	0.275 (0.44)	0.246 (0.28)	0.015 (0.26)	0.244 (0.65)
Selective-Context	0.283 (0.19)	0.203 (0.06)	0.060 (0.12)	0.264 (0.39)	0.190 (0.18)	0.136 (0.05)	0.004 (−0.02)	0.148 (0.33)
LLMLingua-2	0.398 (0.48)	0.352 (0.34)	0.051 (0.07)	0.316 (0.52)	0.281 (0.46)	0.238 (0.26)	0.004 (−0.01)	0.188 (0.46)
KVzip†	0.439 (0.58)	0.476 (0.57)	**0.221** (0.89)	0.280 (0.43)	0.201 (0.22)	0.182 (0.14)	0.020 (0.40)	0.094 (0.15)
ARC-Encoder‡	0.395 (0.47)	0.391 (0.41)	0.108 (0.35)	0.300 (0.48)	0.271 (0.43)	0.201 (0.18)	0.005 (0.01)	0.201 (0.51)
MemToken (ours)	**0.526** (0.79)	**0.582** (0.77)	0.191 (0.75)	**0.399** (0.72)	**0.406** (0.85)	**0.438** (0.68)	**0.040** (0.89)	**0.309** (0.86)
8× compression								
First+last	0.284 (0.20)	0.165 (−0.01)	0.069 (0.17)	0.280 (0.43)	0.214 (0.26)	0.178 (0.14)	0.009 (0.12)	0.200 (0.50)
Selective-Context	0.220 (0.04)	0.118 (−0.10)	0.042 (0.03)	0.221 (0.28)	0.159 (0.09)	0.106 (−0.02)	0.003 (−0.04)	0.115 (0.22)
LLMLingua-2	0.285 (0.20)	0.186 (0.03)	0.037 (0.01)	0.253 (0.36)	0.202 (0.22)	0.154 (0.09)	0.003 (−0.03)	0.140 (0.30)
KVzip†	0.245 (0.10)	0.152 (−0.04)	0.114 (0.38)	0.136 (0.06)	0.148 (0.06)	0.118 (0.01)	0.008 (0.09)	0.058 (0.04)
ARC-Encoder‡	0.361 (0.39)	0.340 (0.31)	0.066 (0.15)	0.272 (0.40)	0.242 (0.34)	0.139 (0.05)	0.010 (0.15)	0.163 (0.38)
MemToken (ours)	**0.453** (0.62)	**0.482** (0.58)	**0.166** (0.63)	**0.367** (0.64)	**0.357** (0.70)	**0.278** (0.34)	**0.031** (0.66)	**0.230** (0.60)

5 EXPERIMENTS

This section evaluates MemToken against query-agnostic compression methods on four long-context benchmarks, with further details and results in Appendices A and B.

5.1 EXPERIMENTAL SETUP

Backbones and baselines. We apply MemToken to three frozen LLMs, Qwen3.5-2B, Qwen3.5-9B and Nemotron-3-Nano-4B (Qwen Team, 2026; NVIDIA, 2026), and train one compressor per LLM and ratio ($4\times$ and $8\times$). We compare with five query-agnostic methods at the same measured token budget: first+last truncation, Selective-Context (Li et al., 2023b), LLMLingua-2 (Pan et al., 2024), KVzip (Kim et al., 2025) and the trained ARC-Encoder (Pilchen et al., 2025) (Appendix A.1). Answering without the context (no context) and with the uncompressed context (full text) gives the floor and the ceiling of each benchmark.

Benchmarks. We evaluate on four long-context benchmarks across various tasks, i.e., LongBench (Bai et al., 2024), LooGLE (Li et al., 2023a), CrossCodeEval (CCEval) (Ding et al., 2023), and FRAMES (Krishna et al., 2024), and no data from these benchmarks is used to train the compressor.

Metrics. Each benchmark is scored with its official code and metric, averaged over tasks with equal weight. Token-overlap metrics under-credit correct answers worded differently from the reference, so Table 1 reports LooGLE, FRAMES and the nine question-answering tasks of LongBench as the accuracy of an LLM judge, DeepSeek-V4-Flash (DeepSeek-AI, 2026) (validation in Appendix A.3, official metrics in Appendix B.1). Because the LLMs differ in how well they read the full context, we also report the normalised score

$$R = \frac{S_{\text{method}} - S_{\text{no-context}}}{S_{\text{full-text}} - S_{\text{no-context}}}, \tag{7}$$

computed per LLM on the same items, so that $R = 0$ means the compressed context is worth nothing and $R = 1$ means that it is worth the full context.

5.2 RESULTS AND DISCUSSION

MemToken has the best score among compression methods in 15 of the 16 benchmark–LLM–ratio settings of Table 1, including all eight at $8\times$. On LongBench and LooGLE every baseline is significantly below it with both LLMs at both ratios (40 exact McNemar tests, all $p \leq 3.5 \times 10^{-12}$). The exception is CrossCodeEval with Qwen3.5-9B at $4\times$, where KVzip keeps the verbatim local code that line completion relies on (R 0.89 against 0.75). On FRAMES with Qwen3.5-2B at $8\times$,

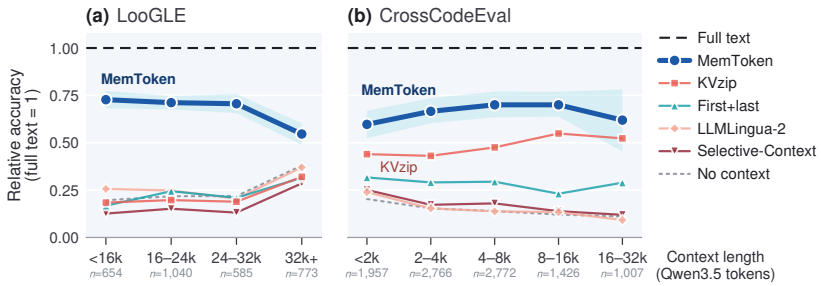


Figure 4: Accuracy relative to full text by context length with Qwen3.5-9B at $8\times$: (a) LooGLE, judge accuracy, and (b) CrossCodeEval, exact match. Shading is a 95% bootstrap interval for MemToken, and n counts questions per bin.

MemToken stays ahead of first+last (0.230 against 0.200), although the difference is not significant ($p = 0.09$). On CrossCodeEval with Qwen3.5-2B, full text improves on no context by only 0.039, so small absolute differences move R a lot.

Doubling the ratio costs the untrained baselines most of their gain on Qwen3.5-9B (KVzip falls from R 0.58 to 0.10 on LongBench and below the floor on LooGLE), whereas MemToken keeps about three quarters of it (0.79 to 0.62 and 0.77 to 0.58). MemToken at $8\times$ therefore matches or exceeds every baseline at $4\times$ on LongBench and LooGLE with both LLMs, at half their token budget.

5.3 Performance Across Context Lengths

Figure 4 splits the $8\times$ results of Qwen3.5-9B on LooGLE and CrossCodeEval by context length. Up to 32K tokens MemToken keeps a flat share of full-text accuracy, 0.71–0.73 on LooGLE and 0.60–0.70 on CrossCodeEval, while no LooGLE baseline rises more than 0.06 above the no-context line and KVzip, the strongest CrossCodeEval baseline, reaches 0.43–0.55. The 16K–32K CrossCodeEval bin holds only 8 repositories, so the gap to KVzip there is not resolved. Beyond 32K tokens the LooGLE share falls to 0.55, and because full text itself drops in that bin (from 74% to 52% accuracy), the loss is larger than the ratio shows.

5.4 Ablation Study

Both training stages are needed (Table 2): Stage 2 alone stays at the no-context floor (R 0.01–0.12), and Stage 1 alone falls below it ($R = -0.47$ on LongBench) because the LLM never learns to stop (median 128 generated tokens against 7). Adding a KL or hidden-state distillation loss from the full-text LLM does not improve Stage-1 reconstruction (Appendix B.4).

Replacing the trained selector at inference lowers R step by step as more of the learned selection is removed, from no effect when the positional prior P is dropped to 0.68 against 0.88 when each window keeps only its first token. About half of this loss falls on the two passage-retrieval tasks (Appendix B.5), so what the selector adds is mainly positional fidelity. Yet a compressor retrained with the selector fixed to the window mean matches MemToken on LongBench (no significant difference on either axis) and trails it only on LooGLE (judge R 0.61 against 0.68), so the training recipe rather than the operator carries most of the result.

On Nemotron-3-Nano-4B, a hybrid Mamba LLM, MemToken at $4\times$ recovers 0.62 of the full-text gain on LongBench and 0.63 on FRAMES (Table 2, bottom, and Appendix B.7).

5.5 Efficiency Analysis

MemToken is used as a memory: a context is compressed once and every later query is answered from the stored vectors. Figure 5 therefore separates the cost paid at every query from the one-off cost of building the memory, on Qwen3.5-9B with one A800 GPU and 64 MuSiQue questions padded to each length. At answer time the decoder reads a quarter or an eighth of the positions, so at 32K tokens it spends 23% of the full-text FLOPs at $4\times$ and 11% at $8\times$. From 16K to 64K it therefore answers 3.7–4.4 times as many queries per second as full text at $4\times$ and 6.7–8.1 times at

Table 2: Ablations on Qwen3.5-2B at 4×: score and, in parentheses, R. Override rows replace the trained selector at inference, and *mean pool* is retrained with the selector fixed to the window mean. Each row is tested against MemToken on the same items (paired t on the continuous axes, exact McNemar on the judge axes): * $p < 0.00125$ (Bonferroni over 40 tests), † $p < 0.05$. The last block is MemToken on Nemotron-3-Nano-4B, normalised by that LLM's own floor and ceiling, untested.

Variant	LongBench (official, 29 lanes)	LongBench-QA (judge)	LooGLE (token-F1)	LooGLE (judge)
MemToken (ours)	0.367 (0.88)	0.406 (0.85)	0.275 (0.86)	0.438 (0.68)
Selector replaced at inference on the trained checkpoint				
P removed (Z only)	0.366 (0.88)	0.406 (0.85)	0.275 (0.86)	0.435 (0.67)
P only	0.355 (0.82)*	0.382 (0.77)*	0.269 (0.82)†	0.421 (0.65)†
Uniform (window mean)	0.354 (0.82)*	0.378 (0.76)*	0.269 (0.82)†	0.421 (0.64)†
Shuffled	0.348 (0.79)*	0.377 (0.76)*	0.261 (0.77)*	0.409 (0.62)*
First token of window	0.323 (0.68)*	0.345 (0.66)*	0.224 (0.51)*	0.279 (0.35)*
Selector fixed to a mean from step 0 and retrained				
Mean pool	0.363 (0.86)	0.410 (0.86)	0.262 (0.77)*	0.405 (0.61)*
Training stages				
Stage 1 only	0.072 (−0.47)*	0.008 (−0.38)*	0.043 (−0.74)*	0.016 (−0.20)*
Stage 2 only	0.184 (0.04)*	0.168 (0.12)*	0.154 (0.03)*	0.116 (0.01)*
No context	0.175 (0.00)*	0.130 (0.00)*	0.150 (0.00)*	0.114 (0.00)*
Nemotron-3-Nano-4B, normalised by its own floor and ceiling				
MemToken 4×	0.301 (0.62)	0.322 (0.59)	0.141 (0.00)	0.175 (0.13)
MemToken 8×	0.246 (0.33)	0.306 (0.54)	0.136 (−0.03)	0.139 (0.04)

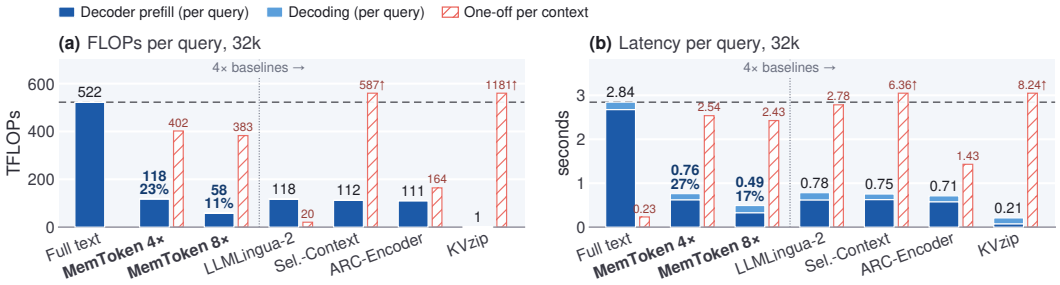


Figure 5: Answer-time cost on Qwen3.5-9B at 32K tokens, baselines at 4×. (a) FLOPs and (b) latency per query: solid bars recur at every query, with MemToken's share of full text, and hatched bars are paid once per context.

8× (Table 19). The stored memory is also small: 130 or 65 MB at 64K, 16 or 32 times less than the full-text KV cache. A token-dropping baseline at the same budget pays the same answer-time cost, so between the two the difference is the accuracy in Table 1. Building the memory costs 402 TFLOPs per 32K context at 4×, so a query that includes it costs about as much as reading the full text (520 against 522 TFLOPs). The baselines pay more for their own preprocessing: 1,181 TFLOPs for KVzip and 587 for Selective-Context.

6 Conclusion

In this work, we propose MemToken, a novel query-agnostic context compression method that aims to scale the inference ability of LLMs to long contexts. Existing methods overfit to specific tasks and are limited to compression lengths of hundreds or several thousand tokens. We propose the first training dataset for context compression, with 80B pretraining tokens and 7B supervised fine-tuning tokens, for downstream task generalization. Moreover, we propose a novel context compressor architecture and a two-stage training strategy to support compression of contexts up to 128K tokens. Extensive experiments on four benchmarks show that our method significantly outperforms state-of-the-art methods.

AI USE STATEMENT

Generative models were used in three roles, all disclosed here. (1) *Evaluation.* DeepSeek-V4-Flash serves as the LLM judge described in Section 5.1 and Appendix A.3; its labels are validated against human annotation and against a second, independent model annotation, and every headline number is also reported under the benchmark's official metric. (2) *Training data.* Questions in the Stage-2 training set were generated by an instruction-tuned language model and answers by the frozen decoder under training, as described in Section 4.3; no evaluation benchmark was used to generate training data. (3) *Writing and analysis.* An AI coding assistant (Claude) was used to draft text, write the scripts that produce every table and figure from the logged results, and act as the second annotator in the judge validation. All numbers in the paper were recomputed independently of the assistant's drafts from the per-item outputs, and the authors reviewed and are responsible for every claim.

ETHICS STATEMENT

This work compresses text for a language model to read; it does not train or release a new language model and does not change the decoder's behaviour on uncompressed input. All benchmarks are public and used under their licences; the training corpora are public web text, Wikipedia-derived question-answering data and permissively licensed code. Compressed representations are opaque, so a compressed document cannot be inspected by a human the way pruned text can; deployments that require auditability of what the model was shown should prefer token-level methods or keep the source alongside the vectors.

REPRODUCIBILITY STATEMENT

Appendix A lists the exact benchmark subsets, item counts, metrics and generation budgets (Table 3), the official scoring code used for each benchmark, the judge model, prompts and voting protocol, and the training hyperparameters and hardware. Every table and figure is generated by a script from a single results file; scripts, the results file, the trained compressors for all three backbones and both ratios, the evaluation harness including all baseline arms, and the per-item judged outputs will be released.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2308.14508.

Ziyang Chen, Xing Wu, Junlong Jia, Chaochen Gao, Qi Fu, Debing Zhang, and Songlin Hu. LongBench Pro: A more realistic and comprehensive bilingual long-context evaluation benchmark. *arXiv preprint arXiv:2601.02872*, 2026.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme context compression for retrieval-augmented generation with one token. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.13792.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv:2305.14788.

DeepSeek-AI. DeepSeek-V4, 2026. VERIFY citation; judge model DeepSeek-V4-Flash-0731.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks*, 2023. arXiv:2310.11248.

Yair Feldman and Yoav Artzi. No mean feat: Simple, strong baselines for context compression. *arXiv preprint arXiv:2510.20797*, 2025.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.06945.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv:2310.05736.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2310.06839.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. *arXiv preprint arXiv:2505.23416*, 2025.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. *arXiv preprint arXiv:2409.12941*, 2024.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context language models understand long contexts? *arXiv preprint arXiv:2311.04939*, 2023a.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023b. arXiv:2310.06201.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a. arXiv:2404.14469.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xCompressor: Generalized prompt compression for large language models. *arXiv preprint arXiv:2408.03094*, 2024b.

Fanfan Liu and Haibo Qiu. Context cascade compression: Exploring the upper limits of text compression. *arXiv preprint arXiv:2511.15244*, 2025.

Maxime Louis, Hervé Déjean, and Stéphane Clinchant. PISCO: Pretty simple compression for retrieval-augmented generation. *arXiv preprint arXiv:2501.16075*, 2025.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2402.17753.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2304.08467.

NVIDIA. NVIDIA Nemotron 3 Nano, 2026. VERIFY citation.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL*, 2024. arXiv:2403.12968.

Aleksandar Petrov, Mark Sandler, Andrey Zhmoginov, Nolan Miller, and Max Vladymyrov. Long context in-context compression by getting to the gist of gisting. *arXiv preprint arXiv:2504.08934*, 2025.

Hippolyte Pilchen, Edouard Grave, and Patrick Pérez. ARC-Encoder: Learning compressed text representations for large language models. *Transactions on Machine Learning Research (TMLR)*, 2025. arXiv:2510.20535.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023. arXiv:2211.05102.

Qwen Team. Qwen3.5 technical report, 2026. VERIFY citation.

Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. ZeroSCROLLS: A zero-shot benchmark for long text understanding. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023. arXiv:2305.14196.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1706.03762.

Yudong Wang, Zi-Xuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, et al. Ultra-FineWeb: Efficient data filtering and verification for high-quality LLM training data. *arXiv preprint arXiv:2505.05427*, 2025.

Haoran Wei, Yaofeng Sun, and Yukun Li. Deepseek-ocr: Contexts optical compression. *arXiv preprint arXiv:2510.18234*, 2025.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with compression and selective augmentation. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.04408.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2306.14048.

A Experimental Details

A.1 Baselines

All baselines are query-agnostic and receive the same token budget as MemToken on the LLM's tokenizer, measured on the compressed input rather than requested from the method. First+last truncation keeps half of the budget from each end of the context. Selective-Context (Li et al., 2023b) drops tokens by self-information, and LLMLingua-2 (Pan et al., 2024) drops them with a learned token classifier. KVzip (Kim et al., 2025) evicts key-value entries after a context-reconstruction pass, and its ratio is defined on KV-cache bytes. ARC-Encoder (Pilchen et al., 2025) is a trained encoder, which we retrain for each LLM with the authors' recipe (Appendix B.2). The no-context input is the query without the compressible context, and the full-text input keeps that context uncompressed.

A.2 Evaluation protocol

Two lanes of ZeroSCROLLS (SpaceDigest and BookSumSort) are excluded from every aggregate because their full-text ceiling does not exceed their own null baseline (a constant prediction and a random permutation respectively); the exclusion rule was fixed before any compressed arm was scored and removes two lanes on which MemToken scores above the ceiling. On LongBench the aggregate is the mean over all 29 lanes on every backbone, and the per-lane floor–ceiling gate is

Table 3: Evaluation protocol. n is scored items per arm; "max new" the generation budget; metrics are each benchmark's own scoring code.

Benchmark	Subset	n	Metric	Max new	Notes
LongBench	29 English lanes: 16 tasks and 13 LongBench-E variants	7,418	official per lane (F1, ROUGE-L, edit sim., accuracy, classification); LLM-judge accuracy on the nine QA lanes (1,974 items) in Table 1	128; 1,024 on the four summary lanes	total = mean over lanes
LooGLE	shortdep + longdep QA	3,052	LLM-judge accuracy (token-F1 in appendix)	64	first line kept
CrossCodeEval	Python, Java, TypeScript, C#	9,928	exact match, mean over languages	64	cross-file context compressed; current file uncompressed
ZeroSCROLLS	7 of 9 lanes	3,190	official per lane, mean over lanes	512	appendix, 2B; two degenerate lanes excluded
FRAMES	own Wikipedia articles, no padding; 14 of 824 over the 65,536-token cap dropped	810	LLM-judge accuracy in Table 1; SQuAD F1 in the appendix	48	median 14.5k context tokens

diagnostic only: the RepoBench-P lanes fail it on both Qwen LLMs (spans below 6 points) and the LongBench-E LCC lane on Qwen3.5-2B (8.2), the two TriviaQA lanes bypass the compressor (Appendix B.1), and the two passage-count lanes have single-token references on which every metric reduces to exact match; the 2B spans are listed in Table 10. Scoring uses each benchmark's official code. For LongBench we import the official `metrics.py` and `eval.py` unchanged: F1 for question answering; ROUGE-L from the `rouge` package the official code depends on rather than Google's `rouge_score`, which scores the same pair differently (0.667 against 0.615 for "the cat sat on the mat" against "a cat was sitting on the mat"); edit similarity through `fuzzywuzzy` on the first non-comment line for the code lanes; the retrieval and count scores for the synthetic lanes; and the classification score for TREC against the task-level set of 50 labels. As in the official evaluation, only the first line of a prediction is scored on the TREC and SAMSum lanes while the report-summarisation lanes use the whole prediction, and a lane's score is the mean over items of the maximum over references. The official code reports no single total, so the 29-lane mean is our aggregation; the two MultiFieldQA lanes hold the same 150 items, so that task enters the mean twice, which changes no ordering on the 9B LLM and swaps one adjacent pair on the 2B LLM (Selective-Context $4\times$ and ARC-Encoder $8\times$, R 0.12 against 0.12 on 29 lanes).

Decoding and statistics. All methods use greedy decoding, the same instruction template and the per-task budgets of Table 3. Our templates differ from the official prompts, so absolute scores are not comparable with leaderboards, and every comparison is between methods that share the prompt, the items and the LLM. Paired statistics are computed against MemToken: a paired t-test on continuous metrics and an exact McNemar test on binary ones.

Normalised score. R normalises the task-mean score once rather than averaging per-task R (Appendix B.1).

A.3 LLM JUDGE

Token-overlap metrics penalise correct answers sized differently from the reference, and the full-text LLM writes longer answers than the compressed one: on LooGLE with Qwen3.5-9B this lifts MemToken $4\times$ above the full-text ceiling under token-F1 (Table 14). The judge replaces them on free-form question answering, and it comes from a model family different from every evaluated LLM. The judge is DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026) called through an OpenAI-compatible endpoint with its reasoning mode on and a 512-token output budget. Every (item, arm) pair is judged three times at temperature 0.6 and the majority label is used; a vote whose output carries no label abstains, and an item with three abstentions is counted as wrong. The judge receives the question with our instruction suffix removed, all reference answers, and the prediction exactly as the official metric sees it; it never sees the arm or the backbone, and the (item, arm, vote) triples are submitted in one fixed random order. For LooGLE we use the benchmark's own judging prompt; for the other question-answering benchmarks we use a rubric that accepts paraphrase and length differences, requires facts, numbers and dates to match, and counts empty, evasive or off-target answers as wrong.

On the LooGLE 9B batch (7 arms, 3,052 items, 64,092 calls) the three votes were unanimous on 89.7%–95.1% of items per arm, 0.5%–1.0% of votes abstained, and five items in total had three abstentions. Re-judging 500 LooGLE items on all seven 9B arms with fresh votes reproduces the majority label on 97.8% of the 3,500 (item, arm) pairs, with 39 flips in each direction (no drift between rounds, binomial $p = 1.0$); 75% of the flips fall on the 6.9% of pairs whose first-round votes were split 2–1, which flip 24% of the time against 0.6% for unanimous pairs, so the unanimity flag is a per-item confidence marker. The judge label correlates with answer length on the accurate arms (Pearson 0.32 on full text, 0.31 on MemToken 4×, at most 0.12 on the other arms); controlling for token-F1 does not remove the correlation, but token-F1 is itself length-biased in the opposite direction, so LooGLE alone cannot separate a length preference of the judge from long answers being more often correct. On the LongBench question-answering lanes, judged with our rubric, the same correlation is between -0.06 and 0.00 on every arm and the judge agrees with the official metric on 82%–93% of items, so a general verbosity preference of the judge is unlikely. Re-judging 2,596 LooGLE predictions of the full-text and MemToken 4× arms under our rubric instead of LooGLE's prompt changes 3.4%–4.3% of labels and leaves the length correlation unchanged (0.35 and 0.29), and on both benchmarks the label is uncorrelated with the length of the reference answer ($|r| \leq 0.05$ on LooGLE), so neither the prompt nor reference length explains the correlation. The remaining explanation is the task itself: on LooGLE's free-form questions the LLM writes more when it can answer and a short evasive sentence when it cannot, so answer length carries information about correctness there and not on the short extractive LongBench answers.

Validation of the judge. We drew 400 judged items stratified by benchmark (LooGLE on all three backbones, LongBench-QA on 9B), arm group (no context, full text, baselines, MemToken), judge label, prediction length (split at each arm's own median), and whether the judge and the official metric disagree; disagreements were oversampled to 44% of the sample against 21% in the population, because agreements carry little information about which metric to trust. Items were labelled from the question, the reference answers and the prediction only, without the arm, the judge label or the official score. A second, independent annotation by a different model family from the judge (the writing assistant used for this paper, disclosed in the LLM-usage statement) agrees with the judge on 91.0% of items ($\kappa = 0.82$), and on 82.5% of the 177 disagreements sides with the judge over the official metric (146 vs. 31, binomial $p = 5 \times 10^{-19}$); agreement with the official metric binarised at 0.5 is 0.64 (0.62 if the 9 items with a score of exactly 0.5 are counted as wrong). Agreement is 0.90 on MemToken's arms, between the 0.95 on baselines and the 0.90 on the full-text arm, so the judge favours no arm group. On long predictions the judge is noisier (error rate 13.9% vs. 5.0% on short ones) but errs towards *wrong* (14 false negatives vs. 11 false positives, against 2 vs. 9 on short predictions), which is the opposite of what a verbosity preference predicts. Because the second annotator is also a language model, the two could share a bias; a shared preference for long answers would however show up as joint over-crediting of long predictions, not as the judge being stricter on them, so we read this as evidence against the bias rather than proof. An author additionally inspected a sample of judged items across arms and found no systematic error; no complete human annotation of the 400 items was performed, and we do not claim one. The judge axis is therefore adopted on the strength of the cross-model agreement ($\kappa = 0.82$), the self-consistency check and the length analysis above, with the official metric reported alongside it everywhere (Table 1).

Prompts and vote cache. The rubric prompt is a single user turn, reproduced here as sent:

> Your task is to label an answer to a question as CORRECT or WRONG.
> You will be given: (1) a question posed by one user to another user, (2) a gold (ground truth) answer, (3) a generated answer which you will score.
> The question asks about something one user should know about the other from their prior conversations. The gold answer is usually short and names the referenced fact. The generated answer may be longer or phrased differently; be generous with your grading: as long as it conveys the same fact as the gold answer, label it CORRECT.
> For time-related questions the gold answer is a specific date, month or year. Be generous about format ('May 7th' vs '7 May 2023') and about relative references, but the referenced date or period must be the same; a different date is WRONG.
> If the generated answer is empty, evasive, or names a different fact, label it WRONG.
>
> Question: {q}
> Gold answer: {g}
> Generated answer: {p}

> On the FIRST line return ONLY a JSON object, {{"label": "CORRECT"}} or {{"label": "WRONG"}}. Then give a one-sentence explanation.
> QUESTION: ⟨question⟩
> GOLD ANSWER: ⟨reference⟩
> GENERATED ANSWER: ⟨prediction⟩

The rubric was written for the memory-benchmark literature, whose judges share it, and its wording about "users" and "conversations" was left unchanged for every benchmark; the line with the three placeholders is a template the calling code does not fill, so the judge sees it verbatim, followed by the actual question, references and prediction under the capitalised headers (with several references the header reads "GOLD ANSWERS (the generated answer is CORRECT if it matches ANY of them)" and lists them one per line). Both quirks are identical for every arm, item and vote, and the validation of the previous paragraph was run on the labels this prompt produced. For LooGLE the system prompt is the benchmark's own, "Given one question, there is a groundtruth and a predict_answer. Please decide whether they are the same or not in semantic. Please only output 'True' or 'False'.", and the user turn is "Question: ⟨question⟩ / groudtruth = ⟨reference⟩ / predict_answer = ⟨prediction⟩" on three lines, typo included; the departures from LooGLE's call are the output budget (512 tokens instead of 10, so that reasoning fits) and three votes at temperature 0.6 instead of one at 0. Votes are cached under the SHA-256 of the full message list together with the vote index, so byte-identical predictions from different arms on the same item share votes instead of being judged twice; on the LongBench-QA 9B batch 19% of the 41,625 distinct prompts were shared by two or more arms, which saved 15,309 of 56,934 votes, and 162 (item, vote) keys had seven of eight arms produce the same answer.

A.4 Training data

Stage 1. The prepared pool contains 2,430,000 Ultra-FineWeb-derived contexts (Wang et al., 2025, Apache-2.0): 2,147,971 original records and 282,029 contexts made by concatenating shorter records. A full scan of the stored `qwen3_5_tokens` column, counted before training-time text sanitization, gives 39,832,077,525 source tokens, a median of 3,593 per context, and 2,430,000 distinct identifiers; length bins contain 250,000 records each except the longest, which contains 180,000. The two longest bins, 32,768–131,071 tokens, contain 17.7% of contexts and 72.2% of source tokens; concatenated contexts appear only above 16,383 tokens. Rendering a reconstruction sample repeats the source as the answer, so the full prepared pool corresponds to approximately 79,742,328,150 chat-formatted sequence tokens, estimated from the source-token sum and the mean template overhead measured on 300 records sampled evenly across the ten bins. The validation set is split from the same length-stratified sample before the training pool is capped, and a seeded draw keeps 10,000 contexts per bin. Its 100,000 contexts hold 1,856,599,552 source tokens with a median of about 4,096 per context, and its 13,650 concatenated contexts all lie above 16,383 tokens. No identifier is shared with the training pool, either between stand-alone records or among the source records of concatenated contexts.

The reported pretraining runs retain 1,500,000 contexts shorter than 8,192 tokens from that pool; filtering does not truncate any context. These contexts contain 2,805,647,373 source tokens by the stored column, or approximately 5,659,549,746 assembled sequence tokens by the same template-overhead estimate. Validation applies the same length filter, so these runs track reconstruction loss and per-bin reconstruction precision on the 60,000 validation contexts shorter than 8,192 tokens, of which 59,927 enter the Nemotron runs. The on-disk training order starts with 80,000 contexts of 128–255 tokens and then mixes the six eligible bins, with 250,000 contexts in each bin overall; there is no subsequent progressive bin activation. Training makes one pass without repeats, assigning 187,500 contexts to each of eight ranks, and draws the repetition instruction per sample from 29 paraphrases.

Stage 2. The SFT set combines HotpotQA training passages (Yang et al., 2018), rendered as numbered and titled Wikipedia passages, whole Python files from `codeparrot/codeparrot-clean`, synthetic in-context demonstration blocks, and reconstruction replay samples. The demonstration blocks sample a rule or label mapping afresh for each item, so the answer depends on the demonstrations in that item's context. GLM-5.2 writes all questions, paraphrases and two-hop questions under guided JSON decoding and also performs

Table 4: Composition of the Stage-2 training set (Section 4.2). Rows are the baked prompts of the union set; the share is of its 446,093 rows. Question answering and positional lookup use multi-passage Wikipedia contexts; induction uses synthetic demonstrations, code understanding uses whole source files, and replay uses web text. The 9B and Nemotron arms train on the same set filtered to the decoder's context cap, which removes 852 rows at 65,536 tokens.

Group	Task	Rows	Share
Wikipedia question answering	Relational lookup	124,675	27.9%
	Numeric and date lookup	80,504	18.0%
	Paraphrased question	51,094	11.5%
	Whole-sentence extraction	39,299	8.8%
	Two-hop bridge	11,590	2.6%
	Named-entity or date fact	4,577	1.0%
Positional lookup	Title to passage index	9,957	2.2%
	Sentence to passage index	9,833	2.2%
	Passage index to title	9,824	2.2%
	Question to passage index	9,685	2.2%
In-context induction	Rule induction	20,000	4.5%
	Label induction	20,000	4.5%
Code understanding	Returned expression	1,829	0.4%
	Called function	1,599	0.4%
	Raised exception	758	0.2%
	Default argument value	669	0.1%
	Defining file path	568	0.1%
	Base class	510	0.1%
Reconstruction replay	Verbatim repeat	49,122	11.0%
Total		446,093	100.0%

model-based validation and filtering. For the later 107,216 of the 156,089 accepted questions, GLM-5.2 judges each question four times, and it is kept only if all four verdicts pass it, with no tiebreak, so disagreement rejects. The first 48,873 (31.3%) needed two passing verdicts, with a third breaking disagreements. The later rule is the stricter one, since it needs four passing verdicts where the earlier one needed two. Mechanical checks run before the model panel and reject a question whose quote is not verbatim in the target passage, whose answer appears in the question, or which refers to a passage by number. Code questions do not use that panel because their gold answer is read off the abstract syntax tree, and GLM-5.2 only phrases the question around it. For a leakage check, GLM-5.2 is asked each code question without the code, and a question it answers is discarded, which removes 29% of the exception-raising items, 14% of the default-value items and 7% of the base-class items. The same model then audits each surviving question against the code for self-containment and answer leakage, without re-judging the answer. Answers are self-distilled: the frozen Qwen3.5-2B, prompted for the shortest exact span, answers each question greedily from the uncompressed document, and its wording replaces the gold answer when the two agree at token-F1 of at least 0.6. Otherwise the original gold answer is retained. This replaced 105,629 of the 136,197 generated Wikipedia answers (77.6%), and the same targets are used for every backbone, including the 9B LLM and Nemotron. The evidence segment is assigned round-robin over the passages of a context rather than left to the generator, which otherwise draws almost every question from the lead paragraph. Length is added deliberately: each prose context also enters as a diluted copy, padded with distractor passages drawn from other screened contexts until the gold fraction reaches 0.10, 0.05 or 0.02, which takes the pool from 55,758 to 108,800 contexts. The passages of a diluted context are shuffled rather than appended, so the evidence is not always early. A donor is rejected if it repeats a title already present or contains any target answer of at least six characters, which discarded 9,602 and 33,632 candidate donors respectively. Table 4 gives the 19 task types of the baked set and Table 5 its length composition. The set holds 446,093 samples and 6,942,674,134 tokens including instructions, questions, answers and chat-template tokens, of which 6,730,513,448 are context tokens, and the 9B and Nemotron arms train on the same rows filtered to a 65,536-token cap.

Table 5: Length composition of the samples used in the two reported training stages, as a percentage of rows per bin. Bins are measured on the context alone with the Qwen3.5 tokenizer. The Stage-1 row describes the selected 1,500,000 contexts, not the full 2,430,000-context prepared pool; filtering at 8,192 tokens does not truncate a context. Stage-2 long contexts include diluted prose, synthetic demonstrations, and code files (Appendix A.4).

Stage	<0.25k	0.25–0.5k	0.5–1k	1–2k	2–4k	4–8k	8–16k	16–32k	32–64k	64–128k
1 (1,500,000 contexts)	16.7	16.7	16.7	16.7	16.7	16.7	–	–	–	–
2 (446,093 prompts)	0.1	2.0	7.4	35.3	5.4	4.9	13.9	14.2	16.7	0.2

Separation from the benchmarks. HotpotQA training and validation passages are drawn from the same Wikipedia pool, so a paragraph that is gold evidence for a benchmark item appears routinely as a distractor in a training context, and CodeParrot is built from the same public GitHub as the LongBench code lanes. Both are removed before any bake rather than accepted as unavoidable. Offending paragraphs are deleted from the training context instead of the record being dropped, because a context with one contaminated distractor still carries nine clean passages: this rewrote 16,821 of 55,921 contexts, deleted 22,312 paragraphs, and dropped 63 prose and 100 code contexts outright, leaving 146,564 of 162,587 items. Passage numbering, evidence indices and bin assignments are recomputed afterwards, and an item whose own evidence paragraph was deleted is dropped. Disjoint contexts do not make questions disjoint, so every training question was also compared against the 8,833 reachable benchmark questions by word 5-gram Jaccard overlap. Of 140,456 questions scored none matched exactly, none reached the 0.5 threshold, and the highest observed overlap fell in the 0.4 bucket. We use overlap of verbatim 5-grams rather than embedding distance here because the paraphrase lane is designed to be semantically close to some question, which leaves an embedding threshold with no defensible cut point.

A.5 Training and hardware

Compressor configuration. In the reported experiments, we set $\ell = \lfloor 2L/3 \rfloor + 1$, placing the compression layer after the backbone block at zero-based index $\lfloor 2L/3 \rfloor$. The initial value s_0 is the ratio of the expected LLM input-embedding magnitude to the expected unscaled compressor-output magnitude, with both expectations estimated over all tokens. This gives $s_0 = 1/230$ for the Qwen backbones and $s_0 = 1/145$ for Nemotron. The scalar remains learnable during training.

Hardware. All training and evaluation ran on nodes of eight NVIDIA A800-SXM4-80GB GPUs. Inference runs one process per GPU without model parallelism. The longest document validated end to end through compressor and LLM is 258,048 tokens on one GPU for both Qwen LLMs, with peak memory 26.8 GB on 2B and 69.0 GB on 9B (12.7 and 17.5 GB at 65,536 and 131,072 tokens on 2B; 42.5 and 51.5 GB on 9B); above 131,072 tokens the 9B LLM needs the whole GPU, where the 2B LLM still fits two processes.

Optimization. Stage 1 uses 1,500,000 contexts for 11,718 optimizer updates with AdamW at learning rate 10^{-4} and a warmup–cosine learning-rate schedule. Each backbone and compression ratio is trained for one pass in Stage 2 with AdamW at learning rate 3×10^{-5}, using 6,970 optimizer updates on 8 GPUs or 6,956 on 16 GPUs. Stage-2 contexts are capped at 131,072 tokens on the 2B LLM and at 65,536 on the 9B LLM and Nemotron, which drops 852 samples for the latter two models.

Stage-1 loss ablation. We analyze two auxiliary context-distillation losses alongside the cross-entropy objective in Equation 4. The teacher is the same frozen LLM run without gradients on the uncompressed context and the same query and response. Its outputs are aligned with the student's at the supervised response positions S, including the end-of-turn token. Let p_t^{T} and p_t^{S} be the teacher and student next-token distributions of the t-th token over $\mathcal{V}$, and let $h_{\ell,t}^{\mathrm{T}}$ and $h_{\ell,t}^{\mathrm{S}}$ be their hidden

Table 6: Table 1 under each benchmark's official metric: LongBench per-task metrics over all 29 tasks, LooGLE LLM-judge accuracy, CrossCodeEval exact match, FRAMES SQuAD F1. R normalises the task-mean score once (Appendix B.1). Bold marks the best method per column and ratio. †Ratio on KV-cache bytes. ‡Trained baseline.

	Qwen3.5-9B				Qwen3.5-2B			
Method	LongBench	LooGLE	CCEval	FRAMES	LongBench	LooGLE	CCEval	FRAMES
No context	0.206 (0.00)	0.172 (0.00)	0.035 (0.00)	0.143 (0.00)	0.175 (0.00)	0.114 (0.00)	0.005 (0.00)	0.094 (0.00)
Full text	0.499 (1.00)	0.706 (1.00)	0.242 (1.00)	0.460 (1.00)	0.393 (1.00)	0.590 (1.00)	0.044 (1.00)	0.347 (1.00)
4× compression								
First+last	0.366 (0.55)	0.256 (0.16)	0.103 (0.33)	0.336 (0.61)	0.288 (0.52)	0.246 (0.28)	0.015 (0.26)	0.248 (0.61)
Selective-Context	0.246 (0.14)	0.203 (0.06)	0.060 (0.12)	0.266 (0.39)	0.202 (0.12)	0.136 (0.05)	0.004 (−0.02)	0.161 (0.26)
LLMLingua-2	0.291 (0.29)	0.352 (0.34)	0.051 (0.07)	0.302 (0.50)	0.230 (0.25)	0.238 (0.26)	0.004 (−0.01)	0.209 (0.46)
KVzip†	0.384 (0.61)	0.476 (0.57)	**0.221** (0.89)	0.275 (0.42)	0.214 (0.18)	0.182 (0.14)	0.020 (0.40)	0.116 (0.09)
ARC-Encoder‡	0.310 (0.36)	0.391 (0.41)	0.108 (0.35)	0.269 (0.40)	0.199 (0.11)	0.201 (0.18)	0.005 (0.01)	0.188 (0.37)
MemToken (ours)	**0.440** (0.80)	**0.582** (0.77)	0.191 (0.75)	**0.352** (0.66)	**0.367** (0.88)	**0.438** (0.68)	**0.040** (0.89)	**0.274** (0.71)
8× compression								
First+last	0.319 (0.39)	0.165 (−0.01)	0.069 (0.17)	0.284 (0.45)	0.252 (0.35)	0.178 (0.14)	0.009 (0.12)	0.210 (0.46)
Selective-Context	0.215 (0.03)	0.118 (−0.10)	0.042 (0.03)	0.230 (0.27)	0.186 (0.05)	0.106 (−0.02)	0.003 (−0.04)	0.137 (0.17)
LLMLingua-2	0.242 (0.12)	0.186 (0.03)	0.037 (0.01)	0.253 (0.35)	0.197 (0.10)	0.154 (0.09)	0.003 (−0.03)	0.152 (0.23)
KVzip†	0.253 (0.16)	0.152 (−0.04)	0.114 (0.38)	0.157 (0.04)	0.183 (0.03)	0.118 (0.01)	0.008 (0.09)	0.096 (0.01)
ARC-Encoder‡	0.277 (0.24)	0.340 (0.31)	0.066 (0.15)	0.261 (0.37)	0.202 (0.12)	0.139 (0.05)	0.010 (0.15)	0.161 (0.26)
MemToken (ours)	**0.401** (0.66)	**0.482** (0.58)	**0.166** (0.63)	**0.338** (0.62)	**0.328** (0.70)	**0.278** (0.34)	**0.031** (0.66)	**0.227** (0.53)

states at the ℓ-th LLM layer of width d. The KL divergence loss and hidden-state distillation loss are

$$\ell_{\mathrm{KL}} = \frac{1}{|S|} \sum_{t \in S} \mathrm{KL}\big(p_t^{\mathrm{T}} \,\|\, p_t^{\mathrm{S}}\big)\,,$$
$$\ell_{\mathrm{hid}} = \frac{1}{2|S|d} \sum_{\ell \in \{-1,-2\}} \sum_{t \in S} \big\| h_{\ell,t}^{\mathrm{S}} - h_{\ell,t}^{\mathrm{T}} \big\|_2^2\,, \tag{8}$$

where $\mathrm{KL}(\cdot \,\|\, \cdot)$ denotes the KL divergence between two distributions. The hidden-state term averages the mean squared error of the last two returned decoder states. The optional per-sample objective is $\ell_{\mathrm{total}} = \ell_{ce} + \lambda_{\mathrm{KL}}\ell_{\mathrm{KL}} + \lambda_{\mathrm{hid}}\ell_{\mathrm{hid}}$, followed by an average over samples in the effective batch. Both auxiliary weights default to zero, so the teacher pass is skipped and the reported main runs optimize cross-entropy alone.

The three arms of Figure 6 use Qwen3.5-2B at $4\times$, see the same training samples in the same order, and complete the same learning-rate schedule over 30,720 micro-steps (1,920 optimizer updates). The auxiliary weights, 0.25 for KL and 0.1 for hidden-state distillation, are set so that each added term accounts for about one fifth of the initial loss. The final checkpoints are evaluated on 60,000 held-out reconstruction contexts in six length bins, and validation during training uses 6,000 contexts at six checkpoints. The comparison has a single seed, and the KL arm also uses a different CE kernel, so its small gap from CE cannot be attributed to the auxiliary objective alone.

B Additional Results

B.1 The main table under the official metrics

Table 6 repeats Table 1 with LongBench scored by its official per-task metrics over all 29 tasks instead of the LLM judge on the nine question-answering tasks; the ordering of the compression methods is the same on both backbones. R is computed by normalising the task-mean score once against the no-context and full-text task means. The mean of per-task R is not equivalent: on tasks whose full-text margin over no context is small, per-task R is unbounded, and averaging it reorders the arms (on Qwen3.5-9B at $4\times$ it places KVzip at 0.68 above MemToken at 0.65, against 0.61 and 0.80 under the definition used here). The two TriviaQA tasks carry no usable R at all: full text beats no context by 1.0 points on one and loses by 0.8 on the other, consistent with the document sitting in the question field and bypassing the compressor; dropping both moves every arm's aggregate R by less than 0.004. The two gov_report tasks are the only place where a generation cap binds: MemToken's summaries reach the 1,024-token cap on 64% and 68% of items while no

Table 7: Query-agnostic context compression on Qwen3.5-2B (full comparison). Cells give the official metric and, in parentheses, R: LongBench task-mean official metrics, LooGLE token-F1, ZeroSCROLLS the 7-lane official mean (two lanes fail the admission gate). Bold marks the best method per column and ratio. †Ratio on KV-cache bytes. ‡Trained baseline (about 300 encoder tokens per sequence). Appendix B.3 reads the ZeroSCROLLS column.

Method	LongBench	LooGLE (token-F1)	ZeroSCROLLS
No context	0.175 (0.00)	0.150 (0.00)	0.0894 (0.00)
Full text	0.393 (1.00)	0.295 (1.00)	0.2184 (1.00)
4× compression			
First+last truncation	0.288 (0.52)	0.182 (0.22)	**0.1752** (0.67)
Selective-Context	0.202 (0.12)	0.141 (−0.06)	0.1507 (0.48)
LLMLingua-2	0.230 (0.25)	0.174 (0.17)	0.1735 (0.65)
KVzip†	0.214 (0.18)	0.169 (0.13)	0.1254 (0.28)
ARC-Encoder‡	0.199 (0.11)	0.198 (0.33)	0.1447 (0.43)
MemToken (ours)	**0.367** (0.88)	**0.275** (0.86)	0.1663 (0.60)
8× compression			
First+last truncation	0.252 (0.35)	0.156 (0.04)	**0.1471** (0.45)
Selective-Context	0.186 (0.05)	0.132 (−0.13)	0.1355 (0.36)
LLMLingua-2	0.197 (0.10)	0.151 (0.01)	0.1467 (0.44)
KVzip†	0.183 (0.03)	0.151 (0.01)	0.1039 (0.11)
ARC-Encoder‡	0.202 (0.12)	0.164 (0.10)	0.1359 (0.36)
MemToken (ours)	**0.328** (0.70)	**0.220** (0.48)	0.1432 (0.42)

other arm exceeds 2.4%. Regenerating MemToken and KVzip with a 2,048-token cap changes their LongBench totals by at most 0.01 points (paired t over the 994 summary items, all $p > 0.49$): the compressed decoder writes twice as much and earns no more ROUGE-L, so the pre-registered budget is kept.

B.2 ARC-ENCODER

ARC-Encoder's nominal 4× and 8× measure 4.15× and 8.27× on the decoder's tokens, 3–4% fewer vectors than MemToken, so a small part of any deficit is budget. On the 9B decoder its position relative to the untrained baselines changes with the benchmark: below first+last $8\times$ on the 25 LongBench lanes outside the four summarisation lanes under the official metrics (t $p = 9 \times 10^{-6}$, Wilcoxon $p = 3 \times 10^{-8}$); on the nine judged LongBench tasks tied with first+last $4\times$ and LLMLingua-2 $4\times$ (McNemar $p = 0.07$ and 0.87) and below KVzip $4\times$ ($p = 4 \times 10^{-4}$); tied with first+last $4\times$ on CrossCodeEval ($p = 0.53$); above LLMLingua-2 and first+last on LooGLE. Its ratio order depends on the decoder and on the task set: on the 2B decoder $8\times$ is above $4\times$ over all 29 LongBench tasks under the official metrics (paired difference $+0.7$ points, 1,662 items better against 1,113 worse, Wilcoxon $p = 6 \times 10^{-14}$, t test $p = 7 \times 10^{-3}$: a small difference spread over many items) and on CrossCodeEval ($p = 2 \times 10^{-7}$), but $4\times$ is above $8\times$ on the nine question-answering tasks of Table 1 under both the official metric ($p = 0.006$) and the judge (McNemar 189 vs. 133, $p = 0.002$). The two lane sets differ in what they contain: the 29-lane total includes the four code lanes, on which the 2B decoder's full-text gain over no context averages 4.2 points (LCC 12.0 and 8.2, RepoBench -2.5 and -0.9) and MemToken $4\times$ itself averages below the no-context row (27.8 against 29.4), so on this decoder they do not measure compression fidelity; the RepoBench lanes fail the per-lane gate on the 9B decoder as well (spans 4.5 and 5.8 points), where LCC passes. The ratio order of ARC-Encoder is therefore a property of the lane set; we name the lane set with every statement about it and claim no mechanism. On the 2B decoder it is again indistinguishable from first+last and LLMLingua-2 at $4\times$ ($p = 0.78$ and 0.41).

B.3 FULL COMPARISON ON QWEN3.5-2B

Table 7 adds ZeroSCROLLS to the 2B comparison, and it is the one column where MemToken is behind both first+last truncation and LLMLingua-2. Two properties of that column carry the result: the 7-lane unweighted mean gives the 19-item qasper lane the same weight as the 500-item musique

Table 8: LongBenchby task family under the official metrics on the two Qwen decoders. A family's score is the unweighted mean over its lanes, LongBench-E variants included (lane counts in the header), and R is normalised by the family's own no-context and full-text rows; Total is the 29-lane mean of Tables 1 and 6. Bold marks the best compression method per family and ratio. †Ratio on KV-cache bytes. ‡Trained baseline.

Input	Single-doc QA (5)	Multi-doc QA (5)	Summarisation (5)	Few-shot (6)	Synthetic (4)	Code (4)	Total (29)
Qwen3.5-9B							
No context	10.3 (0.00)	22.4 (0.00)	7.2 (0.00)	40.0 (0.00)	1.8 (0.00)	37.5 (0.00)	20.6 (0.00)
Full text	40.3 (1.00)	54.5 (1.00)	25.6 (1.00)	66.2 (1.00)	57.6 (1.00)	54.7 (1.00)	50.0 (1.00)
4× compression							
First+last	26.4 (0.54)	32.0 (0.30)	**22.2** (0.81)	**63.4** (0.89)	18.8 (0.30)	50.9 (0.78)	36.6 (0.55)
Selective-Context	15.6 (0.18)	24.5 (0.07)	17.6 (0.56)	41.8 (0.07)	4.5 (0.05)	39.3 (0.10)	24.6 (0.14)
LLMLingua-2	24.6 (0.48)	35.4 (0.40)	21.7 (0.79)	39.9 (−0.00)	14.6 (0.23)	34.5 (−0.18)	29.1 (0.29)
KVzip†	31.2 (0.70)	37.7 (0.48)	21.1 (0.76)	58.3 (0.70)	26.5 (0.44)	**52.0** (0.84)	38.4 (0.61)
ARC-Encoder‡	26.8 (0.55)	37.7 (0.48)	17.7 (0.57)	42.9 (0.11)	23.7 (0.39)	34.1 (−0.20)	31.0 (0.36)
MemToken	**32.6** (0.74)	**56.0** (1.05)	21.3 (0.77)	61.3 (0.81)	**50.8** (0.88)	38.4 (0.05)	**44.0** (0.80)
8× compression							
First+last	19.2 (0.30)	25.4 (0.09)	19.8 (0.68)	**61.7** (0.83)	9.0 (0.13)	**49.4** (0.69)	31.9 (0.39)
Selective-Context	10.6 (0.01)	20.6 (−0.06)	13.5 (0.34)	40.6 (0.02)	1.9 (0.00)	37.1 (−0.03)	21.5 (0.03)
LLMLingua-2	17.3 (0.23)	24.7 (0.07)	19.0 (0.64)	39.8 (−0.01)	8.3 (0.12)	31.1 (−0.37)	24.2 (0.12)
KVzip†	14.1 (0.13)	26.8 (0.14)	14.7 (0.41)	40.3 (0.01)	7.1 (0.09)	46.5 (0.52)	25.3 (0.16)
ARC-Encoder‡	24.5 (0.47)	34.0 (0.36)	15.7 (0.46)	42.8 (0.10)	12.7 (0.19)	30.9 (−0.38)	27.6 (0.24)
MemToken	**25.3** (0.50)	**50.0** (0.86)	**20.7** (0.73)	54.0 (0.53)	**50.8** (0.88)	39.0 (0.09)	**40.1** (0.66)
Qwen3.5-2B							
No context	11.2 (0.00)	13.9 (0.00)	9.3 (0.00)	34.6 (0.00)	2.9 (0.00)	29.4 (0.00)	17.5 (0.00)
Full text	37.5 (1.00)	33.8 (1.00)	25.7 (1.00)	52.5 (1.00)	51.5 (1.00)	33.6 (1.00)	39.3 (1.00)
4× compression							
First+last	**23.9** (0.48)	21.4 (0.38)	**22.2** (0.79)	51.0 (0.92)	15.2 (0.25)	32.9 (0.83)	28.8 (0.52)
Selective-Context	14.6 (0.13)	17.2 (0.17)	18.6 (0.57)	34.1 (−0.02)	4.4 (0.03)	27.7 (−0.42)	20.2 (0.12)
LLMLingua-2	22.4 (0.42)	21.0 (0.36)	20.5 (0.69)	35.3 (0.04)	6.0 (0.06)	28.1 (−0.32)	23.0 (0.25)
KVzip†	15.4 (0.16)	18.7 (0.24)	15.3 (0.37)	35.9 (0.08)	6.2 (0.07)	**33.3** (0.91)	21.4 (0.18)
ARC-Encoder‡	18.3 (0.27)	26.0 (0.61)	12.2 (0.18)	31.7 (−0.16)	5.0 (0.04)	21.1 (−1.99)	19.9 (0.11)
MemToken	22.5 (0.43)	**44.3** (1.53)	21.6 (0.75)	**53.1** (1.03)	**48.5** (0.94)	27.8 (−0.39)	**36.7** (0.88)
8× compression							
First+last	18.5 (0.28)	17.6 (0.19)	19.9 (0.65)	**48.5** (0.77)	7.4 (0.09)	**32.7** (0.78)	25.2 (0.35)
Selective-Context	10.8 (−0.02)	15.5 (0.08)	15.7 (0.39)	34.7 (0.01)	3.6 (0.01)	27.0 (−0.58)	18.6 (0.05)
LLMLingua-2	16.1 (0.18)	16.4 (0.13)	17.5 (0.50)	34.7 (0.01)	4.0 (0.02)	24.1 (−1.27)	19.7 (0.10)
KVzip†	11.2 (−0.00)	15.9 (0.10)	9.6 (0.02)	34.2 (−0.02)	5.4 (0.05)	30.1 (0.17)	18.3 (0.03)
ARC-Encoder‡	17.9 (0.25)	22.5 (0.43)	14.5 (0.32)	32.3 (−0.13)	4.1 (0.02)	25.7 (−0.90)	20.2 (0.12)
MemToken	**20.2** (0.34)	**41.1** (1.37)	**23.3** (0.86)	43.7 (0.51)	**35.2** (0.66)	30.9 (0.34)	**32.8** (0.70)

lane, and the deficit itself is the length-calibration failure of Appendix B.9 rather than a loss of content, since the same arm is scored correct by the judge on the question-answering lanes. Table 8 groups the 29 LongBench lanes into the benchmark's six task families on both Qwen decoders, and Table 9 lists every lane on Qwen3.5-9B. The family view shows where the total comes from: at $4\times$ on the 9B decoder MemToken leads on single-document QA, multi-document QA and the synthetic lanes, and is behind first+last on summarisation (R 0.77 against 0.81) and the few-shot family (0.81 against 0.89) and behind KVzip on code (0.05 against 0.84); the 2B decoder repeats the pattern with a larger lead on multi-document QA and a code family whose full-text ceiling is 4.2 points above its floor. On multi-document QA MemToken is above the full-text row on both decoders (R 1.05 and 1.53), the family closest to the two-hop questions of the Stage-2 mix (Section 4.3). The two passage-count lanes are scored by exact match of a single integer: on the 2B decoder the no-context arm answers the constant "1" on all 500 items and the full-text ceilings are 4.5 and 6.0 points (9.0 and 21.3 on 9B), so the lanes separate nothing, and a substring-tolerant metric would have credited the constant on 38% of the items, which is why only the official count score is reported.

B.4 Stage-1 loss ablation

Adding KL or hidden-state distillation from the frozen LLM reading the full context to the Stage-1 objective (Equation 8) does not help (Figure 6, setup in Appendix A.5). KL ends at nearly the same held-out reconstruction cross-entropy as CE alone (0.01135 against 0.01076) at 19% more training time and about twice the peak memory, and hidden-state distillation stays about ten times higher (0.10451) and lowers accuracy in every length bin. The main runs therefore use CE alone, although this single-seed reconstruction comparison does not test downstream tasks.

Table 9: LongBenchon Qwen3.5-9B, lane by lane, under the official per-lane metric (F1, ROUGE-L, edit similarity, accuracy or classification score). Floor and ceiling are the no-context and full-text rows; the other columns are the 4× arms and MemToken at 8×. Bold marks the best 4× compression method on the lane; * marks the eight lanes whose ceiling exceeds the floor by less than 10 points and which therefore fail the diagnostic per-lane gate (Appendix A.2); the two MultiFieldQA lanes hold the same 150 items. †Ratio on KV-cache bytes. ‡Trained baseline.

Lane	n	Floor	Ceiling	First+last	SelCtx	LLMLingua-2	KVzip†	ARC‡	MemToken	MemToken 8×
Single-doc QA										
narrativeqa	200	6.1	25.3	17.3	10.9	14.2	**20.7**	17.4	19.3	14.6
qasper	200	11.1	33.5	19.3	15.4	24.6	23.5	19.9	**27.5**	21.4
qasper_e	224	11.0	38.2	23.1	14.6	23.6	**28.1**	23.7	25.9	20.8
multifieldqa_en	150	11.7	52.2	36.1	18.7	30.2	41.7	36.6	**45.1**	34.8
multifieldqa_en_e	150	11.7	52.2	36.1	18.7	30.2	41.7	36.6	**45.1**	34.8
Multi-doc QA										
hotpotqa	200	26.1	59.5	36.6	31.2	42.8	46.7	45.1	**65.1**	59.4
hotpotqa_e	300	23.8	60.0	40.2	25.2	39.7	44.7	45.5	**63.7**	60.9
2wikimqa	200	26.4	55.6	27.0	25.0	32.7	34.8	38.4	**57.9**	47.1
2wikimqa_e	300	24.5	52.4	31.6	26.1	34.5	37.5	38.7	**54.8**	47.3
musique	200	11.3	45.0	24.5	15.2	27.2	24.9	21.0	**38.8**	35.1
Summarisation										
gov_report	200	4.4	30.6	26.0	21.4	24.4	22.8	18.0	**28.6**	24.9
gov_report_e	300	4.6	30.0	25.5	21.1	24.6	22.8	18.5	**29.4**	25.2
qmsum*	200	14.8	21.9	19.2	17.8	19.7	**20.9**	15.4	9.5	16.3
multi_news	200	6.5	23.6	20.5	13.6	20.2	20.2	20.4	**21.2**	20.5
multi_news_e	294	6.0	22.0	**20.0**	14.1	19.5	19.1	16.2	18.0	16.6
Few-shot										
trec	200	0.2	73.0	**64.5**	5.0	1.0	54.8	2.5	58.5	40.5
trec_e	300	1.2	70.3	**65.3**	5.0	1.0	51.3	1.3	59.3	37.0
triviaqa*	200	89.3	90.3	89.6	89.1	88.4	**90.0**	88.9	88.6	88.8
triviaqa_e*	300	89.5	88.7	**89.1**	88.2	87.1	88.0	88.8	87.7	87.5
samsum*	200	31.2	38.9	36.5	32.8	32.0	34.2	**38.5**	38.1	36.3
samsum_e*	300	28.8	36.0	35.3	30.4	29.9	31.8	**37.3**	35.7	34.0
Synthetic										
passage_count*	200	0.0	9.0	**7.0**	2.0	5.0	2.5	3.5	2.0	1.0
passage_count_e	300	0.0	21.3	8.0	5.0	6.3	7.7	**8.7**	6.7	5.0
passage_retrieval_en	200	1.2	100.0	31.5	4.5	19.0	42.0	36.0	**96.0**	100.0
passage_retrieval_en_e	300	6.0	100.0	28.8	6.7	28.0	54.0	46.7	**98.7**	97.3
Code										
lcc	500	21.0	49.0	**44.8**	23.8	15.4	44.3	15.8	25.3	27.2
lcc_e	300	20.1	50.6	44.9	23.5	13.4	**45.3**	16.1	23.1	25.3
repobench-p*	500	56.0	60.4	57.5	55.1	55.8	**60.4**	54.0	54.3	52.3
repobench-p_e*	300	53.0	58.8	56.5	54.7	53.3	**58.2**	50.3	51.0	51.5
Mean over 29 lanes	7418	20.60	49.95	36.62	24.64	29.10	38.43	31.03	43.96	40.11

B.5 WHERE THE SELECTOR MATTERS, LANE BY LANE

Table 10 decomposes the selector ablations of Table 2 over the 29 LongBench lanes. The effect is concentrated rather than diffuse: on `passage_retrieval_en`, whose full-text gain over no context is 92.0 points, the first-token variant loses 39.0 points and the shuffled variant 17.5, against 29-lane aggregate costs of 4.4 and 1.9 points; summing each variant's lane differences, the two `passage_retrieval` lanes carry 55% of the shuffled variant's loss and 52% of the first-token variant's, the five worst lanes carry 90% and 72%, and 9 and 7 of the 29 lanes move by less than one point. Five lanes (`gov_report_e`, both `multi_news` lanes, `qasper_e` and `repobench-p`) are within one point under all three of the milder overrides. This is why the aggregate is small even though the mechanism is not, and it is consistent with the failure modes reported in Appendix B.9: what the selector buys is positional fidelity, and only some lanes ask for it. The mean-pool column is the clearest case of an aggregate hiding its parts, with losses above 11 points on both dialogue-summarisation lanes and gains above 8 on both classification lanes. The unweighted lane means reproduce the 29-lane totals of Table 2 exactly (36.74 for MemToken, then 36.32, 35.48, 35.44, 34.84 and 32.34 for the five variants), so the table can be read as a decomposition of that row. The last two rows renormalise against the 27 remaining lanes, whose floor and ceiling are 18.41 and 35.00 against 17.55 and 39.34 over all 29, which shortens the ladder from 0.202 to 0.136 of the full-text gain but leaves it monotone and leaves every ordering, and the mean-pool row, unchanged.

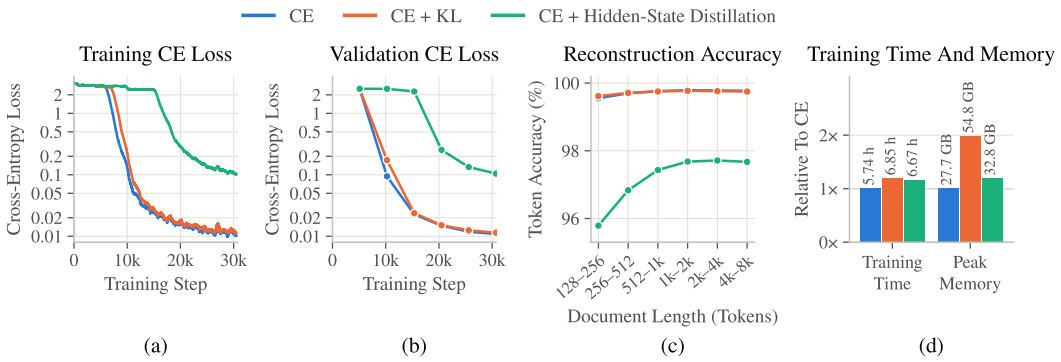


Figure 6: Stage-1 loss ablation on Qwen3.5-2B at $4\times$, with the same samples, order and budget in every arm. (a) Training CE, smoothed over 512 micro-steps, (b) validation CE, (c) final token accuracy by length bin, and (d) net training time and peak memory relative to CE.

The mean-pool pair at $8\times$ (Table 11) separates on one of its four axes and on no other: LooGLE token-F1 puts MemToken ahead by 0.013 ($p = 2 \times 10^{-4}$; 41.0% of items are tied and three quarters of those are items neither arm answers at all), while the LongBench official axis gives $p = 0.086$ with the two arms scoring identically on 3,771 of 7,418 items, the LongBench judge axis $p = 0.038$, and the LooGLE judge axis a dead heat at 250 discordant labels against 249. The one separating axis is also the one whose length bias we document in Appendix B.8, and the bias points the same way here, since the arm that loses on token-F1 is the longer one (median answer 16 words against 14). Length is not a sufficient explanation, because the full-text decoder writes longer answers still (21 words) and leads on both axes; but a difference that appears on one ruler out of four, and on the ruler carrying a known artefact aligned with it, is not one we are willing to call a direction, which is the outcome a two-ruler protocol exists to produce.

B.6 Across backbones

Table 12 gives MemToken on the three backbones with each normalised by its own no-context and full-text rows (Section 5.4); LongBench is the official 29-lane mean and LooGLE the judge accuracy. The cost of going from $4\times$ to $8\times$ on LooGLE is larger on the 2B decoder (0.34 of the full-text gain) than on the 9B decoder (0.19); Nemotron-3-Nano-4B is read under both axes in Appendix B.8.

B.7 Results on Nemotron-3-Nano-4B

Table 13 collects every result on Nemotron-3-Nano-4B, the one hybrid Mamba backbone. At $4\times$ MemToken recovers 0.62 and 0.59 of the full-text gain on the two LongBench axes and 0.62 to 0.63 on FRAMES, but only 0.13 on the LooGLE judge axis and none under token-F1, although the judge still places it above the floor (exact McNemar, 421 against 262 discordant items, $p = 10^{-9}$). We have no mechanism for the LooGLE collapse: the vectors enter every backbone through the same input path, and the collapse is present at both ratios. On FRAMES first+last is ahead at both ratios, by a margin that is not significant at $4\times$ (108 against 91 discordant items, $p = 0.26$) and is significant at $8\times$ (128 against 69, $p = 3 \times 10^{-5}$).

B.8 LooGLE under both scoring axes

Table 14 reports LooGLE on Qwen3.5-9B under the LLM judge and under token-F1. The two axes agree on the order of the compression methods (KVzip is the strongest baseline on both) but disagree on the ceiling: token-F1 places MemToken above the full-text decoder, the judge places the full-text decoder first by a wide margin. The disagreement is concentrated where the answers are longest; the full-text arm has the lowest item-level agreement between the two axes (0.47 at a 0.5 F1 threshold) and the no-context arm the highest (0.84). On the judge axis the long-dependency task clears the floor–ceiling gate (0.161 to 0.382) that it fails on token-F1 (0.151 to 0.175), and on that task KVzip is ahead of MemToken (0.262 vs. 0.233); MemToken's advantage is on the short-

Table 10: Selector ablations on Qwen3.5-2B at 4×, per LongBench lane. *Span* is that lane's full-text ceiling minus its no-context floor on the official metric, *ours* is MemToken's score, and the remaining columns are each variant's difference from MemToken on the same items; bold marks a difference of at least 1.5 points. Lanes are ordered by the *first* column, and those whose span is zero or negative (both RepoBench lanes) carry no usable R. The last two rows drop both `passage_retrieval` lanes and renormalise against the remaining 27 lanes. Appendix B.5 reads the table.

Lane	n	Span	ours	mean pool	P only	uniform	shuffled	first
passage_retrieval_en	200	92.0	92.5	**-4.0**	**-11.5**	**-10.5**	**-17.5**	**-39.0**
passage_retrieval_en_e	300	92.0	93.0	+0.7	**-7.3**	**-8.3**	**-12.7**	**-27.7**
trec	200	39.8	52.5	**+8.5**	**-4.0**	**-4.0**	**-8.0**	**-10.5**
2wikimqa	200	9.7	42.5	**-3.1**	**-3.4**	**-3.5**	**-4.0**	**-7.8**
trec_e	300	39.0	47.3	**+9.0**	**-2.7**	**-3.0**	**-5.3**	**-7.0**
gov_report	200	24.0	33.9	-0.5	-0.8	-1.0	**-2.2**	**-6.7**
hotpotqa_e	300	29.6	53.4	+0.9	**-1.8**	**-1.9**	**-3.2**	**-6.2**
hotpotqa	200	29.2	53.7	**-2.2**	**-1.7**	-0.9	**-2.4**	**-6.1**
gov_report_e	300	23.9	33.5	+0.2	-0.3	-0.7	-0.7	**-4.8**
qasper	200	19.3	19.5	**+2.0**	**-2.4**	-0.9	**-2.7**	**-4.0**
musique	200	22.1	32.6	**+2.5**	**-4.3**	**-4.3**	**-6.0**	**-3.5**
multi_news	200	15.3	16.1	**-2.5**	+0.8	+0.3	+0.2	**-2.3**
qmsum	200	4.1	11.1	**-2.1**	+0.7	+0.6	+1.1	**-2.1**
2wikimqa_e	300	9.0	39.4	+0.4	**+3.1**	**+2.5**	+0.5	**-2.1**
multi_news_e	294	14.8	13.3	**-1.5**	-0.2	-0.2	+0.2	**-2.0**
qasper_e	224	27.4	19.5	**+2.3**	-0.9	-0.9	+0.3	**-1.6**
narrativeqa	200	8.7	11.9	+0.7	+0.6	-0.4	**+1.6**	-1.0
repobench-p_e	300	-0.9	34.7	+0.3	+0.6	+0.9	+1.4	-0.9
multifieldqa_en	150	38.0	30.8	+0.0	-0.8	-0.9	**-1.9**	-0.9
multifieldqa_en_e	150	38.0	30.8	+0.0	-0.8	-0.9	**-2.0**	-0.9
repobench-p	500	-2.5	35.4	**+1.6**	+0.2	-0.0	+0.1	-0.5
passage_count_e	300	6.0	5.3	-0.3	-0.7	-0.7	-1.0	-0.3
triviaqa	200	5.8	83.2	**-2.1**	**-1.5**	-0.6	**+1.6**	+0.1
passage_count	200	4.5	3.0	**+2.0**	+1.0	0.0	+0.5	+0.5
triviaqa_e	300	8.5	84.3	**-2.6**	+1.1	+0.7	**+2.1**	+1.5
lcc_e	300	8.2	19.9	-0.4	+1.1	+0.7	+0.9	**+1.8**
lcc	500	12.0	21.2	+1.1	**+1.6**	**+1.7**	**+2.0**	**+2.1**
samsum_e	300	6.6	24.6	**-11.1**	+0.0	-0.0	**+1.5**	**+2.2**
samsum	200	7.9	26.6	**-11.7**	**-2.1**	-1.3	+0.4	**+2.4**
Mean over lanes		21.8	36.7	-0.42	-1.26	-1.30	-1.90	-4.40
R, all 29 lanes			0.881	0.861	0.823	0.821	0.793	0.679
R, 27 lanes			0.855	0.835	0.815	0.812	0.799	0.719

Table 11: The mean-pool ablation of Table 2 at 8× on Qwen3.5-2B: score and, in parentheses, R. Mean pool is tested against MemToken at 8× on the same items, with the marks of Table 2 (same family of 40 tests).

Variant	LongBench (official, 29 lanes)	LongBench-QA (judge)	LooGLE (token-F1)	LooGLE (judge)
MemToken (ours)	0.328 (0.70)	0.357 (0.70)	0.220 (0.48)	0.278 (0.34)
Mean pool	0.322 (0.67)	0.340 (0.65)†	0.207 (0.39)*	0.278 (0.34)

dependency task (0.779 vs. 0.597). The judge label is correlated with answer length on the arms that answer well: Pearson 0.32 for full text and 0.31 and 0.28 for MemToken at $4\times$ and $8\times$, against at most 0.12 on every other arm. The correlation is therefore not a property of the judge alone, and the validation of Appendix A.3 is the only length-free check of this axis. On Qwen3.5-2B the two axes agree on the order of every arm shown and on the ceiling: MemToken $4\times$ recovers 0.68 of the gain under the judge and 0.86 under token-F1, below full text on both, and the full-text decoder again writes the longest answers (median 21 words against 14). On Nemotron-3-Nano-4B token-F1 places MemToken $4\times$ at the no-context floor ($R = 0.00$; 0.1409 against 0.1410) while the judge

Table 12: MemToken across backbones, each normalised by its own floor and ceiling (first and last row of its block). LooGLE is judge accuracy, LongBench its official per-task metrics.

Backbone	Input	LongBench	LooGLE
Qwen3.5-2B	No context	0.175 (0.00)	0.114 (0.00)
	MemToken 4×	0.367 (0.88)	0.438 (0.68)
	MemToken 8×	0.328 (0.70)	0.278 (0.34)
	Full text	0.393 (1.00)	0.590 (1.00)
Nemotron-4B	No context	0.181 (0.00)	0.123 (0.00)
	MemToken 4×	0.301 (0.62)	0.175 (0.13)
	MemToken 8×	0.246 (0.33)	0.139 (0.04)
	Full text	0.376 (1.00)	0.519 (1.00)
Qwen3.5-9B	No context	0.206 (0.00)	0.172 (0.00)
	MemToken 4×	0.440 (0.80)	0.582 (0.77)
	MemToken 8×	0.401 (0.66)	0.482 (0.58)
	Full text	0.499 (1.00)	0.706 (1.00)

Table 13: All results on Nemotron-3-Nano-4B: score and, in parentheses, R against this backbone's own no-context and full-text scores. First+last was run on FRAMES only (–).

Input	LongBench (official)	LongBench-QA (judge)	LooGLE (judge)	LooGLE (token-F1)	FRAMES (judge)	FRAMES (F1)
No context	0.181 (0.00)	0.119 (0.00)	0.123 (0.00)	0.141 (0.00)	0.052 (0.00)	0.074 (0.00)
Full text	0.376 (1.00)	0.465 (1.00)	0.519 (1.00)	0.318 (1.00)	0.385 (1.00)	0.329 (1.00)
MemToken 4×	0.301 (0.62)	0.322 (0.59)	0.175 (0.13)	0.141 (0.00)	0.263 (0.63)	0.233 (0.62)
MemToken 8×	0.246 (0.33)	0.306 (0.54)	0.139 (0.04)	0.136 (−0.03)	0.167 (0.35)	0.163 (0.35)
First+last 4×	–	–	–	–	0.284 (0.70)	0.264 (0.74)
First+last 8×	–	–	–	–	0.240 (0.56)	0.215 (0.55)

finds it above the floor ($R = 0.13$; McNemar 421 vs. 262, $p = 10^{-9}$), so the lexical axis reports a resolution limit as zero signal (Section 5.4).

B.9 Where compression fails

Three failure modes recur across benchmarks and metrics, and we state them as limitations of the method rather than of the evaluation.

Single-document extractive question answering. On the QASPER answer task (1,005 questions over one paper each, Qwen3.5-9B, judge axis) MemToken $4\times$ recovers 33% of the full-text gain against 64% for LLMLingua-2 and 44% for first+last, both differences significant (exact McNemar $p \leq 3 \times 10^{-3}$), and its answers have a median length of two words against four for both baselines and eight for full text. On the three single-document LongBench lanes the deficit is smaller, appears only under the judge, and reverses under the official F1, where MemToken is ahead of both baselines on all three. The compressed representation supports finding an answer but not reproducing an exact span, and when the task asks for the span the decoder gives up early.

Verbatim continuation. On the two LongBench LCC lanes, where the task is to continue a file, MemToken $4\times$ recovers 0.15 and 0.10 of the full-text gain on the 9B decoder, against 0.83 on both lanes for KVzip and 0.85 and 0.81 for first+last, the two baselines that keep tokens verbatim (the RepoBench lanes fail the per-lane gate, Appendix A.2). MemToken's result on CrossCodeEval (Table 1) is not a counterexample: there the cross-file context is compressed while the current file sits uncompressed in the query slot, and completion draws on both.

Summaries. On ZeroSCROLLS MemToken $4\times$ is behind first+last and LLMLingua-2 on the seven-lane official mean (R 0.60 against 0.67 and 0.65, Table 7), and the deficit sits entirely on the four ROUGE-scored summary lanes: on the question-answering lanes (812 items) the judge finds it indistinguishable from both baselines on the 2B decoder (exact McNemar $p = 0.46$ and 0.31) and on the 9B decoder ($p = 0.31$ and 0.66). On LongBench's GovReport lanes the compressed decoder

Table 14: LooGLE under the two scoring axes on the three backbones. Judge accuracy is the three-vote majority, split by task for 9B in the long- and short-dependency columns; token-F1 is the lexical metric; R is computed per axis and per backbone. MemToken 4× clears the full-text ceiling on the F1 axis on 9B ($R > 1$) because the full-text decoder writes longer answers (median words, last column), which token-F1 penalises; the judge does not. †Ratio on KV-cache bytes.

Input	Judge acc.	by task		Token-F1	Unanimous votes	Words (median)
		long	short			
Qwen3.5-9B						
No context	0.172 (0.00)	0.161	0.178	0.132 (0.00)	0.944	27
Full text	0.706 (1.00)	0.382	0.889	0.281 (1.00)	0.925	32
First+last 4×	0.256 (0.16)	0.223	0.274	0.146 (0.10)	0.951	38
LLMLingua-2 4×	0.352 (0.34)	0.242	0.414	0.158 (0.18)	0.897	32
KVzip† 4×	0.476 (0.57)	0.262	0.597	0.227 (0.64)	0.920	32
MemToken 4×	0.582 (0.77)	0.233	0.779	0.305 (1.16)	0.944	18
MemToken 8×	0.482 (0.58)	0.191	0.646	0.278 (0.98)	0.939	17
Qwen3.5-2B						
No context	0.114 (0.00)	–	–	0.150 (0.00)	0.959	15
Full text	0.590 (1.00)	–	–	0.295 (1.00)	0.931	21
First+last 4×	0.246 (0.28)	–	–	0.182 (0.22)	0.952	18
LLMLingua-2 4×	0.238 (0.26)	–	–	0.174 (0.17)	0.922	17
KVzip† 4×	0.182 (0.14)	–	–	0.169 (0.13)	0.941	19
MemToken 4×	0.438 (0.68)	–	–	0.275 (0.86)	0.958	14
MemToken 8×	0.278 (0.34)	–	–	0.220 (0.48)	0.944	14
Nemotron-3-Nano-4B						
No context	0.123 (0.00)	–	–	0.141 (0.00)	0.952	11
Full text	0.519 (1.00)	–	–	0.318 (1.00)	0.931	13
MemToken 4×	0.175 (0.13)	–	–	0.141 (0.00)	0.961	19
MemToken 8×	0.139 (0.04)	–	–	0.136 (−0.03)	0.946	19

runs to the generation cap and earns no more ROUGE-L for the extra length, so what fails on summaries is the calibration of open-ended generation, not the content the vectors carry.

B.10 Efficiency on Qwen3.5-9B

All measurements here use Qwen3.5-9B on A800-SXM4-80GB GPUs with one process per GPU, in bf16 with greedy decoding (PyTorch 2.9.1 with CUDA 12.8, Transformers 5.13.0, FlashAttention 2.8.3). The length sweep takes the same 64 MuSiQue development questions and pads each with the passages of other questions to 4K, 8K, 16K, 32K and 64K tokens, so the tiers differ only in length. Every item runs at batch size 1, and the first item of every shard is dropped as warm-up because it runs 17% slower than the rest, which leaves 56 items at 64K. A query is split into four stages: preprocessing by the scorer of a token-dropping baseline, context processing (the MemToken compressor, the ARC-Encoder, or KVzip's prefill and KV scoring), decoder prefill, and decoding. Prefill and decoding are separated at the first generated token. All 14 arms of Table 1 are measured, and first+last, which Figure 5 leaves out, is kept in the tables.

FLOPs are counted analytically. The shape of every matrix multiplication is read from the checkpoint's tensor headers, and the quadratic attention term of the 8 full-attention layers out of 32 and the state update of the Gated-DeltaNet layers are added. Three checks agree to 0.00%: the matrix parameters counted from the headers equal those derived from the model configuration (6.9185 billion), and PyTorch's FlopCounterMode measures the same matrix-multiplication FLOPs as the formula for a full-text prefill of 3,858 tokens (53.383 TFLOPs) and for the compressor on the same input (41.633 TFLOPs). The gate follows the 22nd of the compressor's 32 layers, so its first 22 layers process the full length and its last 10 the compressed length. The scorers of the token-dropping baselines are counted at the original length, with LLMLingua-2's XLM-R token count approximated by the Qwen token count, and ARC-Encoder is counted on its own tokenizer's length.

Tables 15 and 16 give every arm at every length. The decoder-side share of MemToken falls slightly with length, from 26% of the full-text FLOPs at 4K to 21% at 64K at 4×, because the full-text

Table 15: Qwen3.5-9B length sweep, 4k, 8k, 16k tokens: one query per context, batch 1, A800-SXM4-80GB, bf16. Slot is the number of positions the context occupies in the decoder input. Latency is split into preprocessing (the scorer of a token-dropping baseline), context processing (the MemToken compressor, the ARC-Encoder, or KVzip's prefill and KV scoring), decoder prefill up to the first generated token, and decoding, each a median over items, so the stages need not sum to the median total. TFLOPs are per-item means of the analytic count (Appendix B.10), with the decoder column covering prefill and decoding. Peak is the GPU memory high-water mark, state the memory that must be kept per context (KV cache or memory vectors), and F1 the SQuAD F1 of the answers. †Ratio on KV-cache bytes. ‡Trained baseline.

		Latency (ms)					TFLOPs						
Method	Slot	Prep.	Context	Prefill	Decode	Total	Prep.	Context	Decoder	Total	Peak (GB)	State (MB)	F1
4k tokens, 64 items													
Full text	3,808	28	0	304	163	498	0.0	0.0	55.4	55.4	18.2	151.4	0.583
No context	0	1	0	68	122	190	0.0	0.0	0.8	0.8	17.6	0.0	0.108
MemToken 4×	952	27	269	96	163	554	0.0	42.9	14.3	57.2	33.2	7.9	0.424
MemToken 8×	476	27	263	75	164	528	0.0	40.8	7.5	48.3	33.2	3.9	0.414
First+last 4×	955	17	0	96	131	249	0.0	0.0	14.3	14.3	17.7	56.8	0.178
First+last 8×	480	14	0	75	129	239	0.0	0.0	7.6	7.6	17.7	41.1	0.121
LLMLingua-2 4×	950	345	0	97	160	585	2.4	0.0	14.2	16.6	20.1	56.6	0.326
LLMLingua-2 8×	472	326	0	75	129	525	2.4	0.0	7.4	9.8	19.9	40.7	0.106
Selective-Context 4×	953	623	0	98	121	857	62.3	0.0	14.1	76.4	21.4	56.3	0.083
Selective-Context 8×	478	615	0	76	85	790	62.3	0.0	7.5	69.9	21.4	41.0	0.046
KVzip† 4×	946	30	1,470	72	133	1,725	0.0	110.9	0.8	111.8	18.7	56.6	0.273
KVzip† 8×	471	29	1,470	71	129	1,694	0.0	110.9	0.8	111.8	18.7	40.8	0.089
ARC-Encoder‡ 4×	902	30	157	90	134	422	0.0	19.5	13.5	33.0	23.5	7.4	0.262
ARC-Encoder‡ 8×	453	29	156	68	126	379	0.0	19.5	7.2	26.7	23.5	3.7	0.235
8k tokens, 64 items													
Full text	7,872	56	0	611	163	828	0.0	0.0	114.5	114.5	19.0	283.6	0.476
No context	0	1	0	68	122	190	0.0	0.0	0.8	0.8	17.6	0.0	0.108
MemToken 4×	1,968	56	541	167	131	899	0.0	89.2	28.5	117.7	33.8	16.2	0.421
MemToken 8×	984	56	513	98	148	816	0.0	84.8	14.6	99.4	33.8	8.1	0.382
First+last 4×	1,972	34	0	169	163	369	0.0	0.0	28.5	28.5	17.9	89.9	0.184
First+last 8×	988	27	0	98	129	253	0.0	0.0	14.6	14.6	17.7	57.6	0.168
LLMLingua-2 4×	1,960	694	0	169	160	1,026	5.0	0.0	28.4	33.3	20.2	89.5	0.270
LLMLingua-2 8×	979	661	0	98	163	868	5.0	0.0	14.4	19.3	20.1	57.0	0.135
Selective-Context 4×	1,966	1,312	0	171	108	1,592	129.7	0.0	27.8	157.5	21.4	88.3	0.131
Selective-Context 8×	984	1,296	0	99	85	1,501	129.7	0.0	14.1	143.8	21.4	56.5	0.098
KVzip† 4×	1,960	58	2,277	72	150	2,561	0.0	235.4	0.8	236.3	18.9	89.5	0.331
KVzip† 8×	975	57	2,286	71	129	2,525	0.0	235.4	0.8	236.3	18.9	57.2	0.112
ARC-Encoder‡ 4×	1,855	60	312	159	132	670	0.0	40.0	26.9	66.8	23.8	15.2	0.236
ARC-Encoder‡ 8×	932	60	303	91	126	577	0.0	40.0	13.8	53.8	23.8	7.6	0.237
16k tokens, 64 items													
Full text	16,090	112	0	1,249	164	1,516	0.0	0.0	241.7	241.7	20.4	553.0	0.425
No context	0	1	0	68	123	191	0.0	0.0	0.8	0.8	17.6	0.0	0.108
MemToken 4×	4,023	113	1,099	313	134	1,665	0.0	187.8	57.8	245.6	34.9	33.0	0.449
MemToken 8×	2,012	112	1,045	175	149	1,481	0.0	178.5	29.1	207.6	34.9	16.5	0.410
First+last 4×	4,026	68	0	315	165	547	0.0	0.0	57.9	57.9	18.3	157.2	0.148
First+last 8×	2,015	53	0	176	144	384	0.0	0.0	29.1	29.1	17.9	91.3	0.086
LLMLingua-2 4×	4,012	1,390	0	314	132	1,836	10.1	0.0	57.5	67.7	20.6	156.4	0.251
LLMLingua-2 8×	2,004	1,329	0	175	126	1,633	10.1	0.0	28.5	38.7	20.5	89.9	0.141
Selective-Context 4×	4,018	2,805	0	317	122	3,270	273.5	0.0	53.1	326.5	21.5	146.2	0.120
Selective-Context 8×	2,011	2,776	0	179	85	3,064	273.5	0.0	28.0	301.5	21.5	88.7	0.085
KVzip† 4×	4,002	117	4,019	73	134	4,366	0.0	515.4	0.9	516.3	20.2	156.5	0.304
KVzip† 8×	1,991	115	4,039	72	129	4,336	0.0	515.4	0.8	516.3	20.2	90.5	0.131
ARC-Encoder‡ 4×	3,785	122	587	297	140	1,154	0.0	81.5	54.4	135.9	24.2	31.0	0.201
ARC-Encoder‡ 8×	1,900	120	562	165	128	982	0.0	81.5	27.5	108.9	24.2	15.6	0.226

Table 16: Qwen3.5-9B length sweep, 32k, 64k tokens: one query per context, batch 1, A800-SXM4-80GB, bf16. Slot is the number of positions the context occupies in the decoder input. Latency is split into preprocessing (the scorer of a token-dropping baseline), context processing (the MemToken compressor, the ARC-Encoder, or KVzip's prefill and KV scoring), decoder prefill up to the first generated token, and decoding, each a median over items, so the stages need not sum to the median total. TFLOPs are per-item means of the analytic count (Appendix B.10), with the decoder column covering prefill and decoding. Peak is the GPU memory high-water mark, state the memory that must be kept per context (KV cache or memory vectors), and F1 the SQuAD F1 of the answers. The 64k tier has 56 items. †Ratio on KV-cache bytes. ‡Trained baseline.

		Latency (ms)					TFLOPs						
Method	Slot	Prep.	Context	Prefill	Decode	Total	Prep.	Context	Decoder	Total	Peak (GB)	State (MB)	F1
32k tokens, 64 items													
Full text	32,516	231	0	2,675	168	3,063	0.0	0.0	522.4	522.4	23.3	1,091.4	0.445
No context	0	1	0	68	122	191	0.0	0.0	0.8	0.8	17.6	0.0	0.108
MemToken 4×	8,130	234	2,303	622	139	3,297	0.0	402.1	118.2	520.3	37.1	66.6	0.443
MemToken 8×	4,065	235	2,191	328	163	2,904	0.0	382.8	58.4	441.2	37.1	33.3	0.430
First+last 4×	8,132	137	0	622	131	889	0.0	0.0	118.3	118.3	19.0	291.8	0.213
First+last 8×	4,068	107	0	330	127	563	0.0	0.0	58.5	58.5	18.3	158.6	0.121
LLMLingua-2 4×	8,123	2,784	0	619	163	3,555	20.5	0.0	117.7	138.2	21.3	290.7	0.228
LLMLingua-2 8×	4,062	2,667	0	327	127	3,097	20.5	0.0	57.8	78.3	20.8	157.0	0.170
Selective-Context 4×	8,128	6,364	0	627	126	7,083	587.5	0.0	112.1	699.5	25.2	278.0	0.109
Selective-Context 8×	4,066	6,297	0	333	87	6,699	587.5	0.0	56.3	643.8	25.2	153.5	0.088
KVzip† 4×	8,097	237	8,002	75	133	8,419	0.0	1,181.4	0.9	1,182.3	22.8	290.6	0.236
KVzip† 8×	4,026	236	8,004	75	132	8,427	0.0	1,181.4	0.8	1,182.2	22.8	157.1	0.079
ARC-Encoder‡ 4×	7,618	251	1,179	577	133	2,142	0.0	164.1	110.5	274.7	25.3	62.4	0.238
ARC-Encoder‡ 8×	3,824	248	1,126	300	128	1,808	0.0	164.0	54.9	219.0	25.3	31.3	0.165
64k tokens, 56 items													
Full text	63,400	474	1	5,856	178	6,519	0.0	0.0	1,145.4	1,145.4	28.8	2,102.7	0.447
No context	0	1	0	68	122	190	0.0	0.0	0.8	0.8	17.6	0.0	0.123
MemToken 4×	15,850	507	4,893	1,227	132	6,778	0.0	866.6	237.6	1,104.1	41.2	129.9	0.326
MemToken 8×	7,926	495	4,669	610	163	5,932	0.0	826.8	115.1	941.9	41.2	64.9	0.390
First+last 4×	15,853	269	0	1,223	165	1,645	0.0	0.0	237.6	237.6	20.4	544.7	0.238
First+last 8×	7,928	208	0	610	130	955	0.0	0.0	115.1	115.1	19.0	285.0	0.146
LLMLingua-2 4×	15,852	5,443	0	1,222	146	6,817	39.9	0.0	236.9	276.8	22.7	543.3	0.296
LLMLingua-2 8×	7,928	5,212	0	609	163	5,928	39.9	0.0	114.7	154.6	21.2	284.0	0.141
Selective-Context 4×	15,846	14,439	0	1,233	110	15,773	1,273.0	0.0	229.2	1,502.3	32.4	527.0	0.144
Selective-Context 8×	7,925	14,333	0	615	107	15,005	1,273.0	0.0	105.3	1,378.3	32.4	263.1	0.109
KVzip† 4×	15,805	473	17,158	80	140	17,856	0.0	2,815.0	0.9	2,815.9	27.8	542.6	0.251
KVzip† 8×	7,844	474	17,185	80	140	17,885	0.0	2,815.0	0.9	2,815.9	27.8	282.3	0.100
ARC-Encoder‡ 4×	14,822	501	2,288	1,141	134	4,075	0.0	319.1	221.1	540.2	27.3	121.4	0.112
ARC-Encoder‡ 8×	7,440	506	2,193	565	129	3,400	0.0	318.9	107.8	426.7	27.3	60.9	0.133

Table 17: One context asked Q times on Qwen3.5-9B (medians over 8 contexts per length, in ms). Once is paid per context before the first query: the prefill for full text, the prefill and KV scoring for KVzip, and the compressor pass for MemToken, followed by the prefill of the memory slot when its KV is cached. Per query is the cost of each later query. *Vectors only* keeps the memory vectors and re-prefills them at every query instead of caching the decoder's KV. State is what must be kept per context. KVzip's cache keeps the full size because the kvpress implementation masks evicted entries rather than freeing them. †Ratio on KV-cache bytes.

Method	Once	Per query	$Q = 1$	$Q = 4$	$Q = 16$	State (MB)
4k tokens						
Full text, KV cache	313	225	529	1,194	3,896	150.9
MemToken 4×, KV cache	353	246	604	1,339	4,294	57.8
MemToken 8×, KV cache	319	262	582	1,369	4,523	42.3
MemToken 4×, vectors only	267	281	555	1,393	4,760	7.8
MemToken 8×, vectors only	257	276	534	1,364	4,675	3.9
KVzip† 4×, KV cache	1,473	195	1,677	2,263	4,613	150.9
KVzip† 8×, KV cache	1,428	256	1,688	2,462	5,516	150.9
8k tokens						
Full text, KV cache	598	227	843	1,534	4,264	285.1
MemToken 4×, KV cache	698	209	913	1,540	4,045	91.3
MemToken 8×, KV cache	592	265	856	1,655	4,842	59.0
MemToken 4×, vectors only	539	314	855	1,798	5,558	16.2
MemToken 8×, vectors only	505	301	808	1,712	5,338	8.1
KVzip† 4×, KV cache	2,275	284	2,549	3,403	6,816	285.1
KVzip† 8×, KV cache	2,235	257	2,470	3,245	6,333	285.1
16k tokens						
Full text, KV cache	1,224	232	1,454	2,149	4,946	552.8
MemToken 4×, KV cache	1,403	231	1,637	2,332	5,109	158.3
MemToken 8×, KV cache	1,186	254	1,441	2,204	5,258	92.5
MemToken 4×, vectors only	1,098	484	1,586	3,041	8,845	32.9
MemToken 8×, vectors only	1,027	365	1,397	2,494	6,879	16.4
KVzip† 4×, KV cache	4,060	244	4,302	5,043	7,976	552.8
KVzip† 8×, KV cache	3,999	225	4,227	4,908	7,608	552.8
32k tokens						
Full text, KV cache	2,625	273	2,909	3,727	7,012	1,092.7
MemToken 4×, KV cache	2,894	211	3,108	3,742	6,277	293.3
MemToken 8×, KV cache	2,470	254	2,724	3,487	6,538	160.0
MemToken 4×, vectors only	2,287	769	3,056	5,362	14,589	66.6
MemToken 8×, vectors only	2,166	518	2,685	4,242	10,451	33.3
KVzip† 4×, KV cache	7,886	230	8,111	8,801	11,554	1,092.7
KVzip† 8×, KV cache	7,898	232	8,146	8,839	11,629	1,092.7
64k tokens						
Full text, KV cache	5,806	215	6,029	6,672	9,256	2,105.3
MemToken 4×, KV cache	6,085	233	6,313	7,018	9,814	546.4
MemToken 8×, KV cache	5,231	235	5,458	6,166	8,988	286.6
MemToken 4×, vectors only	4,883	1,396	6,277	10,463	27,222	129.9
MemToken 8×, vectors only	4,640	780	5,421	7,763	17,133	65.0
KVzip† 4×, KV cache	17,027	246	17,254	17,996	20,951	2,105.3
KVzip† 8×, KV cache	17,032	271	17,284	18,097	21,353	2,105.3

prefill grows faster than linearly with length. KVzip's context-side cost grows with the full length, since it prefills the uncompressed context before it scores and evicts, and its decoder then reads the kept entries without a new prefill. Selective-Context's scorer is the 9B model itself, so its preprocessing alone exceeds the full-text prefill at every length. LLMLingua-2's classifier is small, but its preprocessing time grows from 0.35 s at 4K to 5.4 s at 64K.

Table 17 asks one context Q times. With the decoder's KV cached on both sides, the cost per later query is the same for full text and MemToken at every length, so the difference lies in what must be stored: at 64K the cached state is 546 or 287 MB against 2,105 MB for full text. Storing only the memory vectors reduces the state to 130 or 65 MB, and every query then re-prefills the memory slot. KVzip pays 17.0 s once at 64K, 2.9× the full-text prefill, and keeps a cache of full size because the kvpress implementation masks the evicted entries instead of freeing them.

With a long output, full-text decoding slows from 42.0 to 44.9 ms per token between 4K and 64K, while MemToken $4\times$ stays between 41.9 and 42.2 ms, about 6% faster at 64K (Table 18).

Table 18: Decoding cost with a long output on Qwen3.5-9B: ms per generated token when 256 tokens are forced, medians over 8 contexts per length.

Method	4k	8k	16k	32k	64k
Full text	42.0	42.0	42.0	43.0	44.9
MemToken 4×	41.9	41.9	42.0	42.1	42.2
MemToken 8×	42.0	42.0	42.0	40.6	41.2

Table 19: Batched throughput on Qwen3.5-9B for equal-length contexts (prefill plus 16 generated tokens): queries per second, with peak GPU memory in GB in parentheses, at every batch size up to the first out-of-memory batch. *Answer only* starts from precomputed memory vectors, and *with compression* includes the compressor pass. The last column divides the best throughput by that of full text at the same length. Every method reaches the same largest batch, because the MemToken process keeps a second copy of the 9B weights resident for the compressor.

Method	$b = 1$	$b = 2$	$b = 4$	$b = 8$	$b = 16$	First OOM	vs. full text
16k tokens							
Full text	0.54 (35)	0.65 (38)	0.73 (43)	0.78 (54)	0.80 (75)	32	1.0×
MemToken 4×, answer only	1.07 (36)	1.63 (37)	2.21 (41)	2.68 (50)	2.99 (68)	32	3.7×
MemToken 8×, answer only	1.25 (35)	2.14 (37)	3.26 (41)	4.45 (50)	5.36 (67)	32	6.7×
MemToken 4×, with compression	0.49 (35)	0.59 (37)	0.66 (42)	0.69 (50)	0.71 (68)	32	0.9×
MemToken 8×, with compression	0.55 (35)	0.67 (37)	0.76 (41)	0.81 (50)	0.84 (68)	32	1.0×
32k tokens							
Full text	0.30 (38)	0.34 (44)	0.36 (54)	0.37 (75)	–	16	1.0×
MemToken 4×, answer only	0.80 (38)	1.08 (42)	1.32 (50)	1.47 (68)	–	16	4.0×
MemToken 8×, answer only	1.03 (38)	1.60 (42)	2.18 (50)	2.65 (68)	–	16	7.2×
MemToken 4×, with compression	0.28 (38)	0.31 (42)	0.33 (51)	0.34 (69)	–	16	0.9×
MemToken 8×, with compression	0.32 (38)	0.36 (42)	0.38 (50)	0.39 (68)	–	16	1.1×
64k tokens							
Full text	0.15 (43)	0.16 (54)	0.17 (74)	–	–	8	1.0×
MemToken 4×, answer only	0.54 (42)	0.65 (50)	0.73 (67)	–	–	8	4.4×
MemToken 8×, answer only	0.80 (42)	1.10 (50)	1.35 (67)	–	–	8	8.1×
MemToken 4×, with compression	0.15 (42)	0.15 (50)	0.16 (68)	–	–	8	0.9×
MemToken 8×, with compression	0.17 (42)	0.18 (50)	0.19 (67)	–	–	8	1.1×

Table 19 gives the throughput at every batch size. All methods reach the same largest batch, because the process that runs MemToken keeps the compressor's 9B weights resident, 15 GB more than full text before any input is processed. The compressor adds 173,113,344 LoRA parameters and a gate of 33,570,816 parameters to the backbone, so sharing the backbone with the decoder would remove most of this overhead, which we have not measured. Energy was not measured for every arm and is not reported.

B.11 Earlier inference-cost measurement on Qwen3.5-2B

An earlier measurement on Qwen3.5-2B with 192 LongBench v2 items, reported in Table 20, is kept here for reference. At $4\times$ the prefill shrinks from 36,712 to 9,286 tokens, the KV cache from 451 to 114 MB (3.9×), and decoding from 118 to 54 ms per token (2.2×). At $8\times$ the cache is 58 MB and decoding 43 ms per token. The costs are a one-off compressor pass of 763 ms per document at $4\times$ (LLMLingua-2's tokenisation and classification take 3,259 ms) and a peak memory $1.4\times$ that of the full-text run while the compressor is resident. This measurement reports no end-to-end latency, because the arms generate different numbers of tokens (medians 6 to 12) and that, not the compression, dominates the wall-clock ratio. Figure 7 instead plots context-side time against the number of queries per context. KVzip and MemToken, the two strongest methods in Table 1, make different trades: KVzip keeps the decoder's own cache, is best on verbatim tasks and needs no training, but must prefill the whole document to build the cache and at a matched budget is 0.20 of the full-text gain behind MemToken on LooGLE and 0.21 on the LongBench question-answering lanes (9B, judge axis). MemToken pays a one-off compressor pass of under a second and thereafter processes a quarter or an eighth of the tokens at every query, and its vectors can be stored and reused. Table 20 gives the per-item medians behind Figure 7. The one-off column contains tokenisation, 160 to 290 ms for every arm that reads a document, plus MemToken's compressor pass (763 ms at $4\times$, 728 ms at $8\times$) or LLMLingua-2's token classifier, which puts MemToken at 1,044 and 1,012 ms

Table 20: Inference cost on Qwen3.5-2B (192 LongBench v2 items, one shard per GPU, medians). One-off is the query-independent work before the first token: tokenisation plus MemToken's compressor pass or LLMLingua-2's classifier. KV is median prefill tokens × 12,288 B (6 full-attention layers of 24). Decode is the median latency per output token; peak includes the resident compressor.

Setting	Prefill	Ratio	KV (MB)	One-off (ms)	Decode (ms/token)	Peak (MB)
Full text	36,712	1.00×	451	287	118.0	7,562
MemToken 4×	9,286	3.95×	114	1,044	53.8	10,593
MemToken 8×	4,758	7.72×	58	1,012	43.1	10,593
First+last 4×	9,290	3.95×	114	160	62.1	5,081
LLMLingua-2 4×	9,286	3.95×	114	3,259	53.6	7,607
No context	43	–	0.5	1	35.7	4,244

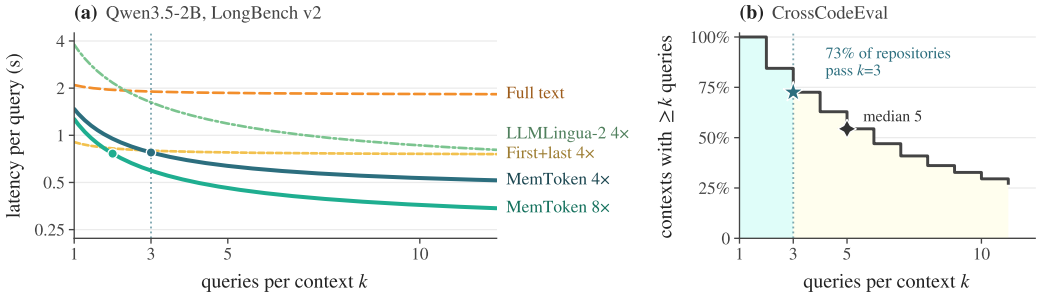


Figure 7: Amortising the context-side cost over queries on Qwen3.5-2B: (a) Context-side latency per query when k queries share one context (Qwen3.5-2B, medians of Table 20, log scale); MemToken is cheaper than full text from the first query and than truncation from the third (dots). (b) Fraction of CrossCodeEval repository contexts that serve at least k completions: 73% reach the third (median 5, mean 9.9).

against 3,259 ms for LLMLingua-2 (3.1× and 3.2× less). The measured ratios are 3.95× and 7.72× on prefill tokens, and the peak memory of 10,593 MB against 7,562 MB for the full-text run is the resident compressor.