End-to-End Context Compression at Scale

Anonymous Author(s)
Affiliation
Address
email

Abstract

Long-context language model inference is bottlenecked by memory, as the KV cache grows with context length. Recent techniques to compress the KV cache fall short: they either degrade model quality substantially, require considerable time, or compute to compress a single long prompt. Furthermore, many methods require the input to fit within the target model's context window, and are generally incompatible with modern production inference engines. Encoder-decoder compressors, which map a long token sequence to a shorter sequence of latent embeddings consumed by a decoder, are an appealing alternative in principle. However, existing approaches are not competitive with KV cache compression on the accuracy-efficiency frontier. In this work, we revisit encoder-decoder compression and close this gap. We first perform an architecture search, pretraining many variants from scratch to determine how best to design and train encoder-decoder compressors. Guided by our findings, we continually pretrain 0.6B-encoder 4B-decoder models on over 350B tokens at compression ratios of 1:4, 1:8, and 1:16. We introduce *Latent Context Language Models* (LCLMs), a family of compressors that improve the Pareto frontier between general-task performance, compression speed, and peak memory usage. We demonstrate LCLMs offer a practical path to long-context agents on complex real-world tasks as efficient backbones for agentic RAG systems.

1 Introduction

Reasoning over long-contexts is a crucial capability for state-of-the-art Large Language Models (LLMs), as it enables them to parse long documents, engage in multi-turn conversations, and perform long-horizon agentic tasks. However, in production systems, the input context, working horizon, and memory can grow to millions of tokens, making inference increasingly constrained by memory and latency due to the growth of the KV cache [Hooper et al., 2024]. Even when inputs fit within the model's maximum context window, models often struggle to reliably use information distributed across long-contexts [Liu et al., 2024a, An et al., 2025]. As a result, context and memory management has emerged as a critical systems and modeling challenge for deploying long-horizon LLM systems.

A natural response is to reduce the size of the KV cache directly. KV cache compression methods reduce the memory footprint of cached activations by evicting cache entries [Li et al., 2024c, Devoto et al., 2025, Kim et al., 2025, Zweiger et al., 2026] or training a compact cache offline [Eyuboglu et al., 2025]. However, they have important limitations for general long-context inference. Many methods still require the full context to be prefilled before compression, and some require substantial additional time or memory beyond the original prefill. Query-dependent methods produce caches that are difficult to reuse across turns [Li et al., 2024b], while methods that evict caches non-uniformly across attention heads and layers [Kim et al., 2025] are difficult to integrate with modern inference engines that assume a shared sequence length across the cache.

Preprint.

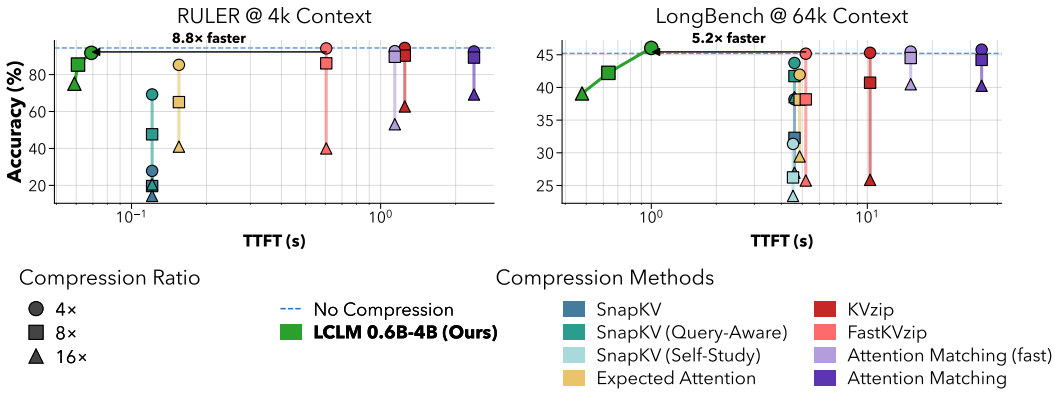

Figure 1: **Our Latent Context Language Models achieve high quality compression while being fast and memory efficient.** We show RULER [Hsieh et al., 2024] (left) and LongBench [Bai et al., 2024] (right) accuracy vs. TTFT (time to first token) per sample on a single H200. All methods are based on the same decoder `Qwen3-4B-Instruct- 2507` [Yang et al., 2025]. We see that our models lie on a new Pareto frontier, compressing contexts much faster while maintaining high accuracy, especially for high compression ratios. KV cache compression baselines appear as vertical lines as the context is prefilled the same regardless of compression ratio and the eviction operation is much faster than the prefill time.

Encoder-decoder soft-token methods [e.g. Lin et al., 2025, Ge et al., 2023, Chevalier et al., 2023, Liao et al., 2025, Yen et al., 2024] offer a promising way to address the limitations of KV cache compression. Rather than manipulating the KV cache directly, these models encode the raw input tokens into a much shorter sequence of continuous embeddings, which are then provided to the LLM in place of the original context. In principle, this approach is highly attractive: compression is parallelizable, supported by standard LLM inference engines, and works with standard LLM decoding while extending the decoder far beyond its native context length.

However, existing soft-token compression methods typically face a trade-off: they either substantially degrade the capabilities of the base language model or depend on domain-specific training to remain effective. This motivates a fundamental question: **can we train a general, task-agnostic compressor that preserves a model's original in-context capabilities?** To this end, we introduce *Latent Context Language Models* (LCLMs), a family of encoder-decoder compressors trained end-to-end at scale. We initialize both the encoder and decoder with pretrained models and jointly train them on up to a total of 350B tokens for the combined encoder-decoder. Crucially, we demonstrate that LCLMs can preserve decoder performance while reducing memory used for input context.

Our contributions are summarized as follows:

1. **Training recipe:** We develop a fine-grained training recipe for encoder-decoder context compression. To this end, we curate continual pretraining and finetuning data, and optimize training details across multiple training stages. See Section 4.

2. **Comprehensive architectural search:** We conduct a large-scale architecture search to identify optimal design choices for encoder-decoder compression models in a controlled setting. See Section 5 and Figure 2.

3. **New memory-performance frontier:** We train a family of models across different compression ratios at scale. Our experiments identify a Pareto frontier that effectively balances memory, latency, and downstream accuracy. We will open-source this suite of models as well as the code and data. See Section 6 and Figures 1, 3, and 4.

4. **Downstream Applications:** We create an agentic system with a high compression ratio, where the agent can select which compressed chunk to expand with tool calls. This agentic harness enhances the model's performance on challenging needle in the haystack tasks. See Section 7 and Figure 5.

2 Related Work

Context compression approaches generally fall into three categories: hard-token, soft-token, and KV cache compression. We defer discussion of hard-token approaches to Appendix B, since they generally underperform the latter two.

KV cache compression. KV cache compression methods evict entries from the KV cache. Most methods rely on hand-crafted or learned heuristics to select which entries to drop.

Prompt-agnostic methods [Kim et al., 2025, Devoto et al., 2025, Zweiger et al., 2026] prune the context without knowledge of the query, whereas prompt-dependent methods such as SnapKV [Li et al., 2024c] require explicit context-prompt pairs and yield query-specific caches that limit multi-turn applicability [Li et al., 2024b, Kim et al., 2025]. An alternative is to anticipate future queries at compression time: Cartridges [Eyuboglu et al., 2025] distills a fixed-size KV cache per corpus. However, this comes at a substantial training cost, with an ICL-quality cache requiring approximately 30 minutes on an $8\times$H100 node for an $8B$ model.

Although KV cache compression methods are well-studied, they are not widely adopted in inference engines such as vLLM [Kwon et al., 2023] or SGLang [Zheng et al., 2024]. Compression methods such as KVzip allocate eviction budgets non-uniformly across attention heads and layers; therefore, they cannot reduce the sequence-length dimension of the KV cache. Instead, KV cache compression methods mask evicted positions in the attention computation, forfeiting the memory and throughput benefits of these paged-attention engines. Some methods further require prefill passes that are two to three times longer than the original input [Kim et al., 2025]. We note that soft token compression methods are fully compatible with these open source inference frameworks.

Soft-token compression. Despite promising results, most prior soft-token methods rely on offline preprocessing and do not convincingly preserve the base model's broad in-context behavior. Prior work is mostly evaluated on domain- or task-specific finetuning that is not transferable across tasks [Tang et al., 2025, Feldman and Artzi, 2025, Dai et al., 2025, Lin et al., 2025, Cheng et al., 2024]. To the best of our knowledge, these methods are not evaluated on long-context benchmarks with information dense tasks that require fine-grained details throughout the whole context, such as RULER [Hsieh et al., 2024], to demonstrate complex long-context understanding.

In Section 5, we discuss the architectural decisions of prior work on soft-token compression as part of our architectural search and analysis. Our work shows that, with large-scale staged training, a single online soft-token compressor can robustly handle general heterogeneous inputs while closely matching uncompressed behavior.

3 The Latent Context Language Model Architecture

An LCLM consists of an encoder with a pooling method that maps token chunks to soft tokens, an adapter to project to the decoder's embedding dimension, and a decoder that consumes the soft tokens as latent context. Given a sequence of input tokens $x_{1:T}$ and a compression ratio N, the encoder maps each contiguous block of N input tokens to a single latent token. Let the number of input tokens processed by the encoder in one forward pass, the encoder window size, be denoted by W. We split the input sequence into $I = \lceil T/W \rceil$ encoder windows, each containing at most W tokens:

$$w_i = x_{(i-1)W+1\,:\,\min(iW,T)}, \qquad i = 1, \ldots, I. \tag{1}$$

For each window w_i, the encoder computes the final hidden states $h^{(i)}_{1:|w_i|}$ for each input token in the window. A pooling operator then aggregates these hidden states into $M_i = \left\lceil \frac{|w_i|}{N} \right\rceil$ latent tokens $z^{(i)}_{1:M_i}$. When $W = N$, each encoder window contains a single compression block, yielding one latent token per forward pass. When $W = T$, the encoder processes the entire input sequence in one forward pass and produces $\lceil T/N \rceil$ latent tokens. We concatenate the latent tokens from all encoder windows to obtain the full compressed latent sequence $z_{1:M}$, where M is the total number of latent tokens after compression.

Because the encoder and decoder may have different hidden dimensions, an adapter $a(\cdot)$ projects the compressed latent sequence $z_{1:M}$ from the encoder hidden dimension into the decoder hidden dimension, producing latent tokens $s_{1:M}$. The decoder then consumes these projected latent tokens

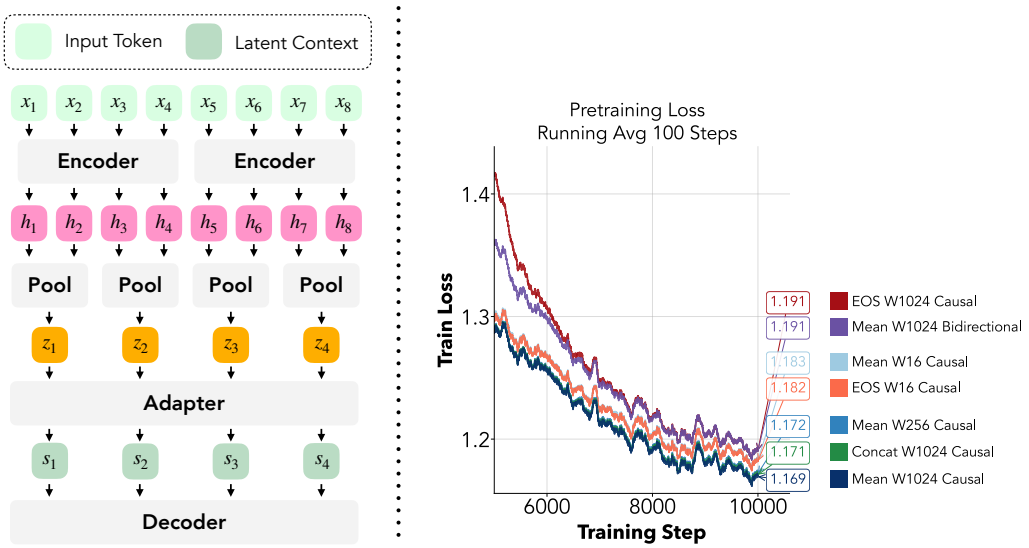

Figure 2: **Left:** We explore how to best train an encoder LM to compress raw text into latent vectors and a decoder LM that uses the vectors as latent context. **Right:** We search end-to-end training recipe across many variants of encoder-decoder compressors with 38B training tokens and compare pretraining loss to find the best architecture. Extended experiments are presented in Appendix Appendix D.

in place of the corresponding input tokens. Full definitions of the architectural variants are provided in Appendix Appendix D.1.

4 Training Recipe

We now describe our multi-stage recipe and data for training. In contrast to prior context-compression work that trains specialized models on small-scale in-domain datasets [Tang et al., 2025, Feldman and Artzi, 2025, Pilchen et al., 2025, Cheng et al., 2024, Liao et al., 2025], our goal is to preserve the strong performance of a powerful LLM across downstream tasks. To this end, we curate three types of data: continual pretraining data, SFT data, and auxiliary reconstruction data.

Continual Pretraining Dataset. We construct a carefully curated high-quality pretraining mixture covering Common Crawl text, code, science and reasoning, long-context data, and instruction-style data. We partition each sequence into multiple segments and alternate between compressed segments and standard token segments. We then compute the next-token prediction loss on the uncompressed tokens *only*. This interleaved format differs from the simple first-half-compressed, second-half-trainable split used in prior work [Lin et al., 2025, Ge et al., 2023, Chevalier et al., 2023, Yen et al., 2024]. By distributing compressed spans throughout the sequence, the model learns to condition on latent context at multiple positions rather than only at the beginning of the context. Since loss is computed only on the uncompressed tokens, the pretraining loss in Figure 2 is substantially lower than a standard next-token prediction loss over all tokens. We provide a visual example of this data format in Figure 6. The exact pretraining mixture is described in Appendix C.1, with additional details on compression formatting in Appendix C.4.

SFT Data. To recover performance after continual pretraining, we further post-train the model with supervised fine-tuning to improve reasoning, instruction following, and long-context understanding when conditioning on compressed inputs. We construct an SFT mixture spanning three components: (i) reasoning, (ii) long-context instruction following, and (iii) general instruction following and alignment. Some completions in the original SFT data were generated by outdated models. To improve response quality, we relabel a subset of the SFT data using `Qwen3-30B-A3B-Instruct-2507` and `Qwen3-235B-A22B-Instruct-2507`, whose model checkpoints are released under the Apache-2.0 license. We report the exact SFT mixture and other details in Appendix C.3.

Auxiliary Reconstruction Data. To accelerate training and encourage the latent representations to preserve fine-grained information, we introduce an auxiliary reconstruction task. Given a sampled document, we compress the text and ask the model to repeat the original content. To increase diversity, we include a variety of sources spanning code, text, long documents, math, and LaTeX, as described in Appendix C.1. To reduce prompt overfitting, we use a bank of 100 prompt templates per source. This auxiliary task encourages the compressor not only to support semantic understanding, but also to retain fine-grained details needed for tasks such as exact retrieval. We include auxiliary reconstruction data in both continual pretraining and SFT.

4.1 Training Stages

We adopt a multi-stage training recipe to improve stability, preserve the original model's capabilities, and mitigate catastrophic forgetting. The stages progressively increase the number of trainable components, beginning with adapter alignment and ending with supervised fine-tuning on compressed inputs.

- **Stage 0: Adapter pretraining.** We train on 39B tokens while freezing both the encoder and decoder, updating only the adapter.
- **Stage 1: Encoder pretraining.** We train on 78B tokens after unfreezing the encoder, while keeping the decoder frozen.
- **Stage 2: Decoder continual pretraining.** We train on 182B tokens after unfreezing the decoder, using a small learning rate for continual pretraining.
- **Stage 3: Supervised fine-tuning.** We train on 51B tokens of the SFT mixture and increase the decoder learning rate.

This staged procedure is analogous to Vision-Language Model (VLM) training pipelines that first align pretrained representations across modalities [Liu et al., 2023a, Tong et al., 2024] and then fine-tune for downstream tasks. For stages 0, 1, and 2, we uniformly sample from the continual pretraining mixture; for stage 3, we train on the SFT mixture. In early experiments, we considered directly training the full model end-to-end from the beginning, but found that this approach underperformed the multi-stage recipe. At the beginning of training, the decoder is not accustomed to taking in the output of the embedding model, so both encoder and decoder parameter gradients can be large, leading to model degradation. The staged approach, where we first train the adapter before unfreezing the encoder and decoder prevents this degradation. We therefore adopt the multi-stage recipe for all scaled experiments. Details of the continual pretraining and SFT data mixtures are provided in Table 1 and Table 2, respectively. All other training details, including token budgets, learning rates, and sequence lengths, are provided in Table 4 and Appendix F.

Model Selection. For our main experiments, we use `Qwen3-Embedding-0.6B` as the encoder with `Qwen3-4B-Instruct-2507` as the decoder. We study alternative model choices, including whether to initialize encoder from an embedding model or language model in Appendix E.1. In Appendix E.4, we also analyze increasing encoder and decoder parameter count.

5 Architecture Ablations: Design Space of Latent Encoders

We conduct a comprehensive search over candidate encoder design choices by pretraining LCLM models from scratch. This cleanroom setting avoids confounding factors from pretrained initializations; for example, we experiment with different attention masks for the encoder, without favoring the causal attention used in the pretrained `Qwen3-Embedding-0.6B` [Zhang et al., 2025b] encoder. Specifically, we use `Qwen3-0.6B` as the base architecture for both the encoder and decoder, initialize all weights from scratch, and train all components end-to-end with no frozen parameters. We pretrain on the mixture described in Section 4 for 38B tokens, using compression ratio $N = 16$.

Pooling Operator. Prior work has primarily considered two classes of pooling strategies. First, token-based pooling appends or prepends one or more designated tokens, such as `EOS` or `CLS`, to the input sequence and uses the corresponding final hidden states as the compressed representations [Lin et al., 2025, Liao et al., 2025, Chevalier et al., 2023]. Second, mean pooling forms each compressed representation by averaging the encoder hidden states over a block of N input tokens [Ge et al., 2023, Tang et al., 2025].

In Figure 2, we compare EOS and mean pooling, finding mean pooling to achieve consistently lower pretraining loss. Empirically, mean pooling has been found to improve performance relative to token-based pooling in the compression setting [Feldman and Artzi, 2025], with similar observations in vision architectures [Chu et al., 2021, DeepMind, 2026]. Our work validates this in a fully controlled setting, training from scratch with a much larger architectural search space. Moreover, we validate our findings at larger scale in Appendix G.4.

Encoding Granularity. Ideally, one would set the encoder window size to the full context length, allowing the encoder to jointly contextualize all input tokens, analogous to ViT-style encoders that process the entire image at once. For language inputs, however, the context length can be substantially larger, making full-context encoding prohibitive in both memory and time.

To study this trade-off, we vary the encoder window size over $W \in \{N, 256, 1024\}$ in Figure 2. We find that increasing W from N (16) to 256 leads to a large improvement, while increasing W further from 256 to 1024 yields an additional smaller gain. Since $W = 1024$ does not introduce significant memory or runtime overhead in our setting, we adopt $W = 1024$ as the default. This setting enables the encoder to attend over a larger local context and produce richer compressed representations. We also confirm at larger scale that $W = 1024$ outperforms $W = 256$ on pretraining loss and downstream evaluations in Figure 9 and Table 22.

Encoder Attention Mask. Unlike standard decoder language modeling, where causal masking is required to match autoregressive inference, context compression is applied to the prompt before decoding. The encoder can therefore use either causal or bidirectional attention, as in prefix-LMs [Raffel et al., 2020]. In Figure 2, we compare these two masking choices and find that causal masking consistently achieves lower pretraining loss.

Overall, our from-scratch pretraining experiments in Figure 2 show that the best-performing compressor uses a causally masked encoder with mean pooling, an encoder window size of $W = 1024$, and an MLP adapter. We therefore use this architecture as the default in our main experiments. We emphasize that prior work does not perform such a large scale and varied sweep over encoder-decoder architectures in a controlled setting. In Appendix D, we further analyze adapter design and overlapping context between encoder windows, finding that an MLP adapter and no overlap perform best.

To verify that small-scale pretraining experiments are predictive of large-scale behavior, we compare our from-scratch architecture sweeps with large-scale ablations using the full pretraining pipeline. In Appendix E, Figure 9, and Appendix G.4, we show that the trends observed in the small-scale sweeps generalize at scale, both in continual pretraining loss and downstream benchmark performance.

6 Results: Improving Latency- and Memory-Performance Tradeoffs

We predominantly focus on long-context benchmarks: we evaluate on RULER [Hsieh et al., 2024], LongBench [Bai et al., 2024], and LongHealth [Adams et al., 2025]. For the long-context evaluation suite, we maintain instructions as uncompressed tokens, while we compress long-context segments. We report which parts of each benchmark are compressed, and which are left as standard uncompressed tokens in Table 5.

We measure two efficiency axes: peak memory during compression and compression time. Peak memory includes both the encoder and decoder for a single sample at each context length. We report the mean over 30 measurements on the same H200 GPU, using 20 warmup steps to reduce kernel-level noise. We define compression time as time-to-first-token (TTFT), i.e., the time required for a method to reach the point at which it can generate the first token. We adapt the KVPress [Devoto et al., 2025] timing code[1] to support our method. All methods are benchmarked using Hugging Face Transformers [Wolf et al., 2020] implementations.

Since LCLMs process input tokens in encoder-window chunks, we can parallelize compression by batching encoder forward passes before passing the resulting soft tokens to the decoder. For all figures in this paper, we use an encoder batch size of 128; with encoder window size $W = 1024$, this corresponds to processing 131,072 input tokens per batched encoder pass. We find this batch size provides a good trade-off between compression speed and encoder memory usage, improving GPU

[1] https://github.com/NVIDIA/kvpress/blob/main/notebooks/speed_and_memory.ipynb

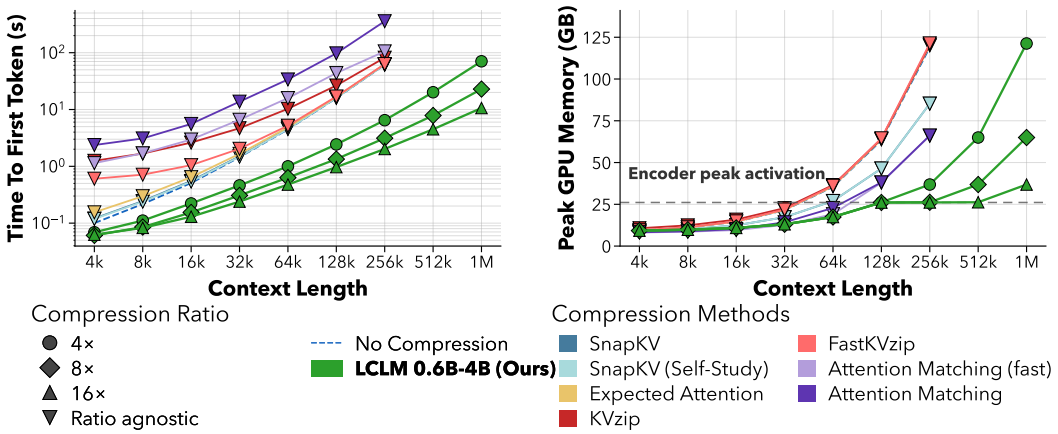

Figure 3: **Latent Context Language Models have lower TTFT and peak GPU memory as context length increases.** We plot time to first token (left) and peak GPU memory (right) for a single sample. We note the baselines use the same time and memory regardless of compression ratio, hence there is only one line per baseline method. We record all measurements on the same H200 GPU, and truncate lines when the method cannot be performed within the 141GB of memory. For peak GPU memory, our method plateaus at longer contexts for larger compression ratios: in this regime, the encoder's batched processing dominates memory usage over the decoder. Memory begins to grow again once the decoder's prefill memory surpasses the encoder's activation per forward pass.

utilization without exceeding memory limits for ultra-long-contexts. Moreover, because our method preserves the standard KV cache structure, prefill and generation can be further accelerated with standard inference frameworks such as vLLM [Kwon et al., 2023] and SGLang [Zheng et al., 2024]. Therefore, the throughput measurements reported here are conservative relative to what optimized serving implementations could achieve.

6.1 Baselines

We consider a range of KV cache compression baselines, including Expected Attention [Devoto et al., 2025], Attention Matching [Zweiger et al., 2026], KVzip [Kim et al., 2025], Fast KVzip [Kim et al., 2026], and SnapKV [Li et al., 2024c]. Each method has different native support for query-aware, query-agnostic, and self-study compression configurations.

Query-aware methods, such as SnapKV, are designed to compress a context given a known query; however, the resulting compressed cache may not generalize well to unseen queries or future turns. Query-agnostic methods, such as KVzip and Fast KVzip, compress a cache without access to the downstream query, often using reconstruction-style objectives such as repeating the original context. Self-study, as introduced by Eyuboglu et al. [2025], performs compression using synthetic queries to expand coverage over potential inference-time queries. These synthetic queries are typically produced by prompting the target decoder to generate questions about the context. In Attention Matching [Zweiger et al., 2026], the authors experiment with a mixed query-agnostic and self-study configuration, and find that self-study has limited impact over the repeat-prefill prompt. Since self-study introduces additional compression-time overhead, we do not apply it to RULER.

It is also important to distinguish algorithmic and theoretical cache reduction from systems-realizable cache reduction. Some KV cache methods produce an actual smaller cache that can, in principle, reduce decoding memory and latency. Others are evaluated by masking or subsampling entries from a full cache, or by using non-uniform eviction patterns across heads and layers. These approaches are useful for measuring the quality of a compressed cache, but they do not necessarily translate into wall-clock speedups or memory savings in standard inference engines without specialized kernels that have not been released or even implemented.

We use the KVPress [Devoto et al., 2025] implementation for SnapKV, KVzip, Fast KVzip, and Expected Attention. We use the official implementation for Attention Matching. We discuss all baselines further in Section 2 and Appendix B.

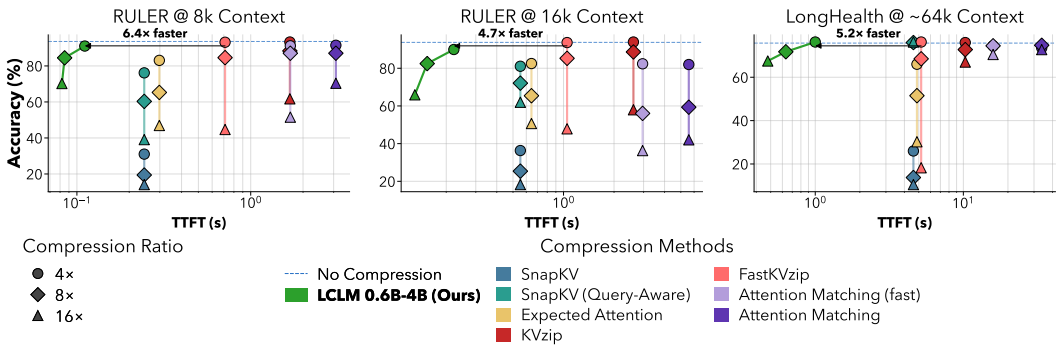

Figure 4: **Latent Context Language Models establish a new Pareto frontier on long-context benchmarks in terms of time to first token.** We plot accuracy across RULER and LongHealth seeing our LCLMs compress samples faster with equivalent or higher accuracy than other compression methods. Full results can be viewed in Tables 6, 8, and 9.

6.2 A New Pareto Frontier

We train LCLMs with compression ratios of $16\times$, $8\times$, and $4\times$. In Figures 1 and 4, LCLMs establish a new Pareto frontier in compression time and accuracy over KV cache compression baselines on RULER, LongBench, and LongHealth. KV cache compression methods appear as nearly vertical lines in these plots because their compression time is largely independent of the target compression ratio. These methods first materialize the full KV cache and then perform eviction, masking, or compaction, whose cost is small relative to the full-context prefill. In contrast, LCLMs reduce the sequence length before decoder prefill, so higher compression ratios directly reduce the amount of decoder-side computation and memory.

Time and memory scaling with context length. In Figure 3, we visualize compression time and peak GPU memory for our method and the baselines across context length from 4K to 1M. LCLMs achieve the fastest compression time and, at longer context lengths, substantially lower peak GPU memory. Attention Matching runs out of memory at 1M tokens and fails at 512K tokens due to numerical instability in the linear solver. All other methods run out of memory at 512K and 1M tokens. This efficiency stems from two properties of LCLMs: compression is performed over fixed-size encoder windows rather than through a full-context decoder prefill, and the encoder is much smaller than the target LLM. Since our encoder processes at most 128K input tokens per batched forward pass, peak memory is initially dominated by encoder activations. As a result, peak memory remains nearly flat for the $16\times$ model from 128K to 512K tokens, and for the $8\times$ model from 128K to 256K tokens. At longer contexts, the decoder prefill over the compressed latent sequence becomes the dominant memory cost, causing peak memory to increase.

Fine-Grained Compression. In Appendix Figure 11, we use GSM8K to analyze fine-grained compression quality, where we compress the entire prompt and context. We find LCLMs achieve the highest accuracy across all compression ratios on GSM8K, with particularly strong gains over baselines at higher compression ratios.

7 Agentic Information Retrieval

Agentic processes are commonly used to process a large amount of information [Zhang et al., 2025a]. However, agents usually rely on lexical or semantic search to locate the information needed. These information retrieval methods can fail when the search keyword is not obvious before reading through the information. For example, in a large codebase, a bug reported in the "dashboard login flow" may actually originate in an indirectly called entitlement module that never mentions "dashboard" or "login". Ideally, an agent can read all information at once before acting, but context limits of existing models prevent this from happening.

LCLM's more efficient long context capacity makes it possible to place, for example, an entire code repo in an agent's context. We further enhance the agent's capability to locate precise information by allowing LCLM agent to expand compressed segments into raw text on demand. We segment the

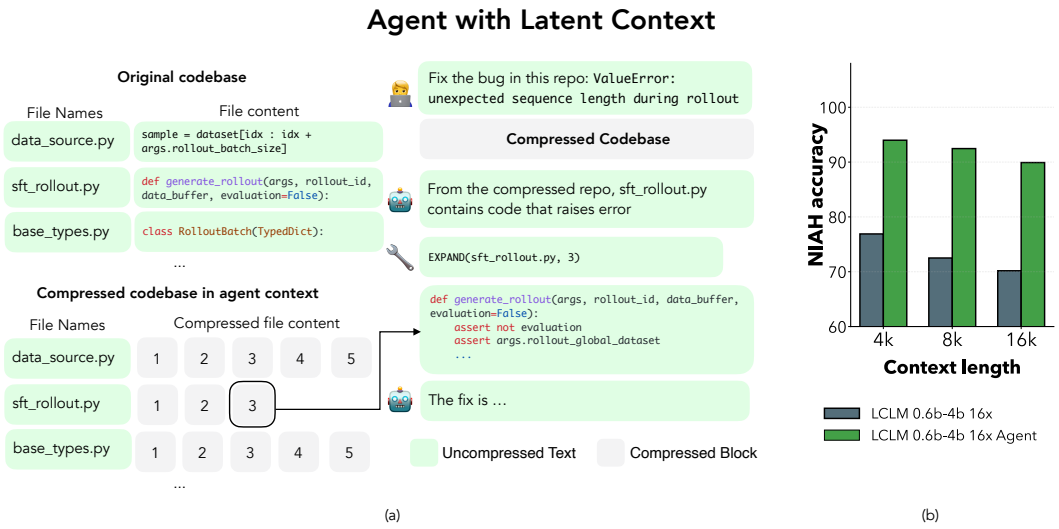

Figure 5: **LCLMs can use tools to retrieve compressed context and improve exact string-match accuracy.** (a) We equip LCLMs with a tool that retrieves compressed context and inserts it into their working context uncompressed. (b) This substantially improves exact string-match accuracy on retrieval tasks.

input into fixed-size chunks of 512 tokens, compress each chunk, and assign it an integer identifier. The model receives the entire compressed sequence in a single prompt, together with one `EXPAND` tool. At each turn, the model can decide to make a tool call in the form of `EXPAND(i)`, and the tool then returns original texts of segments expanded.

We use the needle in a haystack tasks from RULER as a testbed for evaluating this agentic retrieval mechanism with latent context. As shown in Figure 5, our agent substantially improves performance over the raw LCLM with $16\times$ compressed context, and in some settings matches the performance of the original uncompressed context. These results suggest that compressed latent context can provide broad corpus-level visibility, while tool-based expansion further enhances the exact fine-grained information needed. The marked success of this initial agentic attempt suggests the promise of adaptive expansion. LCLMs can skim a large quantity of context globally before deciding which relevant subset to zoom in on and read more carefully. Future work may explore new mechanisms for determining which context to expand or how to learn this behavior end-to-end.

8 Conclusions

Across long-context benchmarks, standard knowledge and instruction following tasks, and in agentic settings, our family of Latent Context Language Models demonstrates that learned compression can preserve strong in-context capabilities while providing substantial efficiency gains, making it a practical building block for efficient long-context systems.

Latent Context Language Models open a broad design space for future work and are naturally compatible with agentic frameworks such as Recursive Language Models [Zhang et al., 2025a]. More generally, LCLMs provide a promising substrate for long-horizon agents with large, persistent working memories by dramatically reducing the size of the inputs at scale. Future iterations of LCLMs could further improve the quality-efficiency trade-off by compressing inputs at multiple granularities and dynamically allocating capacity based on information density or input perplexity. Such adaptive compression could allow models to preserve fine-grained details where needed while maintaining a compact global context, reducing reliance on explicit expansion tools. Another promising direction is to extend compression beyond static input context to the model's generated state, including long chain-of-thought, tool observations, and an agent's accumulated working history, which can grow to dominate the context budget in long-horizon tasks.

References

Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo JWL Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. Longhealth: A question answering benchmark with long clinical documents. *Journal of Healthcare Informatics Research*, 9(3):280–296, 2025.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Tushar Aggarwal, Swayam Singh, Abhijeet Awasthi, Aditya Kanade, and Nagarajan Natarajan. Nextcoder: Robust adaptation of code LMs to diverse code edits. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=3B6fF1PxYD.

Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong. Why does the effective context length of LLMs fall short? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=eoln5WgrPx.

Anthropic. Claude Code, 2025. URL https://github.com/anthropics/claude-code.

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 (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*, 2025.

Longze Chen. Awesome-kv-cache-compression. GitHub repository, 2023. URL https://github.com/October2001/Awesome-KV-Cache-Compression.

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. *Advances in Neural Information Processing Systems*, 37:109487–109516, 2024.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*, 2023.

Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, and Chunhua Shen. Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882*, 2021.

Yu-Neng Chuang, Tianwei Xing, Chia-Yuan Chang, Zirui Liu, Xun Chen, and Xia Hu. Learning to compress prompt in natural language formats. *arXiv preprint arXiv:2402.18700*, 2024.

Domenico Cotroneo, Giuseppe De Rosa, and Pietro Liguori. Pyresbugs: A dataset of residual python bugs for natural language-driven fault injection. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pages 146–150, 2025. doi: 10.1109/Forge66646.2025.00024.

Yuhong Dai, Jianxun Lian, Yitian Huang, Wei Zhang, Mingyang Zhou, Mingqi Wu, Xing Xie, and Hao Liao. Pretraining context compressor for large language models with embedding-based memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 28715–28732, 2025.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Google DeepMind. Gemma 4: Open lightweight language models. 2026. URL https://ai.google.dev/gemma.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, et al. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*, 2025.

Yair Feldman and Yoav Artzi. Simple context compression: Mean-pooling and multi-ratio training. *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. *arXiv preprint arXiv:2307.06945*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Linda He, Jue Wang, Maurice Weber, Shang Zhu, Ben Athiwaratkun, and Ce Zhang. Scaling instruction-tuned llms to million-token contexts via hierarchical synthetic data generation. *arXiv preprint arXiv:2504.12637*, 2025.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

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 (Volume 1: Long Papers)*, pages 1658–1677, 2024.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 2567–2577, 2019.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

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.

Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training, 2024.

Hongyu Li, Liang Ding, Meng Fang, and Dacheng Tao. Revisiting catastrophic forgetting in large language model tuning. In *Findings of the association for computational linguistics: EMNLP 2024*, pages 4297–4308, 2024a.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

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*, pages 6342–6353, 2023.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, et al. Scbench: A kv cache-centric analysis of long-context methods. *arXiv preprint arXiv:2412.10319*, 2024b.

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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024c.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xcompressor: Generalized prompt compression for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25081–25091, 2025.

Zihan Liao, Jun Wang, Hang Yu, Lingxiao Wei, Jianguo Li, and Wei Zhang. E2llm: Encoder elongated large language models for long-context understanding and reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 19212–19241, 2025.

Xiaoqiang Lin, Aritra Ghosh, Bryan Kian Hsiang Low, Anshumali Shrivastava, and Vijai Mohan. Refrag: Rethinking rag based decoding. *arXiv preprint arXiv:2509.01092*, 2025.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023a.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023b.

Wanlong Liu, Junying Chen, Ke Ji, Li Zhou, Wenyu Chen, and Benyou Wang. Rag-instruct: Boosting llms with diverse retrieval-augmented instructions, 2024b. URL https://arxiv.org/abs/2501.00353.

Zihan Liu, Wei Ping, Rajarshi Roy, Peng Xu, Chankyu Lee, Mohammad Shoeybi, and Bryan Catanzaro. Chatqa: Building gpt-4 level conversational qa models. *CoRR*, 2024c.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa

Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. *arXiv preprint arXiv:2308.07124*, 2023.

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v2, aug 2025a. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2.

Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, July 2025b. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1.

NVIDIA, :, Aarti Basant, Abhijit Khairnar, Abhijit Paithankar, Abhinav Khattar, Adithya Renduchintala, Aditya Malte, Akhiad Bercovich, Akshay Hazare, Alejandra Rico, Aleksander Ficek, Alex Kondratenko, Alex Shaposhnikov, Alexander Bukharin, Ali Taghibakhshi, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amy Shen, Andrew Tao, Ann Guan, Anna Shors, Anubhav Mandarwal, Arham Mehta, Arun Venkatesan, Ashton Sharabiani, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaju, Banghua Zhu, Barnaby Simkin, Bilal Kartal, Bita Darvish Rouhani, Bobby Chen, Boris Ginsburg, Brandon Norick, Brian Yu, Bryan Catanzaro, Charles Wang, Charlie Truong, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christian Munley, Christopher Parisien, Dan Su, Daniel Afrimi, Daniel Korzekwa, Daniel Rohrer, Daria Gitman, David Mosallanezhad, Deepak Narayanan, Dima Rekesh, Dina Yared, Dmytro Pykhtar, Dong Ahn, Duncan Riach, Eileen Long, Elliott Ning, Eric Chung, Erick Galinkin, Evelina Bakhturina, Gargi Prasad, Gerald Shen, Haifeng Qian, Haim Elisha, Harsh Sharma, Hayley Ross, Helen Ngo, Herman Sahota, Hexin Wang, Hoo Chang Shin, Hua Huang, Iain Cunningham, Igor Gitman, Ivan Moshkov, Jaehun Jung, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jian Zhang, Jiaqi Zeng, Jimmy Zhang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jonathan Cohen, Joseph Jennings, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kezhi Kong, Krzysztof Pawelec, Kumar Anik, Kunlun Li, Kushan Ahmadian, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Luis Vega, Maer Rodrigues de Melo, Makesh Narsimhan Sreedhar, Marcin Chochowski, Mark Cai, Markus Kliegl, Marta Stepniewska-Dziubinska, Matvei Novikov, Mehrzad Samadi, Meredith Price, Meriem Boubdir, Michael Boone, Michael Evans, Michal Bien, Michal Zawalski, Miguel Martinez, Mike Chrzanowski, Mohammad Shoeybi, Mostofa Patwary, Namit Dhameja, Nave Assaf, Negar Habibi, Nidhi Bhatia, Nikki Pope, Nima Tajbakhsh, Nirmal Kumar Juluru, Oleg Rybakov, Oleksii Hrinchuk, Oleksii Kuchaiev, Oluwatobi Olabiyi, Pablo Ribalta, Padmavathy Subramanian, Parth Chadha, Pavlo Molchanov, Peter Dykas, Peter Jin, Piotr Bialecki, Piotr Januszewski, Pradeep Thalasta, Prashant Gaikwad, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Rabeeh Karimi Mahabadi, Rajen Patel, Ran El-Yaniv, Ranjit Rajan, Ria Cheruvu, Rima Shahbazyan, Ritika Borkar, Ritu Gala, Roger Waleffe, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Sahil Jain, Samuel Kriman, Sanjeev Satheesh, Saori Kaji, Sarah Yurick, Saurav Muralidharan, Sean Narenthiran, Seonmyeong Bak, Sepehr Sameni, Seungju Han, Shanmugam Ramasamy, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shizhe Diao, Shreya Gopal, Shrimai Prabhumoye, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Siddhartha Jain, Somshubra Majumdar, Soumye Singhal, Stefania Alborghetti, Syeda Nahida Akter, Terry Kong, Tim Moon, Tomasz Hliwiak, Tomer Asida, Tony Wang, Tugrul Konuk, Twinkle Vashishth, Tyler Poon, Udi Karpas, Vahid Noroozi, Venkat Srinivasan, Vijay Korthikanti, Vikram Fugro, Vineeth Kalluru, Vitaly Kurin, Vitaly Lavrukhin, Wasi Uddin Ahmad, Wei Du, Wonmin Byeon, Ximing Lu, Xin Dong, Yashaswi Karnati, Yejin Choi, Yian Zhang, Ying

Lin, Yonggan Fu, Yoshi Suhara, Zhen Dong, Zhiyu Li, Zhongbo Zhu, and Zijia Chen. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

OpenAI. Introducing codex, May 2025. URL https://openai.com/index/introducing-codex/. Accessed: 2026-01-09.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards LLMs as operating systems, 2023. URL https://arxiv.org/abs/2310.08560.

Guilherme Penedo. Finewiki, 2025. URL https://huggingface.co/datasets/HuggingFaceFW/finewiki. Source: Wikimedia Enterprise Snapshot API (https://api.enterprise.wikimedia.com/v2/snapshots). Text licensed under CC BY-SA 4.0 with attribution to Wikipedia contributors.

Hippolyte Pilchen, Edouard Grave, and Patrick Pérez. Arc-encoder: learning compressed text representations for large language models. *arXiv preprint arXiv:2510.20535*, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Ryan Synk, Monte Hoover, John Kirchenbauer, Neel Jain, Alex Stein, Manli Shu, Josue Melendez Sanchez, Ramani Duraiswami, and Tom Goldstein. Exploiting sparsity for long context inference: Million token contexts on commodity gpus. *arXiv preprint arXiv:2502.06766*, 2025.

Sijun Tan, Xiuyu Li, Shishir G Patil, Ziyang Wu, Tianjun Zhang, Kurt Keutzer, Joseph E Gonzalez, and Raluca Ada Popa. Lloco: Learning long contexts offline. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17605–17621, 2024.

Jiwei Tang, Zhicheng Zhang, Shunlong Wu, Jingheng Ye, Lichen Bai, Zitai Wang, Tingwei Lu, Jiaqi Chen, Lin Hai, Hai-Tao Zheng, et al. Gmsa: Enhancing context compression via group merging and layer semantic alignment. *arXiv preprint arXiv:2505.12215*, 2025.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai C Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *Advances in Neural Information Processing Systems*, 37: 87310–87356, 2024.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. Enhancing large language model with self-controlled memory framework, 2023. URL https://arxiv.org/abs/2304.13343.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models. *NeurIPS Datasets and Benchmarks Track*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. pages 38–45. Association for Computational Linguistics, October 2020. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. Swe-fixer: Training open-source llms for effective and efficient github issue resolution. *arXiv preprint arXiv:2501.05040*, 2025.

Peng Xu, Wei Ping, Xianchao Wu, Chejian Xu, Zihan Liu, Mohammad Shoeybi, and Bryan Catanzaro. Chatqa 2: Bridging the gap to proprietary llms in long context and rag capabilities. *arXiv preprint arXiv:2407.14482*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents, 2025. URL https://arxiv.org/abs/2502.12110.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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*, pages 2369–2380, 2018.

Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2588–2610, 2024.

Chanwoong Yoon, Taewhoo Lee, Hyeon Hwang, Minbyul Jeong, and Jaewoo Kang. Compact: Compressing retrieved documents actively for question answering. *arXiv preprint arXiv:2407.09014*, 2024.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2025a. URL https://arxiv.org/abs/2512.24601.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P Xing, et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. *arXiv preprint arXiv:2309.11998*, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast kv compaction via attention matching. *arXiv preprint arXiv:2602.16284*, 2026.

A Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be highlighted here.

B Extended Related Work

Hard-token compression. Hard-token methods achieve compression by deleting or rewriting input tokens. Token pruning approaches remove tokens via importance heuristics [Li et al., 2023, Jiang et al., 2023, 2024], while summarization and rewriting methods compress by paraphrasing into fewer tokens [Chuang et al., 2024, Yoon et al., 2024]. While widely used in practice [e.g. context compaction in Claude Code and Codex; Anthropic, 2025, OpenAI, 2025], and very effective for reducing prompt length, these methods are intrinsically lossy: once information is dropped or paraphrased in the discrete token space, exact lexical and structural detail is unrecoverable. This limits their ability to support code fidelity, format-sensitive reasoning, and lookup-style in-context learning in long-horizon settings.

Relatedly, many agent systems externalize memory via summarization, retrieval, or structured stores. Examples include hierarchical memory management [Packer et al., 2023], long-term conversational memory [Zhong et al., 2023, Wang et al., 2023], and agentic memory frameworks [Xu et al., 2025]. While effective in extending interaction horizons, these approaches typically rely on lossy summaries or task-specific memory representations that are external to the base model.

KV cache compression. KV cache compression removes some entries from the KV cache to reduce the size of the KV cache, most methods rely on a hand crafted or learned heuristic to select which entries to drop. SnapKV [Li et al., 2024c] compresses the cache by removing entries with low aggregated query attention on a per-head basis. SnapKV can be prompt agnostic, where the pruning of the context is done without context of what the prompt will be, or prompt dependent, where the pruning of the context is completed with knowledge of the prompt. Prompt dependent methods have the downside of requiring labeling of the context and prompt for inputs, and pruning is highly specific to each individual input, limiting multi-turn chat applicability [Li et al., 2024b]. To overcome these limitations, self-study [Zweiger et al., 2026] can be used to generate synthetic query signals from the context itself, enabling prompt-agnostic pruning without requiring labeled prompt-context pairs.

KVzip [Kim et al., 2025] performs query-agnostic KV cache compression by using teacher-forced context reconstruction as a proxy objective, assigning each KV pair an importance score from the maximum attention it receives during reconstruction, and pruning low-scoring entries. Fast KV Compaction via Attention Matching [Zweiger et al., 2026] compresses the KV cache by fitting a smaller set of keys and values, together with per-entry bias terms, that match the original cache's attention outputs and attention mass on reference queries. Expected Attention [Devoto et al., 2025] predicts KV importance by approximating future queries with a Gaussian distribution, computing each key's expected attention in closed form, and pruning entries with the lowest expected contribution. There are many other KV cache compression techniques and we refer the reader to Awesome KV cache Compression Github [Chen, 2023] for a comprehensive review.

We note that KV cache compression methods also pool before eviction, both Li et al. [2024c] and [Zweiger et al., 2026] use max pooling. In contrast, Devoto et al. [2025] use max pooling, Li et al. [2024c] note that max pooling doesn't significantly impact their performance.

It is also possible to do targeted KV cache compression offline for specific prompts. This approach applies to the setting where it is logical to expend more compute to compress the cache once as this cost will be amortized over many user queries. Synk et al. [2025] compute the full KV cache before applying top-k cache compression and reuse the smaller cache. Eyuboglu et al. [2025] use self study to learn small KV caches per corpus to be encoded.

Soft-token compression. A complementary direction encodes long-contexts into fewer learned continuous tokens. The most successful examples of soft token approaches to compression require training on the target context, such as prefix tuning [Li and Liang, 2021, Mu et al., 2023, Tan et al., 2024]. These methods can achieve comparable in-context performance to the base model with large compression ratios for the specific contexts they are trained on. Other soft token approaches which

do not require training on individual contexts are in-context autoencoders [Ge et al., 2023], and encoder-decoder compression frameworks [Chevalier et al., 2023, Tan et al., 2024, Liao et al., 2025, Dai et al., 2025, Li et al., 2025]. Parallel context encoding methods such as CEPE [Yen et al., 2024] similarly rely on soft representations to expose long-contexts to a decoder without full attention over raw tokens.

C Dataset Details

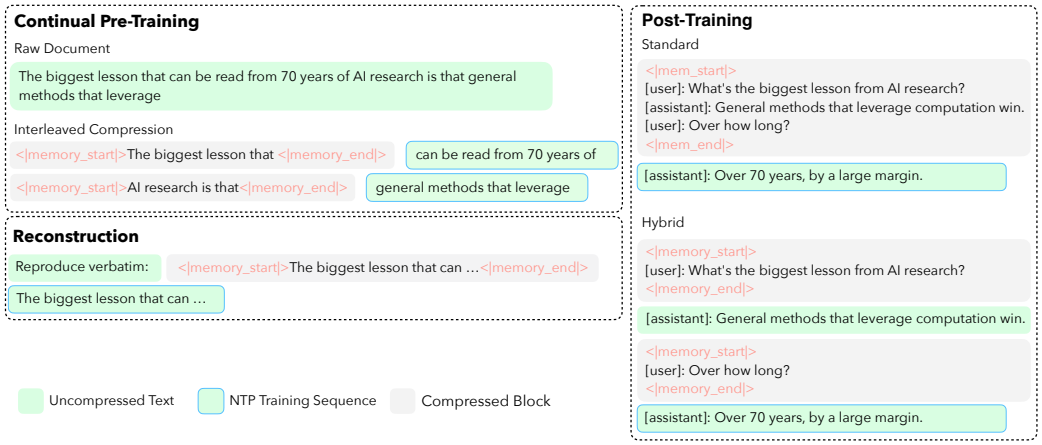

Figure 6: Compression data format examples.

C.1 Continual Pretraining Dataset Curation

We find that training on reconstruction data alone allows the model to reconstruct $x_{1:T}$ with negligible loss, but it generalizes poorly beyond reconstruction-style tasks. In fact, it cannot perform any other tasks, even when keeping the LLM decoder frozen. This might not seem surprising, as we are essentially performing a general version of prefix tuning. Thus, the learned representation, though general for different contexts, collapses to the task of reconstruction. On the other hand, training solely on next-token prediction enables the model to leverage compressed context for downstream generation and reasoning. However, the resulting representations often lack fine-grained fidelity (e.g. exact-string details), leading to weaker reconstruction and brittleness on tasks requiring precise recovery.

Motivated by this trade-off, we train on a mixture of next-token prediction and reconstruction data. Our working hypothesis is that next-token prediction provides task-aligned learning signals that teach the decoder to operate on compressed context, while reconstruction acts as an auxiliary objective that encourages information preservation and can accelerate early-stage representation learning. Theoretically, sufficiently large-scale next-token training alone may recover fine-grained fidelity, but we find that in practice, the reconstruction data serves as an auxiliary data source that accelerates the training of the general compressor.

Curation. Our primary sources are NVIDIA's Nemotron pretraining datasets [NVIDIA et al., 2025, Blakeman et al., 2025]. We sample 50M data points from Nemotron's pretraining code and cc code, 50M examples from high quality and synthetic high quality cc text, and 50M examples from Nemotron specialized SFT mixtures that cover diverse reasoning traces. A substantial portion of the synthetic data consists of rewrites or generations produced by strong teacher models (e.g., `Qwen3-30B-A3B`, `Qwen3-235B-A22B` [Yang et al., 2025], `GPT-oss-120B` [Agarwal et al., 2025], and `DeepSeek-R1` [Guo et al., 2025]), which alleviate the problem of forgetting. Furthermore, to expose the compressor to longer documents, we sample approximately 500K examples from the OLMo-3 longmino collection [Olmo et al., 2025]. Finally, to better preserve instruction-following and chat-template behavior, we also include Nemotron pretraining SFT data and convert it into a standardized instruction tuning and multi-turn chat format. Exact dataset composition is reported in Table 1.

Data Name	License	Data Description	Num Samples	Tokens			%
				Pre-Comp	Post-Comp	Trainable	
Nemotron CC	Nvidia	Continual Pretraining (CC Text)	84.84M	60.69B	30.38B	30.08B	20.37
Nemotron Code	Nvidia	Continual Pretraining (Code)	46.85M	51.18B	25.21B	25.01B	16.90
Nemotron Reasoning	Nvidia	Continual Pretraining (Reasoning)	16.62M	52.32B	25.61B	25.50B	17.17
Longmino	odc-by	Continual Pretraining (Long-Context)	0.548M	26.62B	4.24B	4.23B	2.84
SFT	cc-by-4.0	SFT with Compressed Prompt	20.67M	42.21B	38.02B	37.54B	25.49
Reconstruction	Nvidia	Reconstruction from Compression	23.53M	50.75B	25.69B	25.18B	17.23
Total	–		**192.06M**	**283.78B**	**149.16B**	**147.54B**	**100.00**

Table 1: Dataset mixture for continual pretraining composition. Pre/Post-Comp are token counts before/after compression. We refer to the unique Nvidia license as "Nvidia," the license can be found at https://huggingface.co/datasets/nvidia/Nemotron-CC-v2.1/blob/main/LICENSE.md.

C.2 Auxiliary Reconstruction Dataset Curation

We construct reconstruction data spanning three broad categories: text, code, and LaTeX. Text is sampled from Nemotron CC / math, FineWiki [Penedo, 2025], and RedPajama ArXiv [Weber et al., 2024]; code is sampled from RedPajama GitHub, The Stack V2 (we concatenate files from the repo into one document) [Lozhkov et al., 2024], and CodeParrot; and LaTeX is sampled from TexTeller-OCR. Our goal is to maximize data diversity so that the compressor retains sufficient coverage for downstream tasks that may involve any of these modalities.

C.3 SFT Dataset Curation

For reasoning, we curate math, code, and science supervision from the Nemotron SFT datasets and OLMo-3 Dolci-Think [Nathawani et al., 2025a,b, Blakeman et al., 2025, Olmo et al., 2025].

For long-context instruction tuning, we combine long-context and RAG-style QA datasets [He et al., 2025, Jin et al., 2019, Liu et al., 2024c, Xu et al., 2024, Yang et al., 2018, Liu et al., 2024b, Joshi et al., 2017] with long-context code instruction datasets, including SWE-Fixer [Xie et al., 2025], NextCoder [Aggarwal et al., 2025], PyResBugs [Cotroneo et al., 2025], CommitPack [Muennighoff et al., 2023], and RepoBench [Liu et al., 2023b]. Since some public datasets contain outdated or low-quality responses, we regenerate targets by labeling them with `Qwen3-235B-A22B-Instruct-2507`. In addition, we generate three synthetic long-context corpora: (1) repository summarization, using summarization question templates as instructions and the concatenating repository files from The Stack v2 [Lozhkov et al., 2024] as document; (2) document summarization, using long documents from FineWiki [Penedo, 2025] and RedPajama-ArXiv [Weber et al., 2024]; and (3) repository code generation, where we hold out one file and ask the model to reconstruct it given the remaining repository context. All synthetic completions are generated by `Qwen3-30B-A3B-Instruct-2507`.

For instruction following, we include multi-turn conversation data from WildChat and LMSYS-Chat-1M [Zhao et al., 2024, Zheng et al., 2023], along with additional alignment-style data obtained by re-labeling with `Qwen3-30B-A3B-Instruct-2507`. We also add OLMo-3 instruction tuning, Nemotron instruction tuning, and the Tulu-3 mixture [Lambert et al., 2024]; since these datasets are already generated by strong teacher models, we keep their original responses unchanged. Finally, we retain a small amount of reconstruction data to stabilize training and mitigate drift when optimizing on compressed inputs. Exact dataset composition is reported in Table 2.

C.4 Compression Data Format

Continual Pretraining For the reconstruction data, given a sequence, we split it into multiple segments, we wrap every other segment with special tokens `<|memory_start|>` and `<|memory_end|>`, and treat the wrapped spans as compressed content while leaving the remaining segments uncompressed.

For raw pretraining documents, we construct a fixed alternating pattern of wrapped and unwrapped segments, always beginning with a wrapped segment and ending with an unwrapped segment. Documents longer than 16,384 tokens retain 8,192 trainable (unwrapped) tokens, while shorter documents retain half of their tokens as trainable. The number of segment pairs depends on document

Data Name	License	Description	Num Samples	Tokens			(%)
				Pre-Comp	Post-Comp	Trainable	
Reasoning			**2.67M**	**4.57B**	**3.87B**	**3.83B**	**17.88**
Nemotron Reasoning	cc-by-4.0	Nemotron reasoning data	2.20M	3.90B	3.33B	3.31B	15.41
Dolci Think	odc-by	olmo reasoning data	0.36M	0.50B	0.40B	0.40B	1.87
Nemotron Science	cc-by-4.0	Nemotron science	0.11M	0.16B	0.13B	0.13B	0.60
Long-Context Instruction Tuning			**7.32M**	**47.25B**	**10.63B**	**9.90B**	**49.16**
Repo Summarization	Apache	Synthetic repo-level summarization	1.35M	7.93B	2.45B	2.35B	11.34
Document Summarization	Apache	Synthetic long-document summarization	1.75M	18.49B	3.14B	3.09B	14.52
Repo Code Generation	Apache	Synthetic repo-level code generation	2.02M	7.83B	3.54B	3.38B	16.36
Long-Context QA	Apache	Long-context / RAG QA	0.84M	6.21B	0.30B	0.18B	1.37
Long-Context Code Instruct	Apache	Long-context code instruction tuning	1.36M	6.80B	1.21B	0.89B	5.57
General Instruction Following			**5.26M**	**11.68B**	**4.52B**	**4.29B**	**20.91**
Multi-turn Conversation	odc-by	WildChat + LMSYS Chat 1M	2.48M	7.68B	1.95B	1.80B	9.00
Dolci Instruct	odc-by	olmo instruction tuning	1.46M	2.06B	1.76B	1.73B	8.15
Tulu3 SFT Mixture	odc-by	Tulu3 instruction tuning	0.84M	0.69B	0.40B	0.38B	1.83
Nemotron Instruct	Nvidia	Nemotron instruction tuning	0.35M	1.07B	0.32B	0.28B	1.46
HelpSteer	CC-4.0	Alignment	0.13M	0.17B	0.10B	0.10B	0.48
Reconstruction Data			**2.08M**	**5.13B**	**2.61B**	**2.55B**	**12.05**
Reconstruction	Nvidia	Small reconstruction mixture	2.08M	5.13B	2.61B	2.55B	12.05
Total			**17.32M**	**68.62B**	**21.63B**	**20.56B**	**100.00**

Table 2: SFT data mixture statistics. Pre/Post-Comp are token counts before/after compression; Weight (%) is computed over total Post-Comp tokens. During packing some ultra long data are truncated out, leaving us training with 30B token for the encoder and 20B for the decoder. We refer to the unique Nvidia license as "Nvidia," the license can be found at https://huggingface.co/datasets/nvidia/Nemotron-CC-v2.1/blob/main/LICENSE.md.

length: documents shorter than 2,048 tokens receive 1 pair; documents between 2,048 and 4,096 tokens receive 1-2 pairs; documents between 4,096 and 8,192 tokens receive 2-4 pairs; documents between 8,192 and 16,384 tokens receive 4-6 pairs; and documents longer than 16,384 tokens receive 4-8 pairs. Within each sequence, the first wrapped segment contains 30%-40% of the total wrapped tokens, the last unwrapped segment contains 40%-50% of the total trainable tokens, and the remaining segments are allocated with a $\pm 20\%$ jitter. No segment is allowed to be shorter than 128 tokens.

Instruction Tuning For conversational data, we apply wrapping independently to each message. With 50% probability, the entire message content is enclosed within a single pair of tags. Otherwise, with the remaining 50% probability, we insert 1-3 wrapped regions inside the message and leave the remaining tokens outside the tags. The number of wrapped regions depends on message length: messages shorter than 32 tokens are always fully wrapped; messages between 32 and 200 tokens receive 1 region; messages between 200 and 600 tokens receive 1-2 regions; and messages of 600 tokens or more receive 1-3 regions. Wrapped regions cover 80%-100% of the message tokens in aggregate, and we enforce a minimum 5% unwrapped gap whenever feasible. If the realized wrapped coverage falls below 70%, we fall back to fully wrapping the message. Since this decision is made independently for each message, multi-turn conversations naturally contain a mixture of fully wrapped turns and turns with small unwrapped spans, with a bias toward high overall coverage.

D Extended Architecture Search

D.1 Formal Definitions of Pooling Operators

We now give the formal definitions of pooling operators, and illustrate them visually in Figure 7.

Token based pooling. The most common method is appending one or more learned special pooling token per latent output to each encoder window. Let $p_{i,1}, \ldots, p_{i,M_i}$ denote these pooling tokens. We encode

$$(\tilde{h}_1^{(i)}, \ldots, \tilde{h}_{|w_i|+M_i}^{(i)}) = \mathrm{Enc}_\phi([w_i; p_{i,1}; \ldots; p_{i,M_i}]), \tag{2}$$

and use the hidden states at the pooling-token positions:

$$z_k^{(i)} = \tilde{h}_{|w_i|+k}^{(i)}, \qquad k = 1, \ldots, M_i. \tag{3}$$

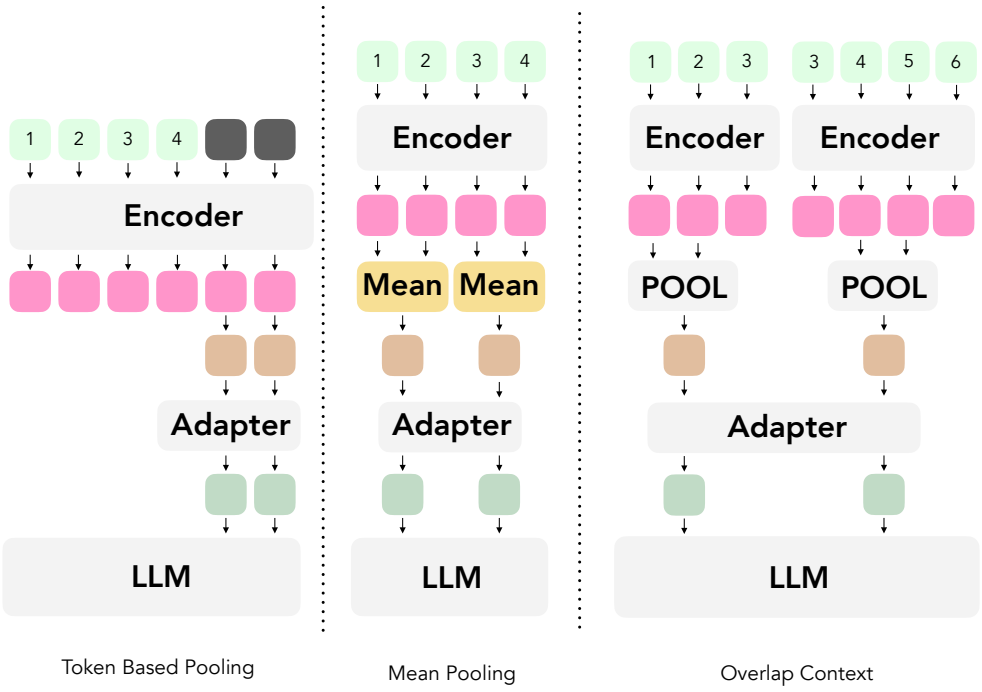

Figure 7: **Full architecture design choices for the compressor.** We visualize token based pooling, mean pooling, and overlap context.

This generalizes the common EOS-pooling strategy: when $W = N$, a single pooling token is appended for each compression block; when $W > N$, several pooling tokens summarize different regions of the same encoder window.

Mean pooling. For each window w_i, we partition its hidden states into consecutive groups of size N. For latent index $k = 1, \ldots, M_i$, define

$$G_{i,k} = \{(k-1)N + 1, \ldots, \min(kN, |w_i|)\}. \tag{4}$$

The corresponding latent vector is the average hidden state over this group:

$$z_k^{(i)} = \frac{1}{|G_{i,k}|} \sum_{j \in G_{i,k}} h_j^{(i)}. \tag{5}$$

Mean pooling directly aggregates the token-level representations assigned to each compression block. Across our architecture search, mean pooling consistently outperforms EOS-style token pooling, matching observations from prior text-compression and vision-encoder work.

In Figure 8, mean pooling achieves lower pretraining loss than EOS/token pooling. We therefore use mean pooling as the default pooling operator.

D.2 Boundary Context

A potential limitation of fixed encoder windows is that information crossing a window boundary is split across two encoder forward passes. Increasing W reduces the frequency of such boundaries, but also increases compression-time cost. As an alternative, we evaluate boundary overlap.

For a target window $w_i = x_{(i-1)W+1:\min(iW,T)}$, we construct an overlapped encoder input $\tilde{w}_i$ by adding O neighboring tokens. For bidirectional attention, we include both left and right overlap when available:

$$\tilde{w}_i = x_{\max(1,(i-1)W-O+1):\min(iW+O,T)}. \tag{6}$$

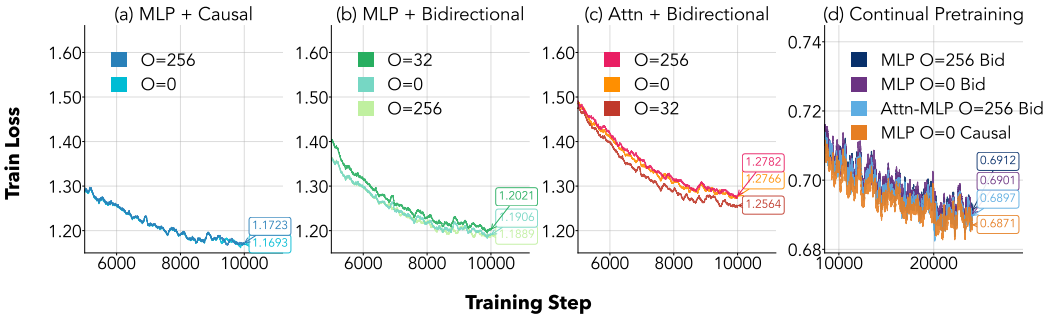

Figure 8: **Adapter, overlap context, encoder mask loss curve** We sweep different combinations between adapter, overlapping context, and encoder masks and found (1) causal mask is strictly better than bidirectional mask, (2) overlapping context does not provide an advantage comparing to non overlapping windows, while significantly increasing computations (3) mlp adapter is better than attention based adapter. We also verify those findings at scale, which can be found in plot d.

For causal attention, future context is unavailable, so we include only left overlap:

$$\tilde{w}_i = x_{\max(1,(i-1)W-O+1):\min(iW,T)}. \tag{7}$$

We apply mean pooling to the hidden states corresponding to the original non-overlapped region w_i, discarding pooled outputs that correspond only to overlap tokens. Thus, overlap changes the encoder context but not the number of latent tokens.

We evaluate $O \in \{0, 32, 256\}$ for $W \in \{256, 1024\}$. Overlap does not noticeably improve pretraining loss in most settings, while increasing training cost because boundary tokens are encoded multiple times. For example, with bidirectional attention, $W = 1024$, and $O = 256$, interior windows process 1536 tokens instead of 1024, increasing encoder-side compute substantially. We therefore use no overlap in the default architecture.

D.3 Adapter Design Choice

The adapter maps encoder latents from dimension d_{enc} to the decoder embedding dimension d_{dec}. Prior encoder-decoder compression work often assumes that the encoder and decoder have matching hidden sizes, but our scaled models use smaller encoders paired with larger decoders. This makes the adapter an important part of the architecture.

We compare three adapter families:

1. an identity adapter when $d_{\text{enc}} = d_{\text{dec}}$;
2. a lightweight MLP projection;
3. attention-based adapters over the latent sequence.

For attention-based adapters, we consider two orderings. In the attention-MLP adapter, self-attention is applied in the encoder hidden dimension before projecting to the decoder dimension. In the MLP-attention adapter, the latents are first projected to the decoder dimension and then processed with self-attention. We use a single attention layer in both cases.

Across the adapter sweep in Figure 8, the MLP-only adapter achieves lower pretraining loss than attention-based adapters while adding less computation. We therefore use an MLP adapter as the default for all main experiments.

E Full Scale Training Recipe Sweeps

E.1 Pretrained encoder representations.

For the encoder initialization, to determine whether an embedding model's representation [Lin et al., 2025, Liao et al., 2025, Yen et al., 2024] is better than the standard language model [Tang et al., 2025, Chevalier et al., 2023, Feldman and Artzi, 2025], we conduct an experiment with the EOS pooling

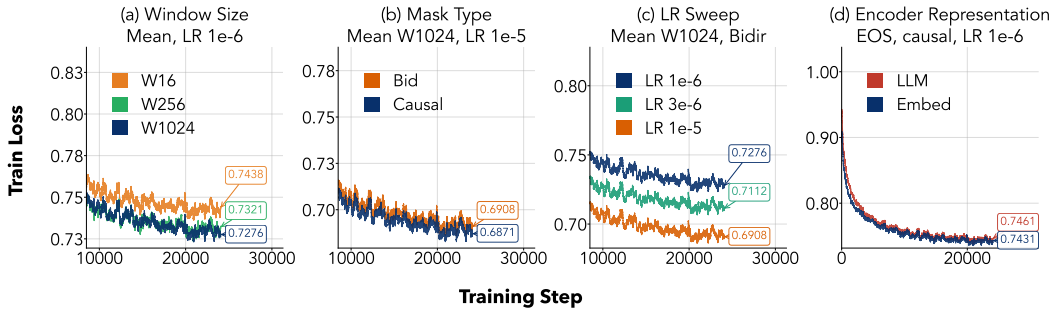

Figure 9: large scale sweep

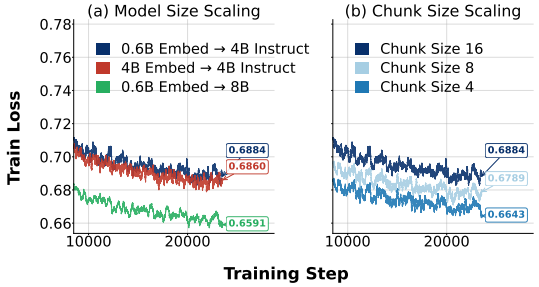

Figure 10: **Left:** We plot the pretraining loss at the end of stage 2 of training and see that increasing the size of the decoder leads to much lower pretraining loss than increasing the size of the encoder. **Right:** We plot the loss at the end of stage 2 of training for our 0.6B-4B models and see decreasing compression decreases the pretraining loss.

token. We can see from Figure 9 (d) that using the embedding model as the encoder outperforms using an LLM as the encoder. Note the embedding model was trained from the LLM model [Zhang et al., 2025b]. This shows that embedding model training provides a better representation for the encoder, though the gap gradually shrinks after training in the long run. We also show downstream evaluation performance in Table 23.

E.2 Encoder Masks.

In Figure 8 (a,b) and Figure 2, we see that causal masking provides better pretraining loss than bidirectional masking in the encoder. We confirm this at full scale, observing the pretraining loss of the causally masked model is again lower than the bidirectionally masked model in Figure 9 and in Table 20.

E.3 LR Sweep

Language models are prone to catastrophic forgetting during continual pretraining and SFT [Luo et al., 2025, Li et al., 2024a]. To actively avoid forgetting, we sweep the peak learning rate for stages 2 and 3 of training. To test for catastrophic forgetting, we then benchmark the language model without compression after SFT and check that its performance is comparable to the benchmark performance of the language model before training. For stage 2, we sweep from $1e{-}6$ to $1e{-}5$ and find that a larger learning rate improves downstream benchmarks without causing forgetting. For stage 3, we sweep from $1e{-}5$ to $5e{-}5$ and find that the optimal performance is achieved at $2e{-}5$ or $3e{-}5$, hence we tune this per model. We show these results in Figure 9 (c). We show in Table 13 that the best LR for stage 2 is $1e{-}5$ and the best LR for stage 3 is $3e{-}5$ across several models.

E.4 Scaling Up Model Size

To study scaling behavior, we ask two questions: (i) does a larger encoder improve compression quality, and (ii) does a larger decoder, which has a wider hidden dimension that could compress

more information per token, improve performance under compression? Accordingly, we add two model configurations: (1) using the larger `Qwen3-Embedding-4B` as the encoder with the same size `Qwen3-4B-Instruct-2507` decoder; and (2) using the same size `Qwen3-Embedding-0.6B` encoder and the larger `Qwen3-8B` as the decoder. In Figure 10 (a), we see that increasing the size of the decoder is much more beneficial in terms of pretraining loss than increasing the size of the encoder.

However, the scaling results in Table 3 are mixed. The 0.6B encoder performs best across the RULER tasks, while the 4B encoder performs best on the remaining evaluations. In contrast, the 8B decoder achieves substantially lower pretraining loss but does not yield the downstream gains we expected. We suspect this is partly due to a mismatch between the training mixture and the decoder initialization: our data curation and post-training recipe were tuned around the 4B instruct decoder, whereas the 8B decoder is a hybrid model and may require a different alignment distribution. Moreover, the 4B instruct model already outperforms the 8B model in the full-cache setting on several evaluations, which may limit the apparent benefit of scaling the decoder in our current setup.

Table 3: Encoder/decoder scaling at 16× compression

Enc / Dec	RULER 4K	RULER 8K	RULER 16K	LongBench	LongHealth5	GSM8K
0.6B / 4B	**75.06**	**70.23**	**65.91**	37.33	<u>67.50</u>	<u>81.05</u>
0.6B / 8B	71.20	<u>67.40</u>	<u>62.22</u>	<u>37.80</u>	64.80	77.30
4B / 4B	<u>71.30</u>	66.90	62.00	**39.00**	**69.80**	**83.20**

F Hyperparameters

Since we continually pretrain an instruction-tuned model, we reset attention masks across different samples within a packed sequence (block-diagonal attention) and use variable-length attention kernels for efficiency [Dao, 2023]. We do not reset the positional index for the decoder across samples to allow the model to generalize to longer context lengths.

We maintain a global batch size of 4M tokens (256 packed sequences per step). We use cosine learning-rate schedules with warmup and decay for all stages and apply no weight decay to the language model parameters. Additional optimizer hyperparameters such as Adam β_1, β_2 are reported in Table 4.

	Stage 0 Adapter Training	**Stage 1 Encoder Training**	**Stage 2 LLM Training**	**Stage 3 SFT**
Training Config				
Adapter Peak LR	1.0×10^{-3}	6.0×10^{-5}	6.0×10^{-5}	3.0×10^{-5}
Encoder Peak LR	NA	6.0×10^{-5}	6.0×10^{-5}	3.0×10^{-5}
LLM Peak LR	NA	NA	1.0×10^{-5}	3.0×10^{-5}
LR scheduler	5% warmup steps with cosine decay to 1.0×10^{-6}			
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$)			
Data				
LLM Batch size (tokens)	4 Million			
LLM Sequence Length	16384			
Encoder Tokens	17.54 B	35.07 B	82.43 B	29.73 B
LLM Tokens (16x)	21.29 B	42.58 B	100.08 B	21.32 B
Total Tokens (16x)	38.83 B	77.65 B	182.51 B	51.05 B
Data Source	Pretraining Mixture	Pretraining Mixture	Pretraining Mixture	SFT

Table 4: **Training recipe for compression model for four different stages.** We report, for each stage, which module(s) are optimized (adapter / encoder / decoder), peak learning rates, learning-rate schedule, optimizer, global batch size (in tokens), sequence length, training-token budget, and the data source used.

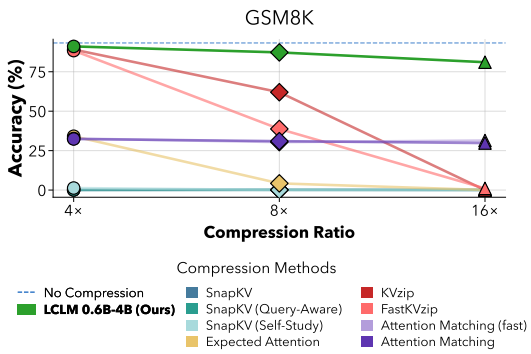

Figure 11: **Latent Context Language Models can compress small dense contexts with high accuracy.** We plot the GSM8K accuracy over different compression ratios, seeing LCLMs can maintain much higher accuracy at larger compression ratios. Full data provided in Table 6. We do not provide compression time for GSM8K as the context is so small, the time to compress is too noisy to provide reliable conclusions.

F.1 Compute

All pretraining from scratch experiments are completed in approximately 16 hours on 16 nodes of MI300A GPUs, each containing 4 GPUs. Full scale continued pretraining experiments for 0.6B-4B models take approximately 4 to 5 days on 32 nodes of MI300A GPUs, each containing 4 GPUs.

All evaluations are conducted on H200 GPU and take more than 5,000 GPU hours to run all baselines.

All scaling experiments, i.e., `Qwen3-Embedding-4B` as the encoder and `Qwen3-8B` as the decoder, are trained on H200 clusters with 32 nodes to alleviate memory issues.

G Benchmarks and Results

G.1 GSM8k Performance vs Compression Ratio

The long-context benchmarks show that LCLMs can compress extended contexts efficiently. We next evaluate on GSM8K to test whether LCLMs can also handle short, information-dense inputs, where nearly every token may be relevant to the answer. In Figure 11, LCLMs achieve the highest accuracy across all compression ratios on GSM8K, with particularly strong gains over baselines at higher compression ratios. These are aggressive compression settings: compression ratios of $16\times$ and $8\times$ remove 93.75% and 87.5% of the input tokens, respectively. Strong performance in this setting demonstrates that LCLMs are not specialized only for long-context QA; they can also compress short, dense contexts, making them practical for general-purpose deployment.

Benchmark	Description	What is Compressed
Long-context		
RULER	Synthetic long-context stress tests spanning 4K-16K.	Task-specific context
LongBench	Aggregated long-context suite covering recall, RAG-style QA, reranking, citation, long QA, summarization, and ICL.	Task-specific context
LongHealth	Clinical-document multiple-choice QA with patient-case context. 5 Document concatenated. No CoT Prompting.	Task-specific context
Fine Grained Compression		
GSM8K	Short and information dense grade-school math word problems.	Whole prompt

Table 5: **Evaluation benchmarks and what content is compressed.** We evaluate on long-context suites, general reasoning, knowledge, and instruction-following benchmarks, and an agentic coding benchmark. The "What is Compressed" column specifies which part of the input is placed inside the compressible memory block (task-specific long-context, whole prompt, or repository context), while remaining components (e.g., instructions) are kept as hard tokens when applicable.

G.2 Additional results

Table 6: Summary across compression ratios for all benchmarks. Best per column within each ratio block in **bold**, second-best underlined.

Model	RULER 4k	RULER 8k	RULER 16k	LB en16	LB cn5	LongHealth5	GSM8K
Qwen 4B Instruct (full KV)	94.41	93.58	93.74	45.21	46.63	75.75	93.25
16x compression							
LCLM 0.6b-4b	**75.06**	70.23	**65.91**	39.08	31.74	67.50	**81.05**
ExpAttn	40.97	46.93	50.67	29.44	20.51	30.25	0.08
SnapKV	14.31	14.11	18.27	26.96	18.30	10.50	0.08
SnapKV-QA	20.54	39.09	61.96	38.52	**42.26**	**76.25**	0.08
KVzip	62.73	61.73	57.97	25.86	17.12	67.00	0.00
KVzipFast	40.01	44.70	47.86	25.76	17.08	18.25	1.06
AM-Fast	53.09	55.45	37.17	**40.49**	31.51	70.50	31.39
AM	69.21	**70.38**	42.02	40.26	31.85	72.75	29.80
8x compression							
LCLM 0.6b-4b	85.42	84.48	82.47	42.23	34.61	71.75	**87.26**
ExpAttn	65.08	65.28	65.41	38.13	30.56	51.50	4.25
SnapKV	19.64	19.42	25.39	32.28	25.18	13.75	0.15
SnapKV-QA	47.66	60.32	72.10	41.73	**45.41**	**76.00**	0.15
KVzip	**90.27**	**88.34**	**88.65**	40.72	36.09	72.75	62.02
KVzipFast	86.08	84.59	85.27	38.17	30.91	68.50	38.74
AM-Fast	89.63	87.05	56.06	**44.50**	41.52	74.50	30.40
AM	89.20	87.05	59.34	44.22	41.11	74.75	31.01
4x compression							
LCLM 0.6b-4b	91.76	91.03	89.96	**46.04**	40.92	**76.25**	**91.05**
ExpAttn	85.26	83.09	82.51	41.95	41.57	66.00	34.12
SnapKV	27.84	31.02	36.33	38.18	34.22	26.00	0.08
SnapKV-QA	69.24	76.16	81.06	43.74	46.19	75.75	0.08
KVzip	**94.43**	**93.28**	**93.93**	45.30	47.45	76.00	89.08
KVzipFast	94.17	93.15	93.64	45.16	**48.66**	**76.25**	88.40
AM-Fast	92.69	91.60	82.37	45.48	48.39	74.50	32.68
AM	92.52	91.47	81.98	45.78	48.07	73.50	32.37

Table 7: RULER per-task at 4k context.

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	vt	cwe	fwe	qa1	qa2	AVG
Qwen 4B Instruct	100.00	100.00	100.00	99.80	96.40	99.80	99.85	100.00	99.84	97.50	87.53	84.40	62.20	94.41
16x compression														
LCLM 0.6b-4b	96.40	89.40	54.60	**85.00**	80.80	**48.60**	**72.45**	**80.20**	83.92	**80.54**	**88.27**	66.00	49.60	**75.06**
ExpAttn	**100.00**	49.20	0.60	57.60	2.20	0.00	56.75	25.70	97.80	18.36	66.20	33.20	25.00	40.97
SnapKV	4.00	1.60	2.40	9.80	1.80	0.00	8.55	8.55	5.48	25.30	73.13	24.80	20.60	14.31
SnapKV-QA	16.60	7.20	2.40	10.20	4.80	0.00	9.10	10.50	15.80	25.14	57.67	60.60	47.00	20.54
KVzip	**100.00**	91.60	**96.00**	62.40	69.80	41.00	32.85	43.35	**99.96**	21.80	70.13	49.00	37.60	62.73
KVzipFast	**100.00**	38.80	31.60	23.00	65.40	26.00	7.70	9.80	99.60	12.66	54.73	24.80	26.00	40.01
AM-Fast	**100.00**	52.80	7.80	33.80	89.00	3.80	22.60	17.85	95.16	76.82	81.40	63.20	46.00	53.09
AM	**100.00**	**95.20**	31.60	74.00	**94.00**	9.40	65.40	56.50	94.96	74.52	81.40	**70.60**	**52.20**	69.21
8x compression														
LCLM 0.6b-4b	98.60	94.20	64.20	93.00	98.20	82.20	92.30	90.70	87.40	**90.80**	**90.60**	73.20	55.00	85.42
ExpAttn	**100.00**	96.20	5.20	92.20	69.20	0.40	93.00	80.20	**100.00**	48.08	85.20	39.20	37.20	65.08
SnapKV	12.40	8.20	2.40	12.60	3.20	0.00	11.20	12.60	9.80	49.06	80.07	31.00	22.80	19.64
SnapKV-QA	54.40	97.40	2.40	46.80	14.40	2.20	21.25	79.65	52.60	43.22	70.20	78.20	56.80	47.66
KVzip	**100.00**	98.20	**99.80**	97.20	98.60	99.40	82.85	96.70	99.96	77.48	82.67	**79.60**	**61.00**	**90.27**
KVzipFast	**100.00**	**100.00**	**99.80**	91.80	98.20	**99.60**	54.15	93.70	99.96	66.52	81.27	76.40	57.60	86.08
AM-Fast	**100.00**	99.40	93.20	**99.20**	98.20	93.00	96.90	**98.45**	96.60	65.90	84.53	79.40	60.40	89.63
AM	99.80	99.40	90.60	98.20	**98.80**	93.60	**97.65**	97.35	97.04	64.80	84.13	78.60	59.60	89.20
4x compression														
LCLM 0.6b-4b	99.40	99.60	93.40	99.00	99.40	93.80	98.25	99.20	94.64	91.42	86.73	79.20	58.80	91.76
ExpAttn	**100.00**	99.80	66.60	**100.00**	99.40	53.60	98.15	99.65	**100.00**	86.74	87.07	62.80	54.60	85.26
SnapKV	32.00	14.80	2.60	18.60	6.00	0.00	13.80	16.55	27.00	71.72	81.87	47.60	29.40	27.84
SnapKV-QA	95.00	**100.00**	2.60	98.60	30.20	40.40	54.30	99.35	85.04	72.14	78.73	81.60	62.20	69.24
KVzip	**100.00**	99.40	**100.00**	**100.00**	**99.60**	99.80	**99.25**	99.80	99.96	**96.24**	85.80	**85.40**	62.40	**94.43**
KVzipFast	**100.00**	**100.00**	**100.00**	**100.00**	**99.60**	99.80	98.50	**100.00**	99.96	95.96	84.40	83.00	63.00	94.17
AM-Fast	**100.00**	99.80	96.80	98.80	**99.60**	**100.00**	98.75	99.40	99.08	78.28	**89.40**	81.80	**63.20**	92.69
AM	**100.00**	**100.00**	97.60	99.00	**99.60**	99.60	99.10	99.20	98.56	76.98	88.07	81.80	**63.20**	92.52

Table 8: RULER per-task at 8k context.

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	vt	cwe	fwe	qa1	qa2	AVG
Qwen 4B Instruct	100.00	100.00	100.00	99.80	97.00	99.40	99.50	100.00	99.24	95.24	87.40	80.40	58.60	93.58
16x compression														
LCLM 0.6b-4b	96.00	90.20	55.20	**83.40**	65.40	36.60	70.25	**77.65**	80.52	55.44	**95.67**	61.80	44.80	70.23
ExpAttn	**100.00**	76.60	1.00	81.20	2.40	0.00	69.30	55.85	99.08	10.52	60.73	30.40	23.00	46.93
SnapKV	8.60	1.20	2.40	10.00	0.80	0.00	9.80	9.80	5.04	24.78	78.60	15.40	17.00	14.11
SnapKV-QA	88.20	62.00	2.40	14.60	9.00	7.60	10.30	44.25	52.68	21.16	68.13	**74.20**	**53.60**	39.09
KVzip	**100.00**	**98.00**	**95.80**	64.80	56.40	**45.20**	31.25	51.05	**100.00**	8.16	73.80	43.20	34.80	61.73
KVzipFast	**100.00**	43.60	38.80	30.80	65.20	39.00	10.85	17.15	99.96	7.42	77.73	23.80	26.80	44.70
AM-Fast	**100.00**	60.00	11.80	42.00	91.24	8.80	28.55	31.70	93.56	**72.90**	73.47	62.80	44.00	55.45
AM	99.20	94.80	38.40	79.40	**93.20**	11.60	**73.45**	72.95	94.08	71.22	75.07	65.20	46.40	**70.38**
8x compression														
LCLM 0.6b-4b	98.80	96.40	71.40	90.80	96.80	74.60	90.40	90.95	89.40	**79.82**	**95.67**	70.60	52.60	84.48
ExpAttn	**100.00**	98.20	6.20	97.00	69.60	0.20	91.95	94.70	**100.00**	34.12	78.73	42.60	35.40	65.28
SnapKV	37.80	2.40	2.40	11.60	1.60	0.00	10.40	10.50	19.56	34.82	86.33	16.80	18.20	19.42
SnapKV-QA	99.60	96.60	2.40	63.20	22.00	40.40	24.85	97.60	91.44	35.48	75.33	**77.40**	57.80	60.32
KVzip	**100.00**	99.40	**99.80**	88.80	97.60	98.00	82.25	**98.60**	**100.00**	72.24	83.53	71.40	56.80	**88.34**
KVzipFast	**100.00**	**99.80**	**99.80**	79.80	**98.60**	**99.00**	59.50	95.50	**100.00**	58.46	83.67	67.40	**58.20**	84.59
AM-Fast	97.60	98.60	91.40	**98.60**	98.20	95.00	**96.05**	97.20	93.84	54.00	79.60	74.20	57.40	87.05
AM	96.80	99.40	93.40	96.40	98.40	95.20	95.70	95.84	94.28	52.88	80.60	75.00	57.80	87.05
4x compression														
LCLM 0.6b-4b	99.40	99.80	95.60	98.60	98.40	91.60	98.80	98.95	94.64	82.04	90.40	76.20	59.00	91.03
ExpAttn	**100.00**	**100.00**	60.60	**99.20**	98.60	48.60	97.65	99.80	**100.00**	77.74	85.13	60.80	52.00	83.09
SnapKV	67.00	23.60	2.40	18.80	3.80	1.00	18.20	17.90	47.52	60.22	**92.87**	23.00	27.00	31.02
SnapKV-QA	**100.00**	99.80	2.40	99.00	46.80	87.40	66.75	99.95	99.28	66.00	86.27	77.00	59.40	76.16
KVzip	**100.00**	**100.00**	**100.00**	98.40	**100.00**	99.80	**99.20**	**100.00**	**100.00**	**91.86**	88.40	77.20	57.80	**93.28**
KVzipFast	**100.00**	**100.00**	**100.00**	98.40	**100.00**	**100.00**	98.75	99.95	**100.00**	91.84	86.27	75.60	**60.20**	93.15
AM-Fast	99.40	**100.00**	98.00	98.60	99.40	99.20	96.40	98.90	96.96	78.62	86.47	**79.20**	59.60	91.60
AM	99.60	**100.00**	98.40	98.40	99.40	98.20	95.95	98.05	97.40	78.02	87.33	78.60	59.80	91.47

Table 9: RULER per-task at 16k context.

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	vt	cwe	fwe	qa1	qa2	AVG
Qwen 4B Instruct	100.00	100.00	100.00	99.60	95.40	99.80	98.95	99.95	99.52	88.04	98.93	79.60	58.80	93.74
16x compression														
LCLM 0.6b-4b	95.80	88.00	57.40	75.60	62.20	30.00	70.70	73.05	77.32	24.78	**98.73**	59.40	43.80	**65.91**
ExpAttn	**100.00**	83.00	1.60	86.60	1.60	0.00	**78.30**	76.35	99.32	8.52	71.07	28.40	24.00	50.67
SnapKV	41.80	4.20	2.40	11.20	2.00	0.20	10.00	10.85	17.60	16.64	92.20	12.40	16.00	18.27
SnapKV-QA	99.80	**99.40**	2.40	**88.40**	11.60	27.00	30.75	**99.15**	93.68	22.22	98.33	**74.80**	**58.00**	61.96
KVzip	**100.00**	92.20	**91.60**	55.20	37.60	18.40	35.70	53.10	**99.96**	3.18	88.87	41.20	36.60	57.97
KVzipFast	**100.00**	53.60	42.00	38.40	**64.40**	**44.00**	19.05	25.40	99.76	3.58	79.00	23.40	29.60	47.86
AM-Fast	52.05	33.00	0.20	28.40	20.60	0.00	19.40	23.70	54.92	**55.44**	94.67	58.60	42.20	37.17
AM	61.00	31.20	1.20	38.60	24.80	0.00	32.30	34.50	66.84	52.28	93.33	63.00	47.20	42.02
8x compression														
LCLM 0.6b-4b	99.20	95.00	72.60	88.80	94.20	69.20	88.10	90.10	89.04	**67.34**	97.73	68.40	52.40	82.47
ExpAttn	**100.00**	97.80	6.80	97.00	58.00	0.60	**93.65**	96.90	99.92	30.14	95.73	39.00	34.80	65.41
SnapKV	63.60	11.00	2.40	12.40	2.60	0.60	11.70	12.65	47.72	34.86	97.40	14.40	18.80	25.39
SnapKV-QA	**100.00**	**100.00**	2.40	**99.20**	27.40	67.60	65.70	**99.90**	99.00	40.72	**99.53**	**75.40**	**60.40**	72.10
KVzip	**100.00**	99.80	99.60	90.40	92.60	93.00	83.95	98.65	**99.96**	66.62	98.67	71.40	57.80	**88.65**
KVzipFast	**100.00**	**100.00**	**100.00**	76.80	**95.60**	**96.00**	65.85	96.95	**99.96**	58.10	97.07	68.40	53.80	85.27
AM-Fast	51.20	60.20	10.40	58.20	75.40	13.40	56.45	58.20	55.24	67.10	96.20	74.00	52.80	56.06
AM	53.80	62.20	13.00	70.60	78.60	17.80	63.25	66.80	60.84	65.30	95.20	71.80	52.20	59.34
4x compression														
LCLM 0.6b-4b	99.60	99.60	94.80	99.20	98.20	90.40	98.15	98.80	91.40	67.34	99.53	75.60	56.80	89.96
ExpAttn	**100.00**	99.80	57.40	99.00	98.00	34.60	98.95	99.60	**99.96**	73.42	98.47	61.20	52.20	82.51
SnapKV	80.60	46.40	2.40	19.20	4.00	2.20	14.95	18.30	78.16	59.58	99.13	22.00	25.40	36.33
SnapKV-QA	**100.00**	99.80	6.20	**99.80**	59.00	94.60	93.45	**99.95**	99.88	66.60	**99.73**	75.40	59.40	81.06
KVzip	**100.00**	**100.00**	**100.00**	99.20	**99.80**	**100.00**	**99.60**	**99.95**	**99.96**	85.00	98.80	77.60	**61.20**	**93.93**
KVzipFast	**100.00**	**100.00**	**100.00**	98.60	99.40	99.80	99.20	**99.95**	**99.96**	**85.46**	98.93	76.00	60.00	93.64
AM-Fast	60.00	94.40	80.20	93.00	91.60	91.20	91.60	92.45	63.56	79.80	96.46	**78.80**	57.80	82.37
AM	54.40	94.60	78.60	95.00	94.00	92.00	91.85	94.10	63.16	79.08	95.40	77.60	56.00	81.98

Table 10: LongBench English (en16) per-subtask.

Model	narr	qasp	mfq	hpot	2wiki	musq	govr	qmsm	mnews	trec	triv	samsm	pcnt	pr	lcc	rbp	AVG
Qwen 4B Instruct	30.18	43.60	46.63	52.05	36.18	17.26	30.32	22.91	23.79	75.50	85.14	40.03	3.50	94.91	65.00	56.36	45.21
16x compression																	
LCLM 0.6b-4b	25.57	39.80	**47.98**	47.96	**38.44**	21.17	28.56	22.42	23.48	45.00	**89.43**	32.50	3.00	81.50	37.47	41.05	39.08
ExpAttn	17.67	23.16	28.88	33.76	23.45	8.78	23.06	19.86	19.54	36.00	81.02	31.80	7.36	30.32	28.24	**58.14**	29.44
SnapKV	12.97	13.68	22.14	20.93	16.13	5.61	21.61	17.66	14.70	32.75	85.87	36.57	3.25	17.22	**53.42**	56.78	26.96
SnapKV-QA	**28.36**	28.81	40.37	**53.20**	29.33	17.33	21.57	22.78	14.06	35.00	83.92	37.66	2.37	**92.50**	**53.42**	55.60	38.52
KVzip	16.08	16.38	32.53	10.30	8.68	4.93	24.43	21.38	17.49	43.00	78.22	31.34	6.50	14.00	31.96	56.53	25.86
KVzipFast	11.96	18.26	24.60	23.57	18.08	4.31	20.15	20.48	15.03	34.00	72.78	34.22	**7.44**	8.25	46.89	52.12	25.76
AM-Fast	26.31	**42.64**	45.22	48.31	30.57	24.17	31.59	23.90	**24.14**	**74.50**	85.84	**40.37**	5.11	47.04	48.26	49.81	**40.49**
AM	24.83	41.04	45.28	45.85	30.76	**24.84**	**31.62**	**24.02**	24.08	74.00	86.17	39.60	5.08	47.38	48.79	50.76	40.26
8x compression																	
LCLM 0.6b-4b	**28.60**	43.92	**51.54**	**54.02**	**45.36**	**28.05**	29.05	22.85	23.56	62.00	**86.39**	33.17	5.45	71.50	43.57	46.63	42.23
ExpAttn	23.27	32.95	36.13	40.83	31.71	13.74	27.81	21.46	22.15	65.75	84.11	33.95	5.03	77.22	36.30	57.69	38.13
SnapKV	18.27	21.07	25.80	29.97	17.90	6.88	24.52	18.96	18.22	47.50	85.01	38.58	3.27	46.62	**58.34**	55.63	32.28
SnapKV-QA	28.50	34.45	45.16	53.20	34.12	18.53	25.11	22.86	17.56	51.50	85.33	38.70	2.19	**94.20**	**58.34**	58.00	41.73
KVzip	25.80	41.38	49.59	46.31	27.10	15.05	28.73	22.71	23.07	69.50	83.99	37.62	3.79	68.67	49.10	**59.16**	40.72
KVzipFast	20.72	35.80	47.81	45.06	29.11	15.01	27.96	23.00	22.73	66.00	83.63	36.81	3.33	41.96	54.86	56.91	38.17
AM-Fast	28.23	44.29	49.74	52.61	36.39	25.60	30.70	**23.19**	**24.02**	73.50	85.27	**39.87**	**5.67**	86.50	54.38	52.05	**44.50**
AM	25.12	**45.13**	47.57	52.98	36.38	24.04	**30.99**	23.07	23.99	**74.00**	85.51	39.39	4.05	87.50	55.42	52.34	44.22
4x compression																	
LCLM 0.6b-4b	**30.74**	**46.76**	**49.76**	52.90	**42.75**	**28.61**	30.12	22.89	23.83	68.50	**87.90**	34.10	**6.67**	**100.00**	52.01	59.17	**46.04**
ExpAttn	27.13	38.72	44.59	49.87	36.76	17.10	29.42	22.63	23.12	71.00	84.55	35.97	3.67	87.29	42.13	57.28	41.95
SnapKV	21.88	30.57	30.90	37.39	26.71	11.58	27.18	20.75	20.71	59.00	84.86	**39.63**	3.00	78.86	**61.53**	56.36	38.18
SnapKV-QA	29.55	39.75	45.91	53.23	37.11	17.92	27.85	23.22	20.13	66.50	84.94	38.60	2.96	93.54	**61.53**	57.14	43.74
KVzip	28.38	45.62	48.61	**54.37**	40.26	18.45	29.86	23.03	23.86	73.00	85.37	39.15	2.28	93.92	59.70	58.92	45.30
KVzipFast	28.61	44.93	48.54	51.89	36.63	18.63	30.24	22.98	23.67	73.00	84.87	38.86	4.55	94.71	60.82	**59.66**	45.16
AM-Fast	26.27	44.45	49.18	53.99	37.78	22.66	**30.74**	22.83	23.89	**74.50**	83.87	39.06	5.50	99.00	58.13	55.85	45.48
AM	26.30	44.98	47.17	54.16	41.41	24.10	30.57	**23.40**	**24.05**	**74.50**	85.19	39.39	3.11	99.17	58.30	56.67	45.78

Table 11: LongBench Chinese (cn5) per-subtask.

Model	mfq	dure	vcsum	lsht	pr	AVG
Qwen 4B Instruct	61.60	25.45	11.38	43.50	91.23	46.63
16x compression						
LCLM 0.6b-4b	46.83	23.63	10.55	18.67	59.00	31.74
ExpAttn	25.52	16.37	11.22	17.75	31.67	20.51
SnapKV	20.36	16.77	10.44	35.00	8.93	18.30
SnapKV-QA	**48.61**	21.46	10.44	**41.50**	**89.28**	**42.26**
KVzip	25.95	16.30	12.50	23.75	7.12	17.12
KVzipFast	30.95	17.04	10.92	19.00	7.47	17.08
AM-Fast	48.55	27.13	13.62	39.50	28.76	31.51
AM	48.25	**28.02**	**13.75**	39.25	30.00	31.85
8x compression						
LCLM 0.6b-4b	51.27	24.83	10.79	25.65	60.50	34.61
ExpAttn	32.75	18.70	11.02	21.00	69.32	30.56
SnapKV	26.07	19.01	10.81	**43.25**	26.78	25.18
SnapKV-QA	58.07	23.37	10.81	42.50	**92.29**	**45.41**
KVzip	58.26	25.30	11.98	40.75	44.17	36.09
KVzipFast	57.35	24.84	12.77	24.00	35.60	30.91
AM-Fast	**59.11**	**28.47**	13.08	42.75	64.21	41.52
AM	58.88	28.40	**13.18**	42.75	62.34	41.11
4x compression						
LCLM 0.6b-4b	45.52	26.07	11.00	25.00	97.00	40.92
ExpAttn	54.32	23.86	11.63	34.25	83.81	41.57
SnapKV	33.48	20.48	11.07	**44.75**	61.32	34.22
SnapKV-QA	60.69	24.33	11.07	43.50	91.34	46.19
KVzip	62.35	26.90	11.86	43.75	92.37	47.45
KVzipFast	**63.62**	27.23	11.95	44.50	96.02	**48.66**
AM-Fast	58.73	28.45	**12.97**	43.75	**98.03**	48.39
AM	59.04	**29.09**	12.61	43.25	96.37	48.07

G.3 Continual Pretraining LR sweep

Table 12: Continual Pretraining LR sweep summary (16 × ratio compression, bidirectional mask, mlp adapter, $O = 0$).

Model	RULER 4k	RULER 8k	RULER 16k	LB en16	LB cn5	LongHealth5	GSM8K
1e-6	62.33	58.75	56.65	33.29	22.88	60.25	66.94
3e-6	70.73	65.41	63.55	32.89	22.09	62.50	72.25
1e-5	**71.62**	**66.83**	**63.63**	**33.52**	**23.94**	**62.75**	**74.53**

Table 13: Continual Pretraining LR sweep – RULER per-task

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	vt	cwe	fwe	qa1	qa2	AVG
Context length 4096														
1e-6	97.40	66.40	31.40	62.00	47.20	17.40	71.70	62.10	84.88	**69.16**	88.00	62.80	49.80	62.33
3e-6	**98.80**	**90.60**	**42.60**	**81.40**	54.40	21.80	84.50	81.15	**91.48**	65.72	88.07	65.20	**53.80**	70.73
1e-5	97.80	83.00	39.40	**81.40**	**64.60**	**29.80**	**85.55**	**81.30**	90.96	68.24	**91.07**	**66.20**	51.80	**71.62**
Context length 8192														
1e-6	96.20	77.80	28.00	60.80	40.00	12.40	60.40	57.95	84.84	**38.70**	**96.40**	**62.20**	48.00	58.75
3e-6	99.00	88.00	40.80	**75.60**	44.20	14.40	**78.55**	**76.15**	**91.40**	34.18	95.47	61.60	**51.00**	65.41
1e-5	**99.20**	**89.80**	**43.60**	73.60	**58.60**	**19.60**	76.35	74.10	91.04	36.74	95.33	61.80	49.00	**66.83**
Context length 16384														
1e-6	98.60	78.00	32.00	64.60	38.40	8.00	60.75	58.50	81.96	**12.66**	**99.53**	56.60	46.80	56.65
3e-6	**99.40**	**90.20**	**46.60**	74.00	38.00	11.20	77.40	**77.15**	**91.24**	11.82	**99.53**	59.00	50.60	63.55
1e-5	98.80	84.60	42.00	**75.60**	**49.80**	**12.80**	**78.40**	74.60	89.12	9.88	99.40	**61.20**	**51.00**	**63.63**

Table 14: Continual Pretraining LR sweep – LongBench English (en16) per-subtask.

Model	narr	qasp	mfq	hpot	2wiki	musq	govr	qmsm	mnews	trec	triv	samsm	pcnt	pr	lcc	rbp	AVG
1e-6	22.98	37.30	**42.23**	37.19	**31.51**	**14.42**	28.61	22.15	22.72	**31.00**	85.35	**33.31**	1.50	**50.00**	35.85	36.53	33.29
3e-6	**23.67**	38.81	41.70	**38.09**	29.49	13.43	**29.25**	**22.98**	**22.98**	30.00	86.44	32.39	0.50	41.50	36.19	38.82	32.89
1e-5	23.32	**40.05**	42.18	35.36	31.18	12.64	29.12	22.95	22.85	29.50	**88.68**	32.28	**4.29**	44.00	**38.07**	**39.92**	**33.52**

Table 15: Continual Pretraining LR sweep – LongBench Chinese (cn5) per-subtask.

Model	mfq	dure	vcsum	lsht	pr	AVG
1e-6	38.79	24.13	**11.00**	**21.00**	19.50	22.88
3e-6	36.39	25.97	10.85	17.75	19.50	22.09
1e-5	**39.65**	**27.07**	10.99	19.50	**22.50**	**23.94**

G.4 Architecture comparison

Table 16: Architecture comparison (16 × compression).

Model	RULER 4k	RULER 8k	RULER 16k	LB en16	LB cn5	LongHealth5	GSM8K
Causal-MLP-O0	**75.06**	**70.23**	**65.91**	**39.08**	**31.74**	**67.50**	**81.05**
Bidirectional-MLP-O0	71.62	66.83	63.63	33.52	23.94	62.75	74.53
Bidirectional-ATTN-MLP-O256	71.91	68.11	64.96	36.28	26.54	64.00	73.77
Bidirectional-MLP-O256	62.03	58.66	58.24	36.97	26.63	63.25	74.98

Table 17: Architecture comparison – RULER per-task

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	vt	cwe	fwe	qa1	qa2	AVG
Context length 4096														
Causal-MLP-O0	96.40	**89.40**	**54.60**	**85.00**	**80.80**	**48.60**	72.45	80.20	83.92	**80.54**	88.27	66.00	49.60	**75.06**
Bidirectional-MLP-O0	97.80	83.00	39.40	81.40	64.60	29.80	85.55	81.30	90.96	68.24	**91.07**	**66.20**	**51.80**	71.62
Bidirectional-ATTN-MLP-O256	**99.80**	80.40	50.20	78.60	62.20	26.20	**85.70**	**81.35**	95.00	80.00	87.00	61.80	46.60	71.91
Bidirectional-MLP-O256	99.00	60.60	21.60	61.80	55.40	17.60	69.90	61.60	**97.12**	57.66	87.27	66.00	50.80	62.03
Context length 8192														
Causal-MLP-O0	96.00	**90.20**	55.20	**83.40**	**65.40**	**36.60**	70.25	**77.65**	80.52	**55.44**	**95.67**	61.80	44.80	**70.23**
Bidirectional-MLP-O0	99.20	89.80	43.60	73.60	58.60	19.60	76.35	74.10	91.04	36.74	95.33	61.80	**49.00**	66.83
Bidirectional-ATTN-MLP-O256	**100.00**	86.00	**58.60**	72.00	52.60	25.60	**76.75**	74.65	96.12	43.78	91.93	**63.20**	44.20	68.11
Bidirectional-MLP-O256	98.80	67.20	25.40	61.20	47.20	15.60	60.55	54.15	**96.68**	28.76	95.60	62.80	48.60	58.66
Context length 16384														
Causal-MLP-O0	95.80	88.00	**57.40**	75.60	**62.20**	**30.00**	70.70	73.05	77.32	**24.78**	98.73	59.40	43.80	**65.91**
Bidirectional-MLP-O0	98.80	84.60	42.00	75.60	49.80	12.80	**78.40**	74.60	89.12	9.88	99.40	**61.20**	**51.00**	63.63
Bidirectional-ATTN-MLP-O256	**100.00**	**89.00**	57.00	**78.20**	42.40	17.60	78.25	**77.40**	93.80	10.12	**99.47**	56.20	45.00	64.96
Bidirectional-MLP-O256	97.80	75.40	28.60	68.20	36.80	12.80	62.95	60.65	**96.00**	12.48	99.20	59.80	46.40	58.24

Table 18: Architecture comparison – LongBench English (en16) per-subtask.

Model	narr	qasp	mfq	hpot	2wiki	musq	govr	qmsm	mnews	trec	triv	samsm	pcnt	pr	lcc	rbp	AVG
Causal-MLP-O0	25.57	39.80	**47.98**	47.96	**38.44**	**21.17**	28.56	22.42	**23.48**	**45.00**	**89.43**	**32.50**	3.00	**81.50**	37.47	41.05	**39.08**
Bidirectional-MLP-O0	23.32	**40.05**	42.18	35.36	31.18	12.64	**29.12**	**22.95**	22.85	29.50	88.68	32.28	4.29	44.00	38.07	39.92	33.52
Bidirectional-ATTN-MLP-O256	**27.45**	38.87	42.56	**52.31**	36.98	20.22	27.98	22.44	23.01	27.50	87.23	32.16	**6.00**	51.50	41.22	43.00	36.28
Bidirectional-MLP-O256	23.20	39.67	42.77	46.56	31.94	16.41	28.64	22.50	22.65	40.00	86.67	31.80	4.50	67.50	**41.76**	**44.89**	36.97

Table 19: Architecture comparison – LongBench Chinese (cn5) per-subtask.

Model	mfq	dure	vcsum	lsht	pr	AVG
Causal-MLP-O0	**46.83**	23.63	10.55	18.67	**59.00**	**31.74**
Bidirectional-MLP-O0	39.65	**27.07**	10.99	19.50	22.50	23.94
Bidirectional-ATTN-MLP-O256	44.35	23.26	10.75	18.83	35.50	26.54
Bidirectional-MLP-O256	38.06	24.02	**11.05**	**22.50**	37.50	26.63

Table 20: Mask-type ablation (Mean pooling, W=1024, LR = 1e-5).

Mask	RULER 4k	RULER 8k	RULER 16k	LB en16	LB cn5	LongHealth5	GSM8K
Bidirectional	71.62	66.83	63.63	33.52	23.94	62.75	74.53
Causal	**75.06**	**70.23**	**65.91**	**39.08**	**31.74**	**67.50**	**81.05**

Table 21: Adapter & overlapping-context ablation (Mean pooling, W=1024).

Variant	RULER 4K	RULER 8K	RULER 16K	LB en16	LB cn5	LongHealth5	GSM8K
MLP, O=0, bidir (LR 1e-5)	71.62	66.83	63.63	33.52	23.94	62.75	74.53
MLP, O=0, causal (LR 1e-5)	**75.06**	**70.23**	**65.91**	**39.08**	**31.74**	**67.50**	**81.05**
MLP, O=256, bidir (LR 3e-5)	62.03	58.66	58.24	36.97	26.63	63.25	74.98
Attn-MLP, O=256, bidir (LR 3e-5)	71.91	68.11	64.96	36.28	26.54	64.00	73.77

Table 22: Window-size ablation (mean pooling, LR = 1e-6). W=16 and W=256 use causal attention; W=1024 uses bidirectional attention. We do not run W=1024 with causal attention at LR = 1e-6. However, Table 20 shows that causal masking is strictly better than bidirectional masking, and W=1024 (bidirectional) already outperforms W=256 (causal) on RULER tasks while also achieving lower pretraining loss. We therefore conclude that W=1024 is the better setting.

Model	RULER 4k	RULER 8k	RULER 16k	LB en16	LB cn5	LongHealth5	GSM8K
Mean W16 (causal)	24.85	23.87	22.54	29.62	22.62	56.50	46.55
Mean W256 (causal)	58.37	54.39	50.18	**34.15**	**25.97**	**62.25**	**76.80**
Mean W1024 (bidir)	**62.33**	**58.75**	**56.65**	33.29	22.88	60.25	66.94

Table 23: Encoder-representation ablation (CPT, 0.6B→4B). EOS pooling, causal, LR = 1e-6. RULER columns use the official RULER `string_match` scorer.

Encoder init	RULER 4k	RULER 8k	RULER 16k	LB en16	LB cn5	LongHealth5	GSM8K
LLM (Instruct)	38.29	34.39	33.50	30.48	22.73	**58.25**	54.51
Embed (Qwen3-Emb)	**42.00**	**38.76**	**38.12**	**31.72**	**25.32**	58.00	**59.36**

G.5 NIAH Agent results

Table 24: RULER NIAH (retrieval) per-task at 4k context. Rows grouped by compression ratio. The agent at 16x is the multi-round expand-or-answer loop; Δ *agent* is the cell-wise gain over the 16x baseline. 8x and 4x are uncompressed-baseline numbers (no agent run at those ratios).

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	AVG
Qwen 4B Instruct (full KV)	100.00	100.00	99.80	100.00	99.20	99.40	93.30	100.00	98.96
16x compression									
LCLM 0.6b-4b	96.80	90.00	55.20	86.40	82.20	49.00	74.10	81.45	76.89
LCLM 0.6b-4b **agent**	**98.20**	**97.00**	**97.60**	**97.00**	**86.60**	**89.60**	**99.55**	**86.35**	**93.99**
Δ *agent*	*+1.40*	*+7.00*	*+42.40*	*+10.60*	*+4.40*	*+40.60*	*+25.45*	*+4.90*	*+17.10*
8x compression									
LCLM 0.6b-4b	98.80	95.40	64.40	93.00	98.20	82.80	92.40	91.55	89.57
4x compression									
LCLM 0.6b-4b	99.40	99.60	93.60	99.00	99.40	94.20	98.25	99.20	97.83

Table 25: RULER NIAH (retrieval) per-task at 8k context. Same conventions as Table 24.

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	AVG
Qwen 4B Instruct (full KV)	100.00	99.80	99.80	99.00	98.80	98.20	94.85	99.95	98.80
16x compression									
LCLM 0.6b-4b	96.00	91.20	55.80	84.00	66.80	37.60	70.45	78.20	72.51
LCLM 0.6b-4b **agent**	**99.20**	**97.40**	**97.00**	**98.20**	**74.60**	**83.00**	**97.10**	**93.20**	**92.46**
Δ *agent*	*+3.20*	*+6.20*	*+41.20*	*+14.20*	*+7.80*	*+45.40*	*+26.65*	*+15.00*	*+19.95*
8x compression									
LCLM 0.6b-4b	99.00	96.40	71.60	91.00	97.20	76.20	90.60	91.30	89.16
4x compression									
LCLM 0.6b-4b	99.80	99.80	95.60	98.60	98.40	91.80	98.80	98.95	97.72

Table 26: RULER NIAH (retrieval) per-task at 16k context. Same conventions as Table 24.

Model	ns1	ns2	ns3	nm1	nm2	nm3	nmv	nmq	AVG
Qwen 4B Instruct (full KV)	100.00	100.00	100.00	99.60	91.40	96.00	96.85	99.85	97.96
16x compression									
LCLM 0.6b-4b	96.20	89.80	58.00	77.00	63.60	31.60	71.80	73.40	70.18
LCLM 0.6b-4b **agent**	**98.00**	**98.00**	**98.00**	**96.00**	**65.20**	**72.00**	**96.35**	**95.75**	**89.91**
Δ *agent*	*+1.80*	*+8.20*	*+40.00*	*+19.00*	*+1.60*	*+40.40*	*+24.55*	*+22.35*	*+19.74*
8x compression									
LCLM 0.6b-4b	99.60	95.20	72.60	89.00	94.40	72.00	88.10	90.10	87.63
4x compression									
LCLM 0.6b-4b	100.00	99.60	94.80	99.20	98.80	90.40	98.15	98.80	97.47

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In Section 1, we reference the sections where each claim is evidenced.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss future work in Section 8.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe all models and data in Section 4, Appendix C, and Appendix F. We will open source, all code, data mixes and models with the paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The data and models are too large to upload in the supplementary material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify training data in Section 4, and Appendix C. We specify hyperparameters in Appendix F. We specify evaluation details in Table 5. We describe our design decisions in Section 5 and Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not provide error bars for our pretraining from scratch experiments but do test at larger scales in Appendix G. We perform many large scale ablations in Appendix G to reduce noise in our results. For our TTFT measurements we take the mean of 30 timings on the same H200 GPU.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix F.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: this paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Appendix A.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our models are trained on data already widely available on the internet and distilled from models widely provided, hence we believe the additional risk posed by our models and data to be negligible. We discuss our wider impact in Appendix A.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use existing datasets, models, benchmarks, and codebases, and cite their original papers or release pages throughout the paper. The Qwen model checkpoints used for initialization and relabeling are released under the Apache-2.0 license. We report dataset licenses and terms of use in Table 1 and Table 2.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We introduce a suite of LCLMs checkpoints spanning compression ratios of $4\times$, $8\times$, and $16\times$, together with our training code and the curated continual pretraining, SFT, and auxiliary reconstruction data mixtures. Architectural details are documented in Section 5 and Appendix D; the training recipe and hyperparameters are documented in Section 4 and Appendix F (Table 4); data composition, sources, and licenses are documented in Table 1 and Table 2; and evaluation protocols are documented in Table 5. Upon acceptance, models, code, and data mixtures will be released with documentation (model cards, dataset cards, and training/evaluation scripts) under permissive licenses consistent with the upstream sources listed in Tables 1 and 2.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.