LooComp: Leverage Leave-One-Out Strategy to Encoder-only Transformer for Efficient Query-aware Context Compression

Anonymous ACL submission

Abstract

Efficient context compressing is crucial for improving the accuracy and scalability of question answering. For the efficiency of Retrieval Augmented Generation, attached context should be fast delivered, compact but precise to ensure clue sufficient and budget-friendly LLM reader cost. We propose a margin-based framework for query-driven context pruning, which identifies sentences that are critical for answering a query by measuring changes in clue richness when they are omitted. The model is trained with a composite ranking loss that enforces large margins for critical sentences while keeping non-critical ones near neutral. Built on a recent lightweight encoder-only Transformer, our approach achieves competitive exact match and F1 scores while requiring substantially less memory and high-throughput inference than decoder-based baselines. In addition to efficiency, our method yields effective compression ratios without degrading downstream answer performance, demonstrating its potential as a lightweight and practical alternative for retrieval-augmented tasks.

1 Introduction

Retrieval-augmented generation (RAG) has emerged as a powerful paradigm for enhancing large language models (LLMs) with external knowledge, significantly improving factual accuracy and reducing hallucinations (Lewis et al., 2020; Ram et al., 2023). However, as RAG systems scale to handle more complex queries, a fundamental issue arises: retrieving more documents improves coverage of relevant information but also introduces computational overhead and potential distraction that can degrade performance (Shi et al., 2023; Liu et al., 2023).

Context compression has emerged as a potential solution to this challenge, enabling RAG systems to retain essential information while reducing computational costs (Li et al., 2023; Wingate et al., 2022). Current approaches fall into 2 categories: abstractive methods that generate condensed summaries (Yoon et al., 2024; Xu et al., 2024; Li et al., 2024) and extractive methods that select relevant text segments (Jiang et al., 2023). While abstractive compression can achieve high compression ratios, the token-by-token generation process introduces significant latency overhead. This often exceeds the time saved from reduced context length. Extractive methods offer faster compression but typically rely on rigid selection criteria that fail to adapt to query complexity or capture the inter-dependencies between sentences.

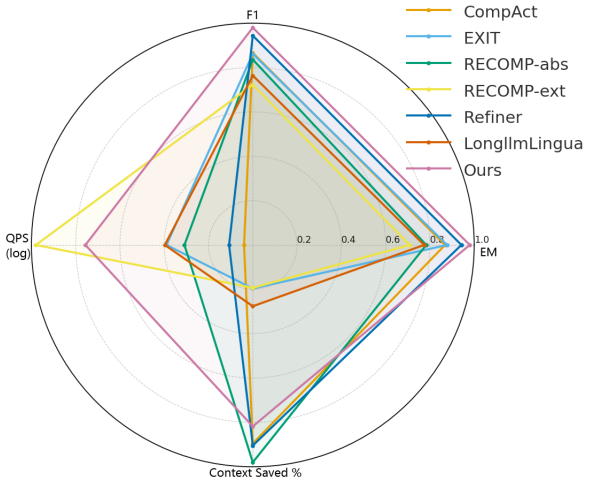

Figure 1: Answering perfomance (EM, F1) and compression efficiency (QPS, Saved %) across compressors. Questions Per Second (QPS) is from compression latency; Context Saved is 100% – Compression ratio.

Recent work has tried to address some of these limitations. EXIT (Hwang et al., 2024) introduces context-aware extractive compression that uses full-document context to make sentence-level decisions in parallel with a goal to reduce latency. However, EXIT retains two drawbacks: it relies on decoder-based LLMs, creating unnecessary computational overhead for what is essentially a classification task; and it inflates inputs by repeating the inspected sentences. Meanwhile, comparative

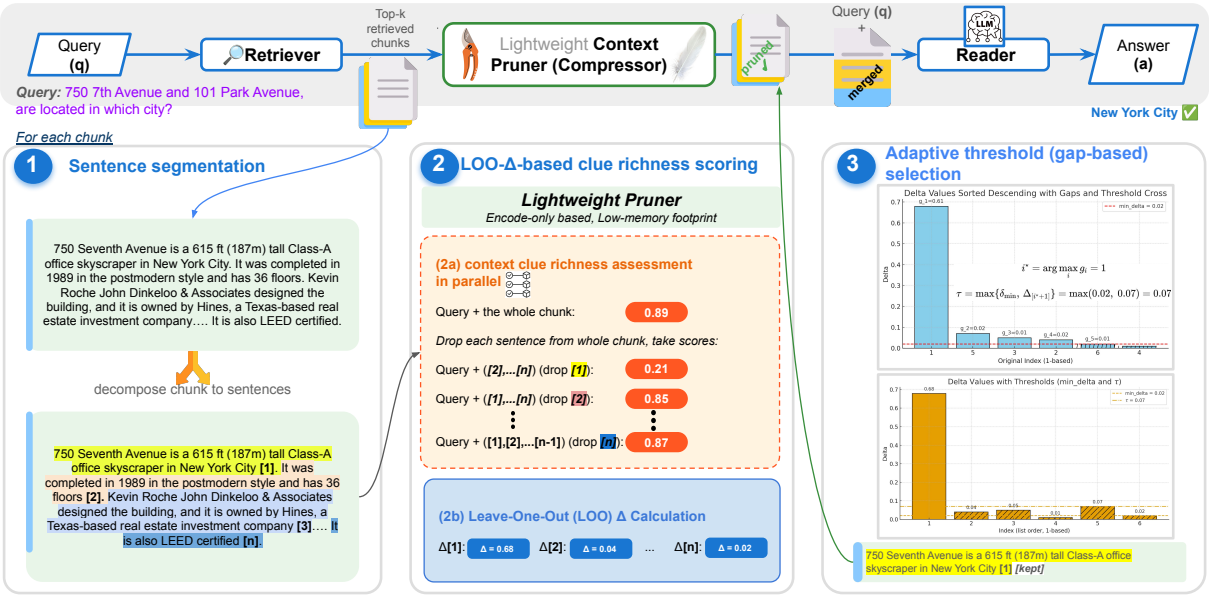

Figure 2: Our Compression Pipeline Overview

studies show that encoder-only models can match or surpass decoder-only models on various natural language understanding (NLU) tasks (Benayas et al., 2025; Qorib et al., 2024; Nielsen et al., 2024).

In this work, we address these limitations by revisiting the fundamental design choices in sentence-level compression. Our key insight is that encoder-only models are sufficient and more efficient for relevance classification; and thus, leave-one-out delta (LOO-Δ) scoring was created to provide a more principled measure of sentence importance than binary classification. Specifically, we propose using a lightweight encoder-only model based on ModernBERT (Warner et al., 2024) (≈395M parameters) to score query-context pairs, then measuring each sentence's contribution as the drop in answerability when that sentence is removed. We also introduce a gap-based selection rule that identifies natural breakpoints in the relevance score distribution, automatically adapting the compression rate to each query without manual tuning. Our main contributions are as follows:

1. We propose an encoder-only based model and demonstrate that it is sufficient and more efficient than decoder LLMs for sentence-level compression in RAG.
2. We introduce LOO-Δ scoring as a principled measure of sentence importance, based on marginal contribution to answerability. The process is parallelized to accelerate computation even for long contexts. We further propose an adaptive gap-based strategy to select valuable sentences to keep.
3. We generally achieve outstanding answer accuracy while delivering fast compression and substantially compact outputs on standard QA benchmarks across multiple LLM readers (e.g. Table 2 or Fig. 1)

2 Related Work

2.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Ram et al., 2023; Gao et al., 2023; Khattab et al., 2022; Trivedi et al., 2022a) enhances large language models (LLMs) by incorporating information retrieved from external knowledge sources rather than relying solely on parametric knowledge. Early RAG systems (Lewis et al., 2020; Izacard and Grave, 2020) typically used a single-pass retrieval strategy that improved factual accuracy but often missed documents needed for complex, multi-source reasoning. More recent approaches introduce iterative, multi-hop, and recursive retrieval (Khattab et al., 2022; Trivedi et al., 2022a; Shao et al., 2023), which refine search queries or gather additional documents during generation to access dispersed evidence. While these methods improve recall, they increase computational cost and latency, and they often expand the token budget with irrelevant details, which can reduce precision (Liu et al., 2023; Xia et al., 2024). Consequently, growing attention is given to token-reduction and document-compression techniques designed to retain only essential information, aim-

ing to balance comprehensive evidence with low-latency, high-quality generation in RAG pipelines.

2.2 Context Compression in RAG

Context compression plays a crucial role in enhancing RAG efficiency by minimizing the token footprint of retrieved documents before inputting into LLMs. Current methods of context compression can be classified into two main categories: soft compression and hard compression.

Soft compression works at the embedding level to shorten token representations (Mu et al., 2023; Wingate et al., 2022; Ge et al., 2023). Although these methods maintain semantic fidelity and allow for precise control over compression rates, they require extensive fine-tuning or architectural modification. This makes these methods impractical for LLMs that are closed-source LLMs.

Hard compression serves as a filtering mechanism that shortens text by selectively keeping crucial natural language tokens or rephrasing content, primarily aimed at preserving information relevant to the task. This approach encompasses abstractive and extractive methods. Abstractive techniques utilize generative models to summarize retrieved passages (Zhao et al., 2020; Xu et al., 2024; Yoon et al., 2024), facilitating significant token reduction and precise token reduction rates. For instance, RECOMP-Abst (Xu et al., 2024) utilizes a T5-based model to summarize context, which has specific training requirements related to the dataset. Similarly, CompAct(Yoon et al., 2024), Refiner (Li et al., 2024) increase quality and compactness using bigger decoder-only models but also increase resource demands and slow inference significantly.

On the other hand, extractive methods have been proposed such as RECOMP-Extr (Xu et al., 2024) and the LLMLingua family (Li et al., 2023; Jiang et al., 2023; Pan et al., 2024). These methods focus on selecting salient sentences or tokens directly from the retrieved documents, preserving original text and reducing hallucination. However, their query-agnostic selection strategies often disrupt semantic coherence or omit critical information. LongLLMLingua (Jiang et al., 2023) addressed the limitation by introducing query-awareness and configurable compression rates. However, these methods still struggle to achieve a balance between coverage and efficiency. Recently, EXIT (Hwang et al., 2024) employs an extractive pipeline to prune context based on query relevance with sentences. While achieving parallel processing, EXIT still incurs high computations for the compression process due to its decoder-only backbone Gemma-2B (Team et al., 2024).

To avoid the slow inference and high memory footprint of hard compression, we formulate query-driven context pruning as extractive sentence selection. We hypothesize this subtask does not require billion-parameter LLMs (e.g., LongLLMLingua (Jiang et al., 2023), Refiner (Li et al., 2024), EXIT (Hwang et al., 2024)). Accordingly, we use a modest encoder-only backbone and a novel clue-richness score to filter sentences, preserving critical clues and faithfulness to retrieved evidence while remaining efficient and scalable.

3 Methodology

Our approach frames query-driven context pruning as an extractive sentence selection problem, where the goal is to identify the subset of sentences that are critical for answering a given query. Unlike generative compression, ours is extractive and preserves the original text, ensuring faithfulness to the retrieved evidence. The core mechanism relies on computing a delta score—the change in answerability when a sentence is omitted relative to the full chunk. These deltas are computed independently, which enables parallelization, ensures inference remains efficient even for long documents. To capture broader semantic dependencies, each delta is derived with respect to the full context, allowing the model to leverage global cues while assessing partial evidence. The framework illustrated at Fig 2, is built upon ModernBERT, where flash-attention support reduces more memory consumption and enhance speed, making it scalable and practical for real-world application.

Problem Formulation In RAG, a language model M is tasked with producing an output y given an input q (e.g., a question, instruction) along with a set of k retrieved passages (i.e., chunks, documents) $D_k = \{d_1, d_2, \ldots, d_k\}$. The model generates a response conditioned jointly on the query and the retrieved context, i.e., $M(y \mid q, D_k)$.

However, when the number of retrieved documents is large, directly passing D_k to M significantly increases the token budget and computational cost. To address this, recent works such as (Xu et al., 2024) propose compressing the retrieved texts into a compact text that preserves the query-relevant evidence while discarding redun-

dancy. Formally, the objective is expressed as:

$$\arg\max_{\pi} M\big(y \mid q, C_\pi\big), \quad C_\pi = \pi(q, D_k),$$

where C_π denotes the compressed context obtained through a compression function π. The token length of C_π, denoted $l(C_\pi)$, is expected to be much smaller than that of the original D_k while still retaining critical information.

RAG Pipeline with Compression A RAG pipeline working with compression as follows: Given a query q and a document collection C, a retriever first identifies the top-k relevant documents: $D = \{d_1, \ldots, d_k\} = \text{Retriever}(q, C)$.

These documents are then processed by a compression component that produces a shortened but information-preserving set:

$$D' = \text{Compressor}(q, D); \text{s.t. } l(D') \ll l(D).$$

Finally, the compressed set D' is provided to the language model, which generates the final answer: $a = \text{LLM}(q, D')$.

For compression in RAG to be effective, it should: (1) *Evidence Retention* – preserve all essential information for accurate query answering; (2) *Processing Efficiency* – remain lightweight and fast to avoid heavy resource use and latency; (3) *Token Efficiency* – produce a context D' with far fewer tokens, lowering cost and speeding inference.

Sentence-level Pruning Our pipeline applies sentence-level pruning to avoid key-phrase fragmentation and preserve entity relationships often lost in token-level compression. Operating at sentence granularity keeps both syntax and meaning intact. Each retrieved chunk is segmented into sentences with a rule-based tool such as spaCy (Honnibal et al., 2020), yielding for each document $d_i \in D$ a set $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in}\}$, where s_{ij} is the j-th sentence (see **step 1** in Fig. 2).

3.1 Leave-One-Out Delta-based Sentence Classification

From this step, our solution transforms the problem of context pruning into relevance classification of elemental sentences within each chunk, following a similar formulation to (Hwang et al., 2024). However, instead of merely training a model to predict a direct classification signal given a question and its corresponding sentences, our method is built on the intuition that if we are asked to find clues (hints) within a passage, we can sense the difference between the original and a redacted version. Likewise, when assessing the clue richness of a passage versus its redacted version (where a critical sentence is removed), the gap between their scores should be noticeable, whereas these gaps are trivial if the censored lines are irrelevant or valueless to the question.

From this observation, our method teaches an encoder-only language model ability to assess the clue richness of the input sequence made from a question and a passage. The loss function to guide this learning is mimicking the intuition we mentioned: make the score gap Δ between the base case (original passage) and the redacted version noticeable when the censored sentence is critical while minimize this gap if the censored sentence is non-critical.

3.1.1 Formal Loss Function

Let f_θ denote our encoder model parameterized by θ, which produces a scalar score representing the clue richness of an input sequence. For a question q and passage $P = \{s_1, s_2, ..., s_n\}$ with n sentences, we define:

- $p_0 = f_\theta(q, P)$: clue richness score for the full context
- $p_{\setminus k} = f_\theta(q, P \setminus \{s_k\})$: score when s_k is removed
- $\Delta_k = p_0 - p_{\setminus k}$: score gap from removing s_k
- $y_k \in \{0, 1\}$: label for s_k (1 if critical, 0 not)

Our loss function has 2 cases based on the type of passage:

For clue-filled passages (where $\exists k : y_k = 1$)

The loss combines ranking and classification objectives: $\mathcal{L}_{\text{yes}} = \mathcal{L}_{\text{rank}} + \lambda \cdot \text{BCE}(p_0, 1)$ (0)

where $\mathcal{L}_{\text{rank}} = \alpha\mathcal{L}_{\text{ord}} + \beta\mathcal{L}_{\text{crit}} + \gamma\mathcal{L}_{\text{non}}$ with:

$$\mathcal{L}_{\text{ord}} = \sum_{i:y_i=1} \sum_{j:y_j=0} \max(0, m_1 - (\Delta_i - \Delta_j)) \tag{1}$$

$$\mathcal{L}_{\text{crit}} = \sum_{k:y_k=1} \max(0, m_2 - \Delta_k) \tag{2}$$

$$\mathcal{L}_{\text{non}} = \sum_{k:y_k=0} \max(0, \Delta_k + m_3) \tag{3}$$

where $\mathcal{L}_{\text{ord}}$ enforces larger gaps for critical sentences; $\mathcal{L}_{\text{crit}}$ ensures significant drops ($\geq m_2$)

when removing critical sentences; and $\mathcal{L}_{\text{non}}$ penalizes large changes for noncritical ones. Margins $m_1, m_2, m_3 > 0$ are training hyper-parameters with $m_3 << m_2, m_1$.

For clue-free passages (where $\forall k : y_k = 0$)

Originally we only had $\mathcal{L}_{\text{rank}}$ as our observation, however, we added Binary Cross Entropy loss (BCE) after considering the case none of the sentences within a passage has valuable intelligence for the question. Thus, the loss at (0) and (4) have BCE included, where (4) enforces low scores and minimal variation for such passage without any clue for the query:

$$\mathcal{L}_{\text{no}} = \lambda\Big(\text{BCE}(p_0, 0) + \sum_k \text{BCE}(p_{\setminus k}, 0)\Big) + \gamma \sum_k \max(0, |\Delta_k| - m_3) \quad (4)$$

Final Loss For a batch $\mathcal{B}$: $\mathcal{L}_{\text{total}} = \frac{1}{|\mathcal{B}|}\sum_{d\in\mathcal{B}} \mathcal{L}^{(d)}$ where $\mathcal{L}^{(d)}$ is either $\mathcal{L}^{(d)}_{\text{yes}}$ or $\mathcal{L}^{(d)}_{\text{no}}$. We also use a positive weight > 1 in BCE to alleviate class imbalance.

For training efficiency with long passages, we sample m sentences which $n > m >$ number of critical sentences. This sampling reduces memory requirements while preserving all critical sentences for loss function calculation especially when extreme long passages have large number of sentences (i.e. big n).

3.1.2 Inference Strategy

At inference, the encoder f_θ scores clue richness for a question–passage pair and uses leave-one-out deltas to identify critical sentences.

Large Δ_k indicates that drop s_k harms clue richness; near-zero or negative Δ_k indicates little or no contribution. A passage with n sentences takes $n + 1$ forward passes independently as **step 2** in Fig 2: compute $\{p_0, p_1, .., p_n\}$ in parallel (2a), then compute $\Delta_1, \Delta_2, ..., \Delta_n$ as (2b).

As there is a BCE term in the loss, to quickly detect clue-free passages, we use $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid) on p_0 with a threshold d_{min}, formal predicate as: $\sigma(p_0) < d_{min}$ to label the whole passage *non-critical*. Otherwise, we proceed following the below core strategy like **step 3** in Fig 2:

Adaptive threshold (gap-based). Let $\delta_{\min} > 0$ be a minimal significance level, and $D^+ = \{\Delta_k : \Delta_k > \delta_{\min}\}$.

1. If $D^+ = \varnothing$, label all sentences *non-critical*.
2. Else: sort D^+ in descending order $\Delta_{[1]} \geq \cdots \geq \Delta_{[m]}$; compute gaps $g_i = \Delta_{[i]} - \Delta_{[i+1]}$ for $i = 1, \ldots, m-1$. Let $i^\star = \arg\max_i g_i$.
3. Set the adaptive threshold $\tau = \max\{\delta_{\min}, \Delta_{[i^\star+1]}\}$.
4. Classify s_k as *critical* if $\Delta_k > \tau$; otherwise *non-critical*.

If Δ are uniformly high (no clear gap), the rule defaults to $\tau = \delta_{\min}$, preserving more content. The gap heuristic adapts per passage, maintaining high precision on informative sentences while filtering redundancy. $d_{min}, \delta_{\min}$ are inference hyperparameters that we use grid search to tune.

4 Experiment

4.1 Experimental Setup

Training Phase We use a random subset 28897 out of 90447 questions from the **original** HotpotQA (Yang et al., 2018) train set to train with a 9:1 split, where there are passages (and annotated critical sentences) for each question. The HotpotQA dev set (distractor) is used for grid search to find best $d_{min}, \delta_{\min}$ that gain the most F1 of classification. We merged very short passages into longer ones to mitigate the dataset's severe passage-length imbalance, shorten training time, and enable additional data augmentation. Training is done on single NVIDIA RTX 4090 GPUs (multiple similar workstations) using LoRA (Hu et al., 2022) to finetune the model with the standard separator token (e.g. [SEP]) by ModernBERT automatically defined.

RAG Configure The main pipeline consists of 3 primary components. The Contriever-MSMARCO (Izacard et al., 2021) was used as chunk retriever on the *2018 Wikipedia corpus* (dataset) (Karpukhin et al., 2020) to create passages for each question from Question Answering (QA) datasets. Our compressor is based on an encoder-only model – ModernBERT (Warner et al., 2024), large version (395M parameters) used as default, base (139M parameters) used in ablation. For reading to answer, we mainly employed Llama-3.1-8B-Instruct, Llama-3.3-70B-Instruct (Dubey et al., 2024); with Google Gemini-2.5-flash (Comanici et al., 2025) and Moonshot Kimi K2 (Team et al., 2025) for

further exploration. Vertex AI and OpenRouter services were used to provide mentioned reader APIs. Reader outputs are generated with a temperature of 0.0001 and a top-p of 1.

To evaluate, HotpotQA (HQA) (Yang et al., 2018), 2WikiMultihopQA (2WIKI) (Ho et al., 2020), and Musique (Trivedi et al., 2022b) datasets as multi-hop QA; and Natural Questions (NQ) (Kwiatkowski et al., 2019), TriviaQA (TQA) (Joshi et al., 2017) datasets as single-hop QA were selected. The *test* set of TQA and the *dev* sets of other datasets are used; the sets are the same as those evaluated by (Yoon et al., 2024; Hwang et al., 2024) due to the same retriever and corpus.

Baselines We compare our method against the following context compression methods: RECOMP-Abs (Xu et al., 2024) uses T5-based (775M) while RECOMP-Extr employs a dual Contriever-based (110M) transformers; CompAct (Yoon et al., 2024) used Mistral-7B-based iterative compression; Refiner (Li et al., 2024) uses Llama2-7B-based compression; LongLLMLingua (Jiang et al., 2023) uses Llama2-7B-chat for token-level extraction at 0.4 dynamic compression rate; and EXIT (Hwang et al., 2024) with Gemma-2B. For fair comparison, we applied the same preprocessing; we enabled flash attention (Dao, 2024) for baselines (if available); used the same number of threads for multi-threading (if available) and lifted up unreasonable truncated limit defaults to avoid long context cutoff.

Evaluation Metrics Exact Match (EM↑) and F1↑ score to measure effectiveness in question answering. Compression latency (Time↓) in seconds, compression ratio (Rate↓) in percent to evaluate compression speed and efficiency. The latency here is compressing duration measured in an end-to-end manner to simulate a realistic processing timeline. The compression rate is defined as the ratio of the length of the compressed sequence to the length of a original sequence using $cl100k_base$ tokenizer (tiktoken). Each experiment is run 3 times (different batch sizes) on single NVIDIA RTX 4090 GPU and then take average.

4.2 Main Results

Table 1 shows evaluation results across 5 datasets and compressing methods by 2 commonly used readers. Note that the compression latency (Time) and compression ratio (Rate) depend on the top-k value and are repeated for the 2 readers.. Although trained only on HQA questions with original HQA context corpus, our pruner delivers a high generalization across datasets (both single-hop and multi-hop). Compared with other compressors, our method mostly yields best or second best at QnA metrics on datasets on different readers and different number of top chunks context. We also achieve approximately equal answer quality or surpassed *Raw* baselines on several occasions on nearly all datasets except NQ. Regarding the compression efficiency, our method always achieves rapid responsiveness (2nd best, $< 0.05s$ at top-5 and $< 0.2s$ at top-20); whereas context reduction is at a decent level to save considerable token cost: $\leq 20\%$ at top-5 and $< 14\%$ at top-20.

Meanwhile, existing methods struggle to have good trade-off in metrics, e.g. RECOMP-ext is always the fastest but usually retain long lengthy content and yields much lower answer quality. Similarly, CompAct, RECOMP-abs or Refiner compress context to very compact level but come with a price of heavily slow execution (10 to 40 times compared to ours).

For an overall view, we aggregate Table 1 into a single metric vector per compressor as in Table 2. For each metric (EM, F1, Time, Rate), we take unweighted means in 2 steps: (i) average across the 4 experimental settings (2 readers × 2 retrieval depths), then (ii) average across the 5 datasets (NQ, TQA, HQA, 2Wiki, Musique). In general, we outperformed all baselines on answer accuracy with a clear gap to the next contenders (e.g. LongLLMLingua or EXIT) while holding second fastest position and a compact compressed ratio.

In Table 5 in Appendix, we tried experiments with a proprietary LLM Gemini-2.5-flash and a recent mega-size Kimi-K2 as readers on HQA (intra-domain) and 2Wiki (inter-domain) to have further insight on how good compressors perform. Results show a similar story, we achieve mostly best result in answer quality and second best in latency while note that our method yields much more compactness compared to LongLLMLingua (39.1% vs 8.5% in HQA and 39.3% vs 6.4% in 2Wiki).

Robustness To assess the robustness of compression as the retrieved set enlarges, we increase the number of top chunks $k \in \{5, 10, 20, 30\}$ in experiments on HQA dataset with Llama-3.1-8B reader as in Figure3. Our approach consistently improves EM scores, rising from 30.97 points at $k = 5$ to 32.82 points at $k = 30$, while avoiding the perfor-

Table 1: Performance across models and datasets by reader Llama-3.1-8B Instruct and Llama-3.3-70B Instruct. Best (**bold**) and second-best (underlined) are determined using higher-precision values before rounding to shown decimal places. Metrics EM, F1 higher are better while Time, Rate smaller are better.

	NQ				TQA				HQA				2Wiki				Musique			
	EM	F1	Time	Rate	EM	F1	Time	Rate	EM	F1	Time	Rate	EM	F1	Time	Rate	EM	F1	Time	Rate
Llama-3.1-8B Instruct																				*Top-5 retrieved chunks*
Raw	*35.4*	*47.3*	-	*100*	*60.9*	*69.8*	-	*100*	*30.2*	*40.9*	-	*100*	*23.5*	*30.2*	-	*100*	*4.3*	*10.8*	-	*100*
CompAct	31.4	42.0	2.573	12.1	60.0	69.1	2.594	12.0	28.1	37.5	2.517	11.9	22.1	28.8	2.354	10.6	**6.1**	**13.6**	2.781	11.9
EXIT	**33.8**	45.0	0.359	56.1	60.1	68.2	0.340	50.9	28.9	39.4	0.347	48.6	21.3	27.4	0.380	42.6	4.0	10.8	0.349	45.3
RECOMP-abs	33.5	**45.1**	0.765	**7.8**	53.8	63.5	0.578	**7.6**	28.3	39.4	0.527	**7.0**	**24.4**	**29.6**	0.436	**5.5**	4.3	11.8	0.499	**6.6**
RECOMP-ext	33.8	45.1	**0.009**	47.4	57.4	65.7	**0.009**	47.7	26.7	36.5	**0.010**	50.6	19.2	25.9	**0.008**	50.1	4.2	10.5	**0.009**	48.6
Refiner	33.2	44.9	2.403	15.3	60.3	69.4	1.967	13.2	29.7	40.5	1.624	10.4	22.2	28.2	1.325	8.3	5.3	12.0	1.436	8.9
LongllmLingua	29.8	41.2	0.345	46.2	59.6	68.2	0.344	45.9	28.6	38.5	0.329	47.1	21.5	27.0	0.339	36.9	4.4	10.9	0.347	46.3
Ours	33.4	44.9	0.041	20.0	**60.6**	**69.7**	0.046	20.1	**31.0**	**42.0**	0.047	15.9	22.3	28.8	0.043	12.2	5.7	12.7	0.048	10.8
																				Top-20 retrieved chunks
Raw	*37.2*	*49.3*	-	*100*	*64.0*	*72.7*	-	*100*	*31.1*	*41.7*	-	*100*	*25.6*	*32.0*	-	*100*	*4.6*	*11.1*	-	*100*
CompAct	33.8	45.4	4.312	3.7	60.0	69.0	4.102	3.5	30.9	42.1	4.696	3.7	21.6	28.2	4.905	3.5	5.3	12.4	5.869	4.1
EXIT	**34.6**	45.8	1.414	51.6	58.6	66.4	1.452	45.4	29.9	39.8	1.474	41.9	23.1	28.9	1.619	37.6	4.2	10.9	1.480	41.9
RECOMP-abs	32.9	44.9	1.223	**2.1**	55.9	65.3	1.332	**2.0**	28.5	39.8	1.206	**1.8**	22.4	28.2	1.145	**1.4**	5.2	11.9	1.393	**1.6**
RECOMP-ext	30.9	41.3	**0.036**	11.4	54.8	62.6	**0.037**	11.5	23.3	32.1	**0.035**	12.4	15.8	22.3	**0.036**	12.2	4.0	9.7	**0.037**	12.0
Refiner	31.0	42.0	6.727	8.8	59.1	68.0	5.013	6.5	29.1	39.3	3.915	4.4	22.2	28.1	3.219	3.7	5.1	11.6	4.551	5.0
LongllmLingua	33.4	45.5	1.335	39.0	62.8	70.8	1.436	39.1	30.9	41.7	1.357	39.1	24.9	30.7	1.357	39.3	4.7	11.4	1.343	39.4
Ours	34.4	**46.5**	0.169	13.8	**63.9**	**72.2**	0.180	12.9	**32.4**	**43.7**	0.197	8.5	**25.0**	**30.8**	0.183	6.4	**6.5**	**13.8**	0.183	7.0
Llama-3.3-70B Instruct																				*Top-5 retrieved chunks*
Raw	*36.8*	*51.1*	-	*100*	*65.8*	*75.2*	-	*100*	*35.6*	*47.8*	-	*100*	*27.2*	*34.9*	-	*100*	*8.2*	*16.8*	-	*100*
CompAct	35.3	48.4	2.573	12.1	65.2	74.6	2.594	12.0	36.6	48.6	2.517	11.9	27.0	34.6	2.354	10.6	9.2	17.8	2.781	11.9
EXIT	35.0	48.5	0.359	56.1	64.4	73.8	0.340	50.9	34.2	45.6	0.347	48.6	24.3	31.2	0.380	42.6	7.5	16.1	0.349	45.3
RECOMP-abs	33.9	46.6	0.765	**7.8**	56.9	66.9	0.578	**7.6**	32.2	44.3	0.527	**7.0**	26.7	32.3	0.436	**5.5**	6.5	14.2	0.499	**6.6**
RECOMP-ext	**36.4**	**49.1**	**0.009**	47.4	65.0	74.1	**0.009**	47.7	32.7	44.1	**0.010**	50.6	22.0	30.0	**0.008**	50.1	7.0	15.3	**0.009**	48.6
Refiner	35.1	48.6	2.403	15.3	65.6	75.2	1.967	13.2	35.8	47.6	1.624	10.4	25.7	32.7	1.325	8.3	8.3	16.6	1.436	8.9
LongllmLingua	32.8	46.3	0.345	46.2	65.7	74.9	0.344	45.9	34.4	46.2	0.329	47.1	25.1	31.8	0.339	36.9	7.9	16.5	0.347	46.3
Ours	35.1	48.8	0.041	20.0	**66.8**	**76.3**	0.046	20.1	**37.3**	**50.1**	0.047	15.9	**31.2**	**37.6**	0.043	12.2	**9.5**	**18.8**	0.048	10.8
																				Top-20 retrieved chunks
Raw	*39.1*	*53.4*	-	*100*	*68.8*	*77.9*	-	*100*	*38.2*	*50.3*	-	*100*	*30.6*	*38.5*	-	*100*	*10.0*	*18.9*	-	*100*
CompAct	35.2	48.5	4.312	3.7	64.9	74.0	4.102	3.5	36.1	47.8	4.696	3.7	26.5	33.2	4.905	3.5	8.4	15.9	5.869	4.1
EXIT	**37.0**	**51.3**	1.414	51.6	66.3	75.8	1.452	45.4	36.2	48.2	1.474	41.9	27.4	34.7	1.619	37.6	8.7	17.7	1.480	41.9
RECOMP-abs	34.2	47.2	1.223	**2.1**	60.2	69.9	1.332	**2.0**	33.1	45.4	1.206	**1.8**	27.7	33.4	1.145	**1.4**	6.7	14.8	1.393	**1.6**
RECOMP-ext	33.5	45.7	**0.036**	11.4	64.0	72.6	**0.037**	11.5	28.3	38.9	**0.035**	12.4	17.1	24.6	**0.036**	12.2	6.0	14.1	**0.037**	12.0
Refiner	33.4	46.4	6.727	8.8	64.1	73.6	5.013	6.5	33.4	45.0	3.915	4.4	25.7	32.0	3.219	3.7	7.8	16.1	4.551	5.0
LongllmLingua	36.2	50.8	1.335	39.0	**67.9**	**77.2**	1.436	39.1	36.6	48.9	1.357	39.1	29.4	37.1	1.357	39.3	8.9	18.2	1.343	39.4
Ours	36.3	50.5	0.169	13.8	67.3	76.9	0.180	12.9	**38.5**	**51.2**	0.197	8.5	**31.4**	**37.5**	0.183	6.4	**10.7**	**20.1**	0.183	7.0

mance degradation observed in RECOMP variants, Refiner at larger k. Across retrieval sizes, ours outperforms baselines, with an exception being lower than CompAct at $k = 10$. Regarding efficiency, we achieves competitive latency that scales nearly linearly with k (0.047s to 0.263s). Compared to EXIT or LongLLMLingua, our method shrinks length by more than half while still delivering superior performance. Although our compression ratio is slightly higher than Refiner, Compact, and Recomp-Abs, we find this trade-off to be justified, as ours outperforms theirs by clear margins in accuracy with high throughput.

For overview visualization in comparing models, we present Fig. 1 to compare EM, F1, Context Saved %, and Questions Per Second across models with mean values over all datasets at $k = 5$ by the Llama-3.1-8B reader. Although we cannot outperform in all aspects, our broadest coverage in the chart indicate a great balance between performance and efficiency: it achieves competitive compression savings while maintaining SOTA performance which existing works struggled to attain.

In term of GPU memory usage and floating-point operations (FLOPs) required to operate, based on experiments on HQA's subset 500 at top-10 chunks, which are detailed in Table 6 of Appendix section, our method peaks at 2.37GB memory and on aver-

Table 2: Overall performance across compressors, via 2-stage averaging: first over 4 settings (2 readers × 2 retrieved depths), then over all 5 datasets.

	EM	F1	Time	Rate
CompAct	32.2	41.6	3.670	7.7
EXIT	32.0	41.3	0.921	46.2
RECOMP-abs	30.4	39.7	0.910	**4.3**
RECOMP-ext	29.1	38.0	**0.023**	30.4
Refiner	31.6	40.9	3.218	8.5
LongllmLingua	32.3	41.7	0.853	41.8
Ours	**34.0**	**43.6**	0.114	12.8

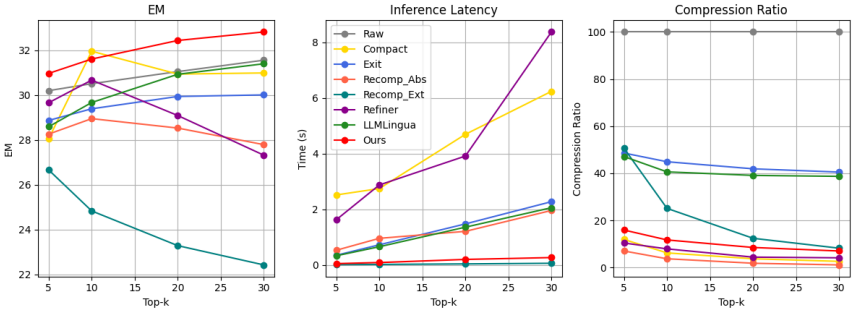

Figure 3: Performance analysis on HQA at increasing top-k = $\{5, 10, 20, 30\}$ by reader Llama-3.1-8B Instruct, comparing EM, compression latency and compression ratio between baselines and our proposed method.

age 7039 GFLOPs per query which is only behind 650MB and 400 GFLOPs that of the RECOMP-ext. Compared to other methods like EXIT with 9.75GB or Refiner with 14.31GB, our model just requires a considerable less memory which enable applicability on edge devices.

5 Ablation Study

Table 3: Performance over subsets 500-sample of 5 datasets on our model trained with different loss variants at top-10 chunks by Llama-3.3-70B Instruct. *Full* uses the complete loss; *-BCE* excludes all the BCE terms; *-BCE -crit* excludes both similarly.

Variants	NQ		TQA		HQA		2Wiki		Musique	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Full	36.8	51.8	69.2	76.7	38.6	51.1	30.8	35.4	10.6	18.8
-BCE	33.7	46.8	67.8	75.9	35.0	46.3	28.1	33.2	8.8	18.3
-crit	35.6	49.3	69.2	76.8	37.6	50.1	29.8	34.0	10.3	17.9
-BCE -crit	32.6	46.4	68.4	76.2	34.6	46.6	24.6	30.1	7.2	17.2

To better analyze the contribution of loss parts, we conduct a study by systematically removing them and evaluating performance. As shown in Table 3, the model trained by the complete loss function consistently achieves the best results across all datasets. Ablating BCE terms causes a significant performance degradation than removing the $\mathcal{L}_{\text{crit}}$ component, indicating that BCE terms are critical for model accuracy. While one might hypothesize that $\mathcal{L}_{\text{ord}}$ is sufficient for creating a discriminative distance between sentences, the performance drop from removing $\mathcal{L}_{\text{crit}}$ confirms its essential role. As expected, excluding both components leads to the lowest performance, underscoring the synergistic value of each term in the final loss calculation.

We also evaluate two backbone sizes, large and base, to analyze the trade-off between performance and efficiency in Table 4. The ModernBERT-large model consistently achieves higher EM and F1 scores across all datasets. Conversely, the base model demonstrates a significant speed advantage, with a latency around half that of the large model (e.g. 44ms vs 87ms at HQA). This presents a clear choice between the superior accuracy of the larger model and the lightweight of the base version.

Table 4: Performance over subsets 500-sample of 5 datasets on our 2 model backbones: ModernBERT-large and ModernBERT-base at top-10 chunks by Llama-3.3-70B Instruct. Compression time (Lat) is in milliseconds.

	NQ		TQA		HQA		2Wiki		Musique	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
-large	36.8	51.8	69.2	76.7	38.6	51.1	30.8	35.4	10.6	18.8
-base	33.2	46.5	66.9	75.1	34.8	47.4	26.8	32.2	9.1	17.2
	Lat	Rate	Lat	Rate	Lat	Rate	Lat	Rate	Lat	Rate
-large	82.0	16.1	86.7	16.6	87.3	11.4	88.7	8.5	89.1	8.4
-base	42.9	11.8	44.2	14.5	44.0	13.8	44.5	13.2	44.9	10.4

6 Conclusion

We present a margin-based framework to enhance context compression in RAG systems. Our framework uses a query-driven pruning strategy to select sentences vital for answering the query. We find that encoder-only models are sufficient and more efficient than decoder-based LLMs for sentence-level compression. Through experiments, our approach outperforms recent methods in single-hop and multi-hop QA tasks while achieving lower memory usage, faster inference, and a more favorable compression ratio. Our method shows strong zero-shot generalization and transfers across open-source models of different scales when trained exclusively on the HQA dataset. These results indicate that our method can outperform larger and more complex compression methods, offering a practical solution for real-world RAG applications.

Limitations

Our approach relies on explicit sentence-level annotations for training, which in the HQA dataset were obtained manually. Although such labels could, in principle, be generated by advanced large language models (e.g., commercial systems such as GPT-5 or Gemini-2.5-Pro), this raises concerns about the cost of API usage and the reliability of LLM-as-judge annotations.

Moreover, because our method performs sentence-level pruning, complex sentences that are long, noisy, or contain extraneous details remain only partially optimized for length. We anticipate that a finer-grained strategy—such as pruning at the phrase or clause level—could address this limitation. However, the scarcity of high-quality annotations at that level presents a significant challenge.

References

Alberto Benayas, Miguel Angel Sicilia, and Marçal Mora-Cantallops. 2025. A comparative analysis of encoder only and decoder only models in intent classification and sentiment analysis: Navigating the trade-offs in model size and performance. *Language Resources and Evaluation*, 59(3):2007–2030.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, Adriane Boyd, and 1 others. 2020. spacy: Industrial-strength natural language processing in python.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Taeho Hwang, Sukmin Cho, Soyeong Jeong, Hoyun Song, SeungYoon Han, and Jong C Park. 2024. Exit: Context-aware extractive compression for enhancing retrieval-augmented generation. *arXiv preprint arXiv:2412.12559*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*.

Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

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

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances*

in neural information processing systems, 33:9459–9474.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Zhonghao Li, Xuming Hu, Aiwei Liu, Kening Zheng, Sirui Huang, and Hui Xiong. 2024. Refiner: Restructure retrieval content efficiently to advance question-answering capabilities. *arXiv preprint arXiv:2406.11357*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

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

Dan Saattrup Nielsen, Kenneth Enevoldsen, and Peter Schneider-Kamp. 2024. Encoder vs decoder: Comparative analysis of encoder and decoder language models on multilingual nlu tasks. *arXiv preprint arXiv:2406.13469*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Muhammad Reza Qorib, Geonsik Moon, and Hwee Tou Ng. 2024. Are decoder-only language models better than encoder-only language models in understanding word meaning? In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16339–16347.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and 1 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022a. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022b. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, and 1 others. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *arXiv preprint arXiv:2412.13663*.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. *arXiv preprint arXiv:2210.03162*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. Recomp: Improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

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

Zheng Zhao, Shay B Cohen, and Bonnie Webber. 2020. Reducing quantity hallucinations in abstractive summarization. *arXiv preprint arXiv:2009.13312*.

Appendix

In the Appendix, additional implementation details, supplementary results and analyses which are not covered in the main text is provided.

A Extended Implementation Details

This section report our training environment, and prompt templates. All training experiments are done on single NVIDIA RTX 4090 GPUs (multiple similar workstations: same GPU, similar CPUs and RAM sizes).

A.1 Training Configuration

We adopt LoRA (Hu et al., 2022) to finetune the pruning model at bfloat16 precision for parameter-efficient training while preserving generalization from the original weights. The model was trained with the following hyper-parameters:

- Batch size: $\{1, 2, 4\}$
- Gradient accumulation steps: 8
- Learning rate: 7e-5
- Weight decay: 0.02
- Warmup steps: 200
- Training epochs: up to 6
- Optimizer: AdamW
- LoRA configuration: Rank = 64, Scaling = 16, Dropout = 0.1
- $m1 = m2 = 0.35, m3 = 0.035$; $\alpha = 1.5, \beta = 1.25, \gamma = 1.0; \lambda = 0.75$; BCE positive weight$= 5.0$
- Max sampling $m = 50$ sentences (to curb memory usage in extreme long passages while still ensure nearly same effect compared with inspecting all n sentences of those)

Regarding detail model architecture, our model is a lightweight scoring architecture built on a pretrained ModernBERT encoder plus a learnable multi-head attention pooling layer. Instead of CLS or mean pooling, H learned query vectors (one per pooling head; H = number of pooling heads $= 8$, with hidden size divisible by H) attend over the token sequence to produce head-specific summaries that are concatenated and linearly projected. Padding is masked before and after softmax for stability. A dropout layer and a final linear unit map the pooled representation to a single scalar score.

Model selection was guided by validation loss and it took around 21 hours for each training (up to 6 epochs).

A.2 Data Augmentation & Pre-processing.

We also add some data augmentation operations: **(1)** Randomly drop extra non-critical sentences when inspecting (dropping) a critical sentence: 10%; **(2)** Randomly drop extra non-critical sentences when inspecting (dropping) a non-critical sentence: 10%; **(3)** Randomly insert punctuation between sentences: 20%; **(4)** Randomly add start word(phrase), end word(phrase): 5%. We only used query and corresponding context set as the **original** HotpotQA set provided, no cross pairing (i.e. a query with non-corresponding context set) was used.

Some pre-processing we used: **(1)** merge overlapped chunks from a same document to reduce duplication in parts and titles (not longer than 20 sentences); **(2)** too short chunks–having less than 4 sentences, were also merged to become longer chunks for less extreme length imbalance in dataset (not longer than 12 sentences).

A.3 More Inference Configuration

Document level threshold and delta min $d_{min}, \delta_{\min}$ were determined by grid search for each model weight set as mentioned before. Our best result weight set is with $d_{min} = 0.12, \delta_{\min} = 0.01$

A.4 QA Prompt Template

Below Listing 1 is the QA prompt template for LLM readers to consider the query with the provided compressed context and answer.

Listing 1: QA Prompt Template

```
Context information is:
```{Merged Compressed Context}```

Given provided context (might not be
    sufficient for below query), answer
    the query without any explanation.
Query: `{question}`
Answer (in plain text):
```

B Additional Experimental Results

Table 5 provided extra insight on compressors performance with commercial and powerful LLM reader as such as Gemini-2.5-flash and Moonshot Kimi-K2.

Table 6 reports the resource requirement of compressors tested on the standard subset 500 samples
```

Table 5: Performance across compressing models on HQA and 2Wiki datasets using Gemini-2.5-flash and Kimi-K2 readers at top-20 retrieved chunks.

Model	HQA				2Wiki				Reader
	EM	F1	Time	Rate	EM	F1	Time	Rate	
Raw	*37.1*	*48.3*	-	*100*	*29.4*	*35.8*	-	*100*	Gemini-2.5-flash
CompAct	30.1	39.5	4.696	3.7	16.6	22.6	4.905	3.5	
EXIT	33.5	44.2	1.474	41.9	23.2	29.7	1.619	37.6	
RECOMP-abs	30.0	40.5	1.206	**1.8**	23.0	27.6	1.145	**1.4**	
RECOMP-ext	20.0	27.8	**0.035**	12.4	11.5	18.4	**0.036**	12.2	
Refiner	29.3	38.6	3.915	4.4	19.8	24.8	3.219	3.7	
LongllmLingua	34.3	45.4	1.357	39.1	**25.4**	**31.5**	1.357	39.3	
Ours	**36.5**	**47.5**	0.197	8.5	24.7	30.4	0.183	6.4	
Raw	*40.8*	*53.4*	-	*100*	*34.4*	*41.6*	-	*100*	Kimi-K2
CompAct	36.1	46.8	4.696	3.7	23.9	29.0	4.905	3.5	
EXIT	38.3	50.5	1.474	41.9	31.0	37.3	1.619	37.6	
RECOMP-abs	34.2	46.1	1.206	**1.8**	29.0	33.5	1.145	**1.4**	
RECOMP-ext	31.2	42.0	**0.035**	12.4	22.6	28.8	**0.036**	12.2	
Refiner	34.7	45.8	3.915	4.4	28.9	34.0	3.219	3.7	
LongllmLingua	39.2	51.7	1.357	39.1	32.4	39.0	1.357	39.3	
Ours	**41.0**	**53.5**	0.197	8.5	**33.4**	**39.2**	0.183	6.4	

Table 6: GPU memory peak and GFLOPs per question across compressors on HQA subset 500 at top-10 chunks

	Mem Peak	GFLOPs/question
CompAct	14.07	25692.7
EXIT	9.75	75948.0
RECOMP-abs	4.73	2050.5
RECOMP-ext	**0.65**	**400.9**
Refiner	14.31	22793.5
LongLLMLingua	14.01	72017.3
Ours	2.37	7039.4

of HotpotQA dataset at top-10 retrieved chunks. GPU memory peak (Mem Peak) is maximum memory peaked during each experiment while average GFLOPs (Giga FLOPs) per question.

Table 7: Average metrics on all 5 datasets across compressors at top-5 chunks by Llama-3.1-8B Instruct reader.

	EM	F1	QPS	Save %
CompAct	29.5	38.2	0.4	88.3
EXIT	29.6	38.1	2.8	51.3
RECOMP-abs	28.9	37.9	1.8	**93.1**
RECOMP-ext	28.3	36.7	**109.6**	51.1
Refiner	30.1	39.0	0.6	88.8
LongLLMLingua	28.8	37.1	2.9	55.5
Ours	**30.6**	**39.6**	22.2	84.2

Table 7 is the averaged metrics on all 5 datasets (NQ, TQA, HQA, 2Wiki, Musique) by compressors at top-5 retrieved chunks by Llama-3.1-8B Instruct reader. Question per second $QPS = 1/Latency$; Context Saved % is $100\%-$ Compression ratio. This is the data was used for the radar chart 1. Note that Table 7 is different with Table 2 because Table 7 is average values at only one setting (Llama-3.1-8B at $k = 5$) while Table 2 are mean values across all 4 settings. It make more sense to draw Fig 1 by data in Table 7 because at the same k, chunks across different datasets are similar due to the same corpus and the same retriever, thus these values yield more stable trade-off profiles between compressors.