Efficient Prompt Compression with Evaluator Heads for Long-Context Transformer Inference

Weizhi Fei[1] **Xueyan Niu**[2*] **Guoqing Xie**[3] **Yingqing Liu**[3] **Bo Bai**[2] **Wei Han**[2]

[1]Department of Mathematical Sciences, Tsinghua University, Beijing, China
[2]Theory Lab, 2012 Labs, Huawei Technologies Co., Ltd.
[3]Architecture & Design, ICT Products & Solutions, Huawei Technologies Co., Ltd.

Abstract

Although applications involving long-context inputs are crucial for the effective utilization of large language models (LLMs), they also result in increased computational costs and reduced performance. To address this challenge, we propose an efficient, training-free prompt compression method that retains key information within compressed prompts. We identify specific attention heads in transformer-based LLMs, which we designate as *evaluator heads*, that are capable of selecting tokens in long inputs that are most significant for inference. Building on this discovery, we develop EHPC, an Evaluator Head-based Prompt Compression method, which enables LLMs to rapidly "skim through" input prompts by leveraging only the first few layers with evaluator heads during the pre-filling stage, subsequently passing only the important tokens to the model for inference. EHPC achieves state-of-the-art results on two major benchmarks: prompt compression and long-context inference acceleration. Consequently, it effectively improves performance with the reduced costs associated with commercial API calls. We further demonstrate that EHPC attains competitive results compared to key-value cache-based acceleration methods, thereby highlighting its potential to enhance the efficiency of LLMs for long-context tasks.

1 Introduction

Large language models (LLMs) have exhibited exceptional capabilities in a variety of real-world tasks and applications, with an increasing need for processing long inputs in areas such as literary novels, legal documents, instruction manuals, and code documentation. Inference tasks that require understanding of long contexts, such as long document summarization Zhang et al. [2024a], reasoning Fei et al. [2024a], and autonomous agents Singh et al. [2024], Chen et al. [2024], are of particular importance due to the high stakes in these scenarios. However, the deployment of LLMs is challenged by the computational and memory demands inherent to transformer-based architectures, resulting in increased latency, particularly when processing lengthy input prompts.

Prompt compression, which involves substituting the input prompts provided to a language model with more succinct versions, has surfaced as a promising strategy for enhancing long-text understanding and mitigating associated costs. Current mainstream methods, such as SelectContext Li et al. [2023], LLMLingua [Jiang et al., 2023a] and LongLLMLingua [Jiang et al., 2023b], typically rely on pre-trained LLMs, utilizing the logits or perplexity of the prompts to evict tokens deemed insignificant. These approaches often necessitate chunking long texts for processing, leading to numerous repeated calls of the LLM and consequently incurring considerable time complexity. More efficient compression techniques, such as LLMLingua-2 Pan et al. [2024], generally require

*Correspondence to: niuxueyan3@huawei.com

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

the training of a secondary, smaller model on labeled datasets. Although these methods reduce compression time, they also incur substantial training expenses and may exhibit performance drops in out-of-distribution contexts compared to direct utilization of the LLM as a compressor. In this paper, we present an **E**valuator **H**ead-based **P**rompt **C**ompression method, dubbed `EHPC`, which is built upon the efficient pre-filling stage of LLMs. Our method leverages the intrinsic attention mechanisms of the LLM, thus being training-free and computationally efficient, achieving SoTA performance across mainstream benchmarks (see Table 1).

The attention mechanism is pivotal in transformer-based LLMs, aggregating information from input prompts via attention scores. In our `EHPC` method, we utilize high attention scores to identify and retain significant tokens. This approach is feasible primarily because the attention scores of tokens in long texts are sparse, explained by the widely studied "attention sink" phenomenon Xiao et al. [2023], Gu et al. [2024]. This phenomenon is characterized by the LLMs' frequently assigning high attention weights to the semantically inconsequential initial token, `<BOS>`. Furthermore, much of the research Zhang et al. [2023], Li et al. [2024], Ge et al. [2024a], Cai et al. [2024], Wu et al. [2024], Tang et al. [2024], Xiao et al. [2024], Fu et al. [2024] has focused on compressing the key-value (KV) cache by eliminating entries with low attention weights. Furthermore, `EHPC` can be efficient because we can leverage the KV cache obtained during the highly parallelizable pre-filling stage and quickly compute the required part attention scores[Jiang et al., 2024].

Table 1: Overall comparison of the proposed method in terms of average performance and latency on the LongBench dataset, under the constraint of a compressed prompt length of 2048 tokens. For comprehensive results, please see Table 4 and Table 5.

Method	Performance	Latency	Training-free
`LongLLMLingua`	48.0	67.44	✓
`LLMLingua`	34.6	7.51	✓
`LLMLingua-2`	39.1	1.27	✗
`EHPC` (*ours*)	**49.6**	**0.88**	✓

Unlike our prompt compression method, which retains important tokens based on the scores of evaluator heads, most KV cache compression methods preserve the cache according to all heads. Recent research [Wu et al., 2024, Tang et al., 2024, Xiao et al., 2024] has identified that certain layers and attention heads play a more significant role in processing long contexts. We have pinpointed a subset of these attention heads, which we designate as *evaluator heads*, allowing LLMs to focus on essential information for inference from any position within the input sequence. To identify these evaluator heads, we designed a pilot experiment using synthetic data (see Figure 1).

We then carried out extensive experiments to demonstrate the robustness of the evaluator heads and their applicability to practical scenarios using real-world datasets. Subsequently, we applied the evaluator heads to develop a prompt compression approach `EHPC`, using the scores given by these heads to select tokens for inference. The proposed `EHPC` requires the local development of LLMs for prompt compression and offers two application settings: Extended Model Inference (EMI) and Native Model Inference (NMI). We demonstrate its effectiveness through two important benchmarks: prompt compression and long text acceleration. When prompts compressed using `EHPC` are employed in commercial models, `EHPC` effectively reduces API costs while enhancing the performance of API outputs. Compared to existing prompt compression methods [Li et al., 2023, Jiang et al., 2023b,a], our approach achieves new state-of-the-art (SoTA) performance and is more efficient, requiring less compression time. Moreover, when applied to native models deployed locally, prompts compressed with `EHPC` accelerate long-text inference by reducing memory usage and achieving competitive results compared to SoTA KV cache compression methods [Li et al., 2024, Zhang et al., 2023].

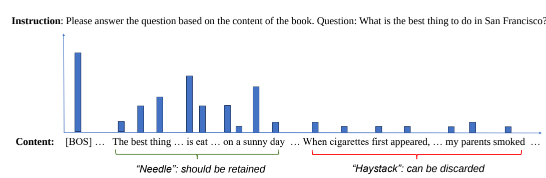

Figure 1: Visualization of attention scores from a single attention head during inference on the "Needle-in-a-Haystack" long-text benchmark. This benchmark requires the LLM to follow instructions and retrieve "needles" – specific pieces of information randomly inserted into a long text – to answer a given question. The evaluator heads are identified as those accurately locate the relevant facts, thereby achieving high scores.

Our contributions are as follows. **(1)** We identify specific attention heads in transformer-based LLMs, which we designate as *evaluator heads*, that are capable of selecting tokens in long inputs that are significant for inference. **(2)** We propose `EHPC`, an efficient prompt compression technique that enables LLMs to quickly "skim through" input prompts by utilizing only the first few layers with the evaluator heads, and then pass only the important tokens to the model for inference. **(3)** We demonstrate that `EHPC` has lower complexity compared to the prior prompt compression method and achieves a new SoTA on the prompt compression benchmarks over LongBench and ZeroScroll, effectively reducing the API cost and memory usage of commercial LLMs. **(4)** We empirically demonstrate that `EHPC` is capable of accelerating long-context understanding, achieving competitive performance relative to KV cache compression methods. Notably, `EHPC` improves upon direct inference by up to 40% on the question-answering datasets.

2 Related work

Two predominant approaches are utilized to accelerate LLMs: implicit methods that reduce the KV cache and explicit methods that decrease the number of tokens. We briefly review these methods with a focus on explicit methods, which can be applied to black-box LLMs such as GPT-3.5-Turbo.

2.1 Implicit token reduction

The key-value (KV) cache reduces redundant calculations and enhances decoding efficiency by storing key and value matrices from previous tokens [Liu et al., 2024a, Adnan et al., 2024]. However, as the input length increases, the memory requirements of the KV cache increase, creating significant challenges for long-context processing. It was found that a small number of tokens account for the majority of the value during the computation of the attention scores, leading to the proposal of the H2O [Zhang et al., 2023] that retains only the KV cache for "heavy hitters", which are tokens with high attention scores. FastGen [Ge et al., 2024a] introduces a dual-phase adaptive KV compression strategy that includes four KV cache compression policies and dynamically evicts caches during generation based on optimal policies identified through profiling. SnapKV Li et al. [2024] shows that specific patterns of attention can be identified by an observation window at the end of the prompts, and compresses KV caches by selecting clustered attention scores through pooling operations. Wu et al. [2024] experimentally investigate how transformer-based models retrieve relevant information from arbitrary locations within long contexts, identifying certain heads, termed *retrieval heads*, as crucial in this process. Subsequently, building upon the concept of *retrieval heads*, several head-wise KV cache compression methods have been proposed [Tang et al., 2024, Xiao et al., 2024]. These methods specifically preserve the KV cache of the retrieval heads to maintain their functionality.

2.2 Explicit prompt compression

Semantic compression Wingate et al. [2022] use soft prompts to condense the context, ensuring that the compressed prompts retain a significant amount of information. Chevalier et al. [2023] introduce AutoCompressor, which adapts LLMs to compress lengthy contexts into concise summary vectors. Similarly, research Bulatov et al. [2022], Mu et al. [2023], Ge et al. [2024b], Monteiro et al. [2024] has focused on learning various types of memory, such as gist tokens and cross-context caching, to compress context through prefix-tuning [Li and Liang, 2021]. Fei et al. [2024b] implement a summarization model to semantically compress input text through a divide-and-conquer strategy.

Token deletion A widely adopted approach among explicit methods is the direct removal of tokens Jha et al. [2024], Shi et al. [2024]. Selective-Context [Li et al., 2023] uses the logits of the model to calculate the mutual information of the tokens, subsequently removing tokens based on it. LLMLingua Jiang et al. [2023a] initially computes the perplexity of each token and then integrates a budget controller with a coarse-to-fine, token-level iterative compression algorithm. Expanding on LLMLingua, LongLLMLingua [Jiang et al., 2023b] introduces the concept of conditional perplexity to intensify the focus on key information according to task-specific instructions, which achieves great improvement over long text situations. LLMLingua-2 Pan et al. [2024] represents a fast prompt compression method, since it uses a small classification model to predict the significance of each token in the prompts. This classification model takes prompt compression as a token classification task and is trained to utilize a compact transformer-based encoder on a labeled dataset.

3 Methodology

We present our prompt compression method, EHPC, which is characterized by the identification and utilization of *evaluator heads*. We find that in LLMs, certain attention heads, which we designate as *evaluator heads*, can be utilized to rapidly determine which tokens can be omitted during the pre-filling stage. Background information on the basic implementation of the multi-head attention mechanism in LLMs, with a focus on the enhanced efficiency of the pre-filling stage through the application of KV cache, is provided in Appendix C.

3.1 Prompt compression strategy

We represent the input prompt as a sequence of tokens, $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$, where $N = |\boldsymbol{x}|$ denotes the length of the sequence. The objective of prompt compression is to identify a shorter sequence $\hat{\boldsymbol{x}}$ to replace the original sequence $\boldsymbol{x}$ for language models. Analogously to the way human readers often skip words during speed reading, EHPC employs a token deletion strategy as in [Li et al., 2023, Jiang et al., 2023a] and compresses the prompt by dropping non-essential tokens directly. In contrast to generating a new context through summarization Fei et al. [2024b], the token deletion strategy effectively simplifies the problem by narrowing the search space. Additionally, the token-level deletion strategy can be seamlessly integrated at the sentence/paragraph level [Liskavets et al., 2024].

3.2 Prompt compression using evaluator heads

Our prompt compression strategy selects salient tokens based on their averaged attention scores on the identified evaluator heads. Given a transformer-based language model f and the evaluator heads $\mathcal{C}_f$ identified through the pilot experiment delineated in Section 3.4. Let $\boldsymbol{A}^{hl} \in \mathbb{R}^{N\times N}$ denote the matrix of attention scores for the layer l and the attention head h using f. For a long input prompt $\boldsymbol{x}$, we utilize the attention scores from $\mathcal{C}_f$ to compute the *utility scores* $\boldsymbol{s} \in \mathbb{R}^N$ for the input during the pre-filling stage according to

$$\boldsymbol{s} = \sum_{(l_j,h_j)\in\mathcal{C}_f} \mathbf{Pool}\Big(\sum_{N_r\le i\le N} \frac{\boldsymbol{A}^{l_j,h_j}[i,:]}{N_o},\ r\Big), \tag{1}$$

where $\mathbf{Pool}(\cdot,\cdot)$ denotes a pooling operation, and r represents the kernel size, N_o is the observed window length and N_r is the length of remaining part such that $N = N_r + N_o$. Subsequently, we employ the scores $\boldsymbol{s}$ to remove non-essential tokens, constructing $\hat{\boldsymbol{x}}$ from the retained tokens in their original order. Although the compressed prompt retains its natural language form, it may lack certain contextual elements. To mitigate this, we adopt a pooling operation, as described in [Li et al., 2024], to group neighboring tokens with similar scores, thus generating a continuous sequence to enhance readability.

While our prompt compression strategy necessitates the processing of prompts by an LLM, it leverages the computationally efficient pre-filling stage, enabling fast compression. The compressed prompts obtained are transferable and can also be applied to black-box models based on API. Our EHPC method gives rise to two scenarios: Extended Model Inference (EMI) and Native Model Inference (NMI). EMI involves using a different language model f' to infer on the compressed prompt. For instance, when applied to an API-based commercial model, this approach can reduce API latency and costs, as the API cost is linearly related to the input prompt length. NMI, in contrast, utilizes the same model f to conduct inference. In this case, our method can reduce computational memory usage and costs, akin to the KV cache compression method. In practice, the language model f used for prompt compression should exhibit robust long-context capabilities, and smaller models are preferred for efficient deployments, such as `Llama-3.1-8B`, with a context length of 128k.

3.3 Evaluator heads

The evaluator heads retain important tokens according to the attention scores. As detailed in Appendix C, each attention head performs a weighted average over the preceding tokens, wherein the tokens assigned lower attention weights contribute less to the information processed by the attention heads. Empirical studies have shown that the contribution of attention heads to the capacity to handle long contexts in LLMs is not equally important Wu et al. [2024], Tang et al. [2024]. Specifically, certain retrieval heads are essential and must be maintained during KV cache compression, as their removal would significantly impair the LLMs' ability to manage long contexts. We find that specialized heads, which we designate as *evaluator heads*, play a pivotal role in assessing the significance of long input prompts, and these evaluator heads alone are sufficient for evaluating tokens.

We conducted experiments based on synthetic data with known evidence to explore and verify the existence of the *evaluator heads* and to ensure that they can effectively identify crucial information in input prompts. For a transformer model f with L layers and H heads in each layer, we define the evaluator heads as the set $\mathcal{C}_f = \{(l_1, h_1), \cdots, (l_m, h_m)\}$, where $1 \leq l_i \leq l_j \leq L$ and $1 \leq h_i \leq h_j \leq H$ for $i \leq j$. These specialized heads are identified, and their attention scores are used to give the final score that represents the utility of each token within the input prompts. We empirically investigate the properties of the evaluator heads, including the existence, generalizability, and robustness, in Section 4.

The primary distinction between the *evaluator heads* we have defined and the *retrieval heads* discussed in Section 2.1 lies in their respective functions: the evaluator heads are designed to assess the significance of tokens within input prompts, while the retrieval heads are intended to maintain the essential KV cache. Although both types of heads aim to identify the most salient components among all attention heads through a data-driven approach, evaluator heads are deemed sufficient for their purpose, whereas retrieval heads are necessary for preserving the integrity of the KV cache. Furthermore, evaluator heads operate explicitly on tokens, in contrast to *retrieval heads*, which function implicitly on the KV cache.

3.4 Pilot experiments for detecting evaluator heads

We design a pilot experiment to identify the evaluator heads. By entering a long prompt containing known evidence, we computed the corresponding accumulated scores for *evidence* to pinpoint the evaluator heads, where *evidence* is part of the input prompts that determine the model output. Let the relevant evidence $\boldsymbol{e}$ be denoted as the subsequence $\boldsymbol{e} = (\boldsymbol{x}_{I_{\boldsymbol{e}}})$ of $\boldsymbol{x}$, where $I_{\boldsymbol{e}} \subset [N]$ indicates the indices of the evidence. We extract the last row of the attention matrix, $\boldsymbol{a}^{hl} = \boldsymbol{A}^{hl}[N, :] \in \mathbb{R}^N$, to represent the scores for the importance of each token, as these scores directly influence the computation of the final hidden state. To assess whether the heads are focusing on relevant information, we compute the accumulated score of the evidence as

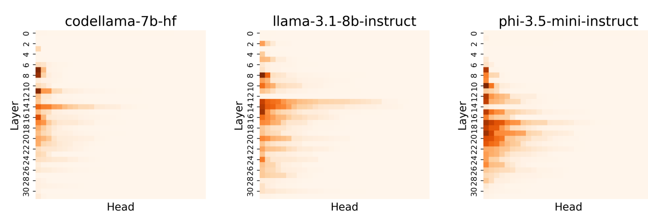

Figure 2: Heatmap of evidence scores for three different LLMs in the pilot experiment, illustrating scores across layers and heads, with heads re-ranked in descending order for clarity.

$$\hat{\boldsymbol{a}}^{hl} = \sum_{j \in I_{\boldsymbol{e}}} \boldsymbol{a}^{hl}[j]. \tag{2}$$

As a practical example, we used the "Needle-in-a-Haystack" benchmark Kamradt [2024], a well-established long-context retrieval benchmark, to demonstrate our pilot experiments (as illustrated in Figure 1). Let $\boldsymbol{x}$ be the synthesized long context and let $\boldsymbol{e}$ represent the "needle" (evidence) inserted at a specific position for f to identify. We record the accumulated score of the evidence $\hat{\boldsymbol{a}}^{hl}$ for each head. We then averaged the accumulated scores of the evidence to generate an *evidence score* matrix $S \in \mathbb{R}^{H \times L}$, which we used to identify the evaluator heads. Visualizations of the evidence score matrices are presented in Figure 2 for several examples of open-source LLMs. Finally, we selected the layer with the highest score and identified the top-k heads from this layer as the evaluator heads, i.e.,

$$\mathcal{C}_f = \underset{|\Lambda| \leq k}{\arg\max} \; \|\boldsymbol{e}_\Lambda S\|_{\mathrm{F}} \qquad \text{s.t.} \quad (i,j) \notin \lambda, i = \max_{1 \leq l \leq L} (S \cdot \mathbf{1}_{L \times 1})_l, \forall j,$$

where $\| \cdot \|_{\mathrm{F}}$ is the Frobenius norm, and $\boldsymbol{e}_\Lambda = (e_{ij})$ is the incidence matrix such that $e_{ij} = 0$ if $(i,j) \notin \Lambda$ and $e_{ij} = 1$ if $(i,j) \in \Lambda$.

3.5 Complexity

We only discuss the complexity of NMI, as EMI follows a similar line of reasoning. Consider an LLM f with L layers, H attention heads per layer, and a hidden dimension $d = d_k H$. Suppose that the model f is given an input prompt of N tokens and generates t new tokens. In the pre-filling stage, each head computes $\mathrm{Softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})$ with $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N \times d_k}$, resulting in a complexity of $O(d_k N^2)$. Thus, the total complexity for the pre-filling stage is $O(LHd_k N^2)$. During the decoding stage, the model generates t tokens based on the pre-filled $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times d_k}$, and the total complexity is $O(LHd_k(tN + \frac{t^2}{2}))$, with details provided in the appendix E.

In the NMI setting, where f is used for both compression and inference, our method involves two pre-filling stages and one decoding stage, as illustrated in Figure 3. Let $\kappa_1 = L/\max_{1 \leq l \leq L}(S \cdot \mathbf{1}_{L \times 1})_l$. The first pre-filling stage utilizes only L/κ_1 layers to compress the prompts, resulting in the complexity of $O(LHd_k N^2/\kappa_1)$. Suppose the compression rate is κ_2, then the second pre-filling stage processes only N/κ_2 tokens, and the complexity is $O(\frac{LHd_k N^2}{\kappa_2^2})$. Thus, the total complexity of combined pre-filling stages is $O(LHd_k N^2(\frac{1}{\kappa_1} + \frac{1}{\kappa_2^2}))$. Therefore, when $\kappa_1, \kappa_2 \geq 2$, which is often the case, the pre-filling with our compressing method has lower complexity since $(\frac{1}{\kappa_1} + \frac{1}{\kappa_2^2}) \leq \frac{3}{4}$. The decoding stage involves only N/κ_2 tokens, and the complexity is $O(LHd(t^2 + \frac{tN}{2\kappa_2}))$, which is naturally lower than the original complexity since $\kappa_2 \geq 1$.

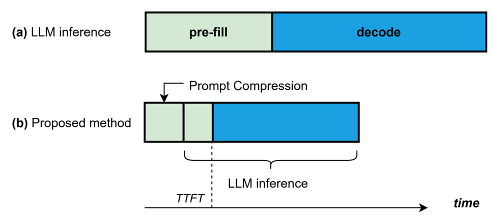

Figure 3: Illustration of the proposed method. **(a)** LLM inference comprises two stages: the pre-filling stage and the decoding stage. **(b)** The proposed prompt compression approach leverages the efficiency of the pre-filling stage, thereby reducing inference latency for both stages during inference with compressed context.

4 Properties of the evaluator heads

We conducted a comprehensive investigation to address the following research questions:

- **(RQ1) Existence:** Do evaluator heads exist significantly across LLMs? Can these heads be reliably identified through a pilot experiment utilizing simple synthetic data?
- **(RQ2) Generalizability:** Can the evaluator heads identified through synthetic data be effectively applied to various tasks in real-world long-text benchmarks?
- **(RQ3) Robustness:** Are evaluator heads task-oriented, i.e., do these heads induced by different tasks demonstrate robustness and consistency?

To investigate (RQ1), (RQ2), and (RQ3), we conducted pilot experiments to identify the evaluation heads using a synthetic benchmark in the "Needle-in-a-Haystack" style. Then, we evaluate these evaluator heads using real-world benchmarks, including LongBench [Bai et al., 2024] and ∞Bench [Zhang et al., 2024b]. Details on the synthetic and real-world benchmarks used in our study are provided in Appendix H.

Existence To illustrate the existence of these heads, we conducted pilot experiments as detailed in Section 3.4, utilizing the widely employed "Needle-in-a-Haystack" benchmark. We provide the results over three models in Figure 2. We evaluated each head's ability to identify key information by examining the accumulated attention scores over the evidence sequence. The distribution of these evaluator heads is sparse and predominantly concentrated in some certain layers. This provides empirical support to find evaluator heads in one early and significant layer.

Generalizability Next, we assess the generalizability of the evaluator heads identified in our pilot experiments by evaluating their performance on diverse downstream tasks. The results over two models are presented in Table 2. We selected 8 heads with the highest scores as evaluator heads from the layer with the highest cumulative score. We also randomly selected 8 heads and 50% heads

Table 2: Performance comparison of two sets of heads on three datasets from different tasks in the LongBench.

Model	Method	NarQA	Musique	LCC	QMSum	MathFind	CodeDebug
Llama 3.1	Random (8 heads)	16.6	10.4	45.2	20.3	13.7	14.2
	Random (50% heads)	19.6	13.2	47.7	23.7	14.9	14.7
	Ours (8 heads)	23.0	14.0	48.9	24.2	27.7	24.7
Phi	Random (8 heads)	12.2	12.0	28.9	18.1	26.0	9.8
	Random (50% heads)	24.4	11.7	40.6	22.3	21.4	16.1
	Ours (8 heads)	23.9	20.3	41.6	22.2	34.0	16.2

for comparison. It's observed that our selected 8 heads have significant improvement for randomly selected 8 heads. Though random 50% achieve comparable performance (still worse than ours), they require significantly more time to evaluate tokens. The findings highlight the practical utility of the evaluator heads and demonstrate the effectiveness of our detection experiments.

Robustness Lastly, we empirically illustrate that detecting the evaluator heads through a pilot experiment is task-agnostic. In addition to the simple QA data from "Needle-in-a-Haystack", we also generate probe data from the following tasks: code completion (code) and multi-hop variable tracking (reasoning). Details on the construction of the probe data are provided in Appendix J. We then compare the heads from the simple QA data with the task-aware heads corresponding to their respective reasoning and coding tasks from the LongBench dataset. The results are presented in Table 3. The results are averaged over the complete data. "Evaluator heads (QA)" refers to the heads identified using QA data from the pilot experiment, while "Task-aware heads" refers to heads identified from generated data customized to the respective tasks. Our findings indicate that the heads identified using customized tasks do not have improvement on their respective downstream tasks. This shows that the evaluator heads are task-agnostic and that the heads identified from the simple QA data exhibit robustness in downstream tasks.

Table 3: Evaluation of task-aware approach on multi-hop reasoning and code completion from corresponding LongBench datasets.

	Multi-hop	Code
Evaluator heads (QA)	16.40	47.86
Task-aware heads	16.87	47.72

5 Experimental results

Having established the existence, generalizability, and robustness of the evaluator heads, we now experimentally assess the effectiveness and efficiency of `EHPC` in tasks aimed at reducing API costs of commercial models and accelerating long-context inference in LLMs. We also compare the efficiency of our prompt compression method with other acceleration methods under the same memory usage.

5.1 Prompt compression benchmark

We evaluated the prompt compression benchmark on LongBench and ZeroSCROLLS, as established by Jiang et al. [2023b], with the objective of improving the quality of compressed prompts for commercial models. Prompt compression is essential for mitigating costs associated with commercial LLMs, given that API fees are often proportional to prompt length. We compare our EMI setting, which employs a local LLM for prompt compression and another model for inferring the compressed prompts, against the following baselines.

Baselines and implementation details We compare our method with Retrieval Augmented Generation (RAG) and other SoTA prompt compression methods. The retrieval models considered include BM25, SentenceBERT [Reimers and Gurevych, 2019], and OpenAI Embedding. For prompt compression methods, we evaluated against Selective-Context [Li et al., 2023], LLMLingua [Jiang et al., 2023a], LongLLMLingua [Jiang et al., 2023b], and LLMLingua-2 [Pan et al., 2024] as baselines. Detailed descriptions of these prompt compression baselines are provided in Appendix I. LongLLMLingua extends LLMLingua by incorporating task-specific information to enhance long-context compression, while LLMLingua-2 improves efficiency through the use of a smaller model. To ensure

Table 4: Performance of various prompt compression methods under different compressed length constraints on LongBench and ZeroSCROLLS. Higher values indicate better performance. The best scores are highlighted in **boldface**.

Methods	LongBench									ZeroSCROLLS		
	SingleDoc	MultiDoc	Summ.	FewShot	Synth.	Code	Avg.	# Tokens	κ_2	Avg.	# Tokens	κ_2
Original Prompt	39.7	38.7	26.5	67.0	37.8	54.2	44.0	10,295	-	32.5	9,788	-
Zero-shot	15.6	31.3	15.6	40.7	1.6	36.2	23.5	214	48×	10.8	32	306×
3,000 tokens constraint												
Retrieval-based Methods												
BM25	32.3	34.3	25.3	57.9	45.1	48.9	40.6	3,417	3×	19.8	3,379	3×
SBERT	35.3	37.4	26.7	63.4	51.0	34.5	41.4	3,399	3×	24.0	3,340	3×
OpenAI	34.5	38.6	26.8	63.4	49.6	37.6	41.7	3,421	3×	22.4	3,362	3×
Compression-based Methods												
Selective-Context	23.3	39.2	25.0	23.8	27.5	53.1	32.0	3,328	3×	20.7	3,460	3×
LLMLingua	31.8	37.5	26.2	67.2	8.3	53.2	37.4	3,421	3×	30.7	3,366	3×
LLMLingua-2	35.5	38.7	26.3	69.6	21.4	62.8	42.2	3,392	3×	33.5	3,206	3×
LongLLMLingua	40.7	46.2	**27.2**	**70.6**	53.0	55.2	48.8	3,283	3×	32.8	3,412	3×
`EHPC (EMI)`	**44.2**	**49.1**	25.1	68.8	**54.0**	**63.0**	**49.7**	2,892	3×	**36.7**	3,005	3×
2,000 tokens constraint												
Retrieval-based Methods												
BM25	30.1	29.4	21.2	19.5	12.4	29.1	23.6	1,985	5×	20.1	1,799	5×
SBERT	33.8	35.9	25.9	23.5	18.0	17.8	25.8	1,947	5×	20.5	1,773	6×
OpenAI	34.3	36.3	24.7	32.4	26.3	24.8	29.8	1,991	5×	20.6	1,784	5×
Compression-based Methods												
Selective-Context	16.2	34.8	24.4	15.7	8.4	49.2	24.8	1,925	5×	19.4	1,865	5×
LLMLingua	22.4	32.1	24.5	61.2	10.4	56.8	34.6	1,950	5×	27.2	1,862	5×
LLMLingua-2	29.8	33.1	25.3	66.4	21.3	58.9	39.1	1,954	5×	33.4	1,898	5×
LongLLMLingua	39.0	42.2	**27.4**	**69.3**	**53.8**	56.6	48.0	1,809	6×	32.5	1,753	6×
`EHPC (EMI)`	**44.5**	**50.7**	24.8	68.9	51.5	**61.9**	**49.6**	2,004	5×	**34.6**	2,041	5×

Table 5: Averaged time (in seconds) for different methods applied to a subset of LongBench.

Method \ Latency	Compression time	Inference time (2048 tokens)	**Total**
LLMLingua	7.51	**1.09**	8.60
LongLLMLingua	67.44	1.31	68.74
LLMLingua-2	1.27	1.15	2.37
`EHPC` (*ours*)	**0.88**	1.11	**1.99**
`ChatGPT-3.5-Turbo` (all tokens)			2.16

reproducibility, we employ greedy decoding and set the temperature to 0 in all experiments. For prompt compression, we utilize the `Llama-3.1-8B` model. Additional details of the hyperparameters are provided in Appendix D.

Results The results of the prompt compression benchmarks are presented in Table 4. Consistent with previous research [Jiang et al., 2023b], we report the averaged results for ZeroSCROLLS and LongBench, as well as the average performance for the sub-tasks on LongBench, with target compressed prompt lengths of $2,000$ and $3,000$ tokens. Table 4 illustrates that our method achieves a superior performance on average on both benchmarks under length constraints. The results show that `EHPC` exceeds the original prompts and significantly enhances the accuracy of the QA tasks, indicating its effectiveness in the retrieval tasks by alleviating the disturbance of the noisy context. In particular, our model performs well on code tasks in the LongBench, which can be attributed to the identification of the key tokens involved in the inference of coding tasks. Furthermore, the performance of `EHPC` in summarization and few-shot learning tasks is competitive. Overall, our compression method achieves high-quality compression, even outperforming the original prompts on specific tasks.

Compression latency We evaluated the running times of various methods using a subset of LongBench, including direct inference, LLMLingua-2, LongLLMLingua, and our proposed approach. The results are presented in Table 5, where the targeting compression is 2,048 tokens. The subset

Table 6: Performance of different acceleration methods over various LLMs under the same KV cache memory usage. Higher values indicate better performance. We simplified some datasets for easier presentation The best scores are **boldfaced**.

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Average
	NarrativeQA	Qasper	MultiQA-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	RepoBench-P	LCC	
LLaMA 3.1 8B Instruct																	
All KV	32.02	13.04	27.34	16.23	16.05	11.22	34.52	23.41	26.89	73.0	91.64	43.8	7.16	97.73	49.25	52.7	38.32
H2O-1024	21.98	10.96	23.17	16.06	15.16	10.15	**30.76**	23.75	**26.32**	**68.0**	90.68	42.54	7.40	71.34	**51.6**	46.4	34.77
SnapKV-1024	**31.98**	11.17	25.33	14.81	15.73	10.69	26.95	22.89	25.86	67.5	**91.89**	**42.85**	**7.67**	**98.16**	48.7	**52.1**	**35.25**
GemFilter-1024	20.71	11.00	**29.28**	19.12	**17.01**	13.01	30.37	21.75	25.17	63.0	90.70	42.50	7.15	92.22	35.0	38.5	34.50
EHPC (*ours*)-1024	22.98	**13.02**	27.41	**20.64**	16.97	**13.99**	29.49	**24.15**	25.24	**68.0**	86.52	41.86	5.79	97.21	42.8	48.9	35.23
H2O-2048	23.16	11.68	25.09	16.17	15.22	9.93	32.13	23.32	**26.73**	68.5	91.32	**43.97**	6.52	72.79	**52.5**	48.1	35.45
SnapKV-2048	**31.45**	11.94	26.24	15.73	16.03	11.66	29.64	23.24	26.44	69.5	91.48	42.68	7.21	98.03	49.5	**52.6**	35.80
GemFilter-2048	24.36	12.63	25.39	**19.58**	**17.03**	**14.11**	**33.15**	22.31	26.49	69.5	**91.59**	42.64	4.61	**98.75**	38.8	47.3	35.87
EHPC (*ours*)-2048	20.98	**14.38**	**30.50**	19.35	16.23	13.02	32.9	**24.54**	26.72	**71.0**	90.47	42.24	**9.35**	97.94	44.9	52.2	**37.86**
Phi 3.5 Mini 3.8B Instruct																	
All KV	27.51	17.23	35.63	21.70	25.70	11.68	34.14	23.17	24.95	71.5	87.37	13.08	7.17	83.85	46.0	46.2	34.62
H2O-1024	18.25	12.97	29.69	20.75	20.90	9.90	**32.02**	21.69	**24.70**	67.5	85.45	20.16	1.37	46.80	**46.6**	43.3	31.38
SnapKV-1024	**24.31**	16.03	34.93	20.72	26.02	13.74	28.27	22.03	24.02	67.5	**87.71**	14.57	**6.08**	**85.60**	44.0	**47.1**	33.68
GemFilter-1024	16.57	18.29	35.91	24.22	26.10	9.70	30.29	18.96	23.64	64.5	85.85	23.02	0.20	81.12	39.0	40.7	32.74
EHPC (*ours*)-1024	23.91	**32.22**	**45.36**	**44.97**	**32.79**	**20.27**	31.90	**22.19**	23.77	**68.5**	85.72	**36.69**	1.80	79.08	38.7	41.6	**39.22**
H2O-2048	18.26	14.05	32.4	20.03	22.51	10.30	32.86	21.43	**24.90**	67.2	86.44	19.65	1.43	46.96	**47.89**	44.71	31.94
SnapKV-2048	26.41	16.59	36.99	21.80	**26.07**	12.57	30.88	22.37	24.51	**69.5**	87.54	13.13	**6.57**	**83.92**	45.30	46.70	34.20
GemFilter-2048	19.63	14.84	35.99	21.38	19.72	10.13	32.39	21.24	24.71	65.0	86.49	20.47	2.17	69.50	46.30	**48.10**	31.69
EHPC (*ours*)-2048	**24.80**	**39.27**	**39.78**	**27.06**	25.22	**14.05**	**33.26**	**23.52**	24.48	68.0	**87.66**	**37.77**	2.54	63.00	47.31	45.44	**36.46**

contains 10 examples from RepoBench-P, with an average of 14,354 tokens. Each example was repeated 5 times to reduce randomness. Specifically, `ChatGPT-3.5-Turbo` (all tokens) serves as the baseline using the original prompts, where the inference time is carried over from [Liskavets et al., 2024]. Each prompt compression method reduces the average prompt length from 14,354 tokens to 2,048 tokens. The experiments were carried out on a GPU with 48GB of VRAM. For a fair comparison, we used the same language model, `Llama-3.1-8B`, for all methods except LLMLingua-2, which requires a specialized model. The results in Table 5 indicate that our method is significantly faster than LongLLMLingua and also outperforms LLMLingua-2, which is known for its efficiency. The lower latency of our compression strategy is attributed to its reliance on the efficient pre-filling stage.

5.2 Acceleration of LLM inference

We demonstrate that EHPC effectively reduces memory overhead and computation costs during long-context inference of LLMs by compressing input prompts. By inferring over the κ_2 times compressed prompt, the KV cache during pre-filling is reduced κ_2 times, also accelerating the decoding stage. In the NMI setting, where the same model is used for both compression and inference, the acceleration achieved is comparable to that of KV cache compression methods with the same compression ratio. Therefore, we compare our method against the KV cache compression method on the LongBench long-text acceleration benchmark.

Baselines We consider acceleration frameworks that include KV cache compression and prompt compression as baselines. For KV cache compression methods, we select SoTA approaches, including H2O [Zhang et al., 2023] and SnapKV [Li et al., 2024], which eliminate unnecessary KV caches based on attention scores. For the prompt compression method, we select GemFilter Shi et al. [2024], which utilizes attention scores of early layers detected by downstream tasks to compress the prompts.

Implementation detail We utilize two popular long-context models: `Llama-3.1-8B` and `Phi-3.8B`, both support a context window length of 128k. For each method, we set the target lengths to 1024 and 2048 tokens for prompt compression and KV cache compression, respectively.

Results In Table 6, we present the results of long text inference acceleration under KV cache constraints of 1024 and 2048 tokens using various methods. EHPC achieves the best average performance compared to the baselines across different compression rates. Relative to KV cache-based methods, our prompt compression method performs particularly well on specific tasks such as QA, even outperforming the results obtained using the full KV cache. This improvement is particularly significant when applying our method to the `Phi-3.8B` model, enhancing its performance on direct inference by an average of up to 40% in QA tasks, including Single-Document QA and Multi-Document QA. However, prompt compression exhibits a decline in performance on code-related tasks and few-shot learning tasks. Although the effectiveness of the KV cache compression method gradually decreases as the KV cache memory is reduced, it remains more advantageous than prompt compression for these tasks. Overall, `EHPC` demonstrates competitive results compared to the KV cache compression method, particularly in QA tasks.

6 Conclusion

We introduced `EHPC`, an efficient and effective prompt compression method that utilizes evaluator heads to leverage attention scores from the pre-filling stage of transformer-based large language models. Unlike previous speedup methods that rely on training specialized small models, our approach is training-free and achieves state-of-the-art performance in prompt compression. We implement our evaluator head-based prompt compression (`EHPC`) method in two settings: native model inference and extended model inference. We demonstrate the effectiveness of `EHPC` in these settings in two main benchmarks, highlighting its potential to significantly reduce API costs for commercial use and accelerate long text inference.

References

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024a.

Weizhi Fei, Xueyan Niu, Guoqing Xie, Yanhua Zhang, Bo Bai, Lei Deng, and Wei Han. Retrieval meets reasoning: Dynamic in-context editing for long-text understanding. *arXiv preprint arXiv:2406.12331*, 2024a.

Simranjit Singh, Andreas Karatzas, Michael Fore, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. An LLM-Tool compiler for fused parallel function calling. *arXiv preprint arXiv:2405.17438*, 2024.

Yanan Chen, Ali Pesaranghader, Tanmana Sadhu, and Dong Hoon Yi. Can we rely on LLM agents to draft long-horizon plans? let's take travelplanner as an example. *arXiv preprint arXiv:2408.06318*, 2024.

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

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, 2023a.

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. *arXiv preprint arXiv:2310.06839*, 2023b.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. When attention sink emerges in language models: An empirical view. *arXiv preprint arXiv:2410.10781*, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b155725//87b78ecfcebb2827f8-Paper-Conference.pdf.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=poE54GOq2l.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=uNrFpDPMyo.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient KV cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. Lazyllm: Dynamic token pruning for efficient long context LLM inference. *arXiv preprint arXiv:2407.14057*, 2024.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024a.

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: KV cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5621–5634, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.412. URL https://aclanthology.org/2022.findings-emnlp.412.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. *ArXiv*, abs/2305.14788, 2023. URL https://api.semanticscholar.org/CorpusID:258865249.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

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

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

Joao Monteiro, Étienne Marcotte, Pierre-Andre Noel, Valentina Zantedeschi, David Vazquez, Nicolas Chapados, Christopher Pal, and Perouz Taslakian. XC-cache: Cross-attending to cached context for efficient LLM inference. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15284–15302, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.896. URL https://aclanthology.org/2024.findings-emnlp.896/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Weizhi Fei, Xueyan Niu, Pingyi Zhou, Lu Hou, Bo Bai, Lei Deng, and Wei Han. Extending context window of large language models via semantic compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5169–5181. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.findings-acl.306.

Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Kurt Keutzer, and Amir Gholami. Characterizing prompt compression methods for long context inference. *arXiv preprint arXiv:2407.08892*, 2024.

Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. Discovering the gems in early layers: Accelerating long-context LLMs with 1000x input token reduction. *arXiv preprint arXiv:2409.17422*, 2024.

Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane Luke. Prompt compression with context-aware sentence encoding for fast and improved LLM inference. *arXiv preprint arXiv:2409.01227*, 2024.

Greg Kamradt. Needle In A Haystack - Pressure Testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2024.

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.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.814.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *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 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. *arXiv preprint arXiv:2401.09670*, 2024.

Junlin Lv, Yuan Feng, Xike Xie, Xin Jia, Qirong Peng, and Guiming Xie. Critiprefill: A segment-wise criticality-based approach for prefilling acceleration in llms. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

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

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, 2021.

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

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31.

Jiaxin Bai, Zihao Wang, Yukun Zhou, Hang Yin, Weizhi Fei, Qi Hu, Zheye Deng, Jiayang Cheng, Tianshi Zheng, Hong Ting Tsang, et al. Top ten challenges towards agentic neural graph databases. *arXiv preprint arXiv:2501.14224*, 2025.

Zihao Wang, Weizhi Fei, Hang Yin, Yangqiu Song, Ginny Y Wong, and Simon See. Wasserstein-Fisher-Rao Embedding: Logical Query Embeddings with Local Comparison and Global Transport. *arXiv preprint arXiv:2305.04034*, 2023.

Hang Yin, Zihao Wang, Weizhi Fei, and Yangqiu Song. Efok-cqa: Towards knowledge graph complex query answering beyond set operation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 5876–5887, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3737426. URL https://doi.org/10.1145/3711896.3737426.

Hang Yin, Zihao Wang, and Yangqiu Song. Rethinking complex queries on knowledge graphs with neural link predictors. *arXiv preprint arXiv:2304.07063*, 2023.

Weizhi Fei, Zihao Wang, Hang Yin, Yang Duan, and Yangqiu Song. Extending complex logical queries on uncertain knowledge graphs. *arXiv preprint arXiv:2403.01508*, 2024c.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association* *for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL `https://aclanthology.org/2021.naacl-main.112`.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. Qmsum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921, 2021.

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084, 2019.

Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, 2023. URL `https://api.semanticscholar.org/CorpusID:259262301`.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems, 2024b. URL `https://arxiv.org/abs/2306.03091`.

Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. ZeroSCROLLS: A zero-shot benchmark for long text understanding. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7977–7989, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.536. URL `https://aclanthology.org/2023.findings-emnlp.536`.

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: The paper proposes EHPC, a training-free method for improving long-context inference efficiency in transformer-based LLMs. EHPC identifies evaluator heads, a subset of attention heads that select the most relevant tokens during the pre-filling stage of inference. By leveraging these heads, the method compresses prompts before inference, reducing API costs and latency while maintaining performance. The main claims in the abstract and introduction reflect these contributions.

   Guidelines:

   - The answer NA 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 NA 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: The limitations of the theorems are discussed throughout the paper, and a general discussion of limitations can be found in Appendix A.

   Guidelines:

   - The answer NA 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: [Yes]

   Justification: The complexity analysis of EHPC is given in Section 3.5 and Appendix E.

   Guidelines:

   - The answer NA 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: All the information needed to reproduce the numerical experiments is given in Sections 3 and 4. In Section 5, we outline details for implementation and the baselines. Additionally, we plan to make our code open source at the time of publication to facilitate reproducibility.

   Guidelines:

   - The answer NA 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: [Yes]

   Justification: Yes, the code is provided in the supplementary material with instructions to reproduce the experiments.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The benchmarks are discussed in Appendix H. In Appendix D we provide details for hyperparameters, and in Section 5 we outline details for implementation.

   Guidelines:

   - The answer NA 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: To make our statements, we set target compressed prompt lengths and performed the evaluation on different datasets. Considering the computational resources required, the error bar is not provided in this paper, and this method of reporting results is common.

Guidelines:

- The answer NA 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: The compute resources used for the experiments are described in Section 5.

   Guidelines:

   - The answer NA 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: We reviewed the NeurIPS Code of Ethics and the paper conforms with it.

   Guidelines:

   - The answer NA 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: The broader impacts are discussed in Appendix A.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: EHPC is an efficient prompt compression method, and no specific models or datasets are employed.

    Guidelines:

    - The answer NA 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: All assets used in this paper are publicly available and properly credited.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: This paper is about a new method without new assets.

    Guidelines:

    - The answer NA 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: [NA]

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

    Guidelines:

    - The answer NA 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: [NA]

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

    Guidelines:

    - The answer NA 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 rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: This paper is about prompt compression using LLMs. The LLMs used in the paper are publicly available. Our method is described in Section 3 and throughout the paper.

    Guidelines:

    - The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Limitations and broader impacts

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 specifically highlighted here. The implementation of prompt compression methods in machine learning can have significant societal impacts. One major concern is the risk of information distortion, where critical nuances may be lost in the simplification process, potentially leading to misinterpretations. This can result in biased decision making, as compressed prompts may not convey the full context necessary for accurate judgments. Furthermore, reliance on these techniques can create communication barriers, particularly between individuals of diverse backgrounds, thereby hindering effective dialogue. Additionally, those skilled in using prompt compression may gain an unfair advantage in information processing, exacerbating knowledge inequality. In general, while prompt compression can enhance efficiency, it is crucial to consider its implications for transparency, inclusivity, and critical thinking within society.

A limitation of our work is that due to the LLMs considered, our results are naturally only applicable to transformer architectures. Specifically, we empirically showed that identified evaluator heads consistently attend to key information in long inputs. We investigate their existence, generalizability, and robustness using both synthetic and real-world data. More rigorous theoretical analysis should be performed on the capacity of the evaluator heads.

B Algorithm

Require: Input prompt $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$, and transformer-based LLM f.
Ensure: Output compressed prompt $\hat{\boldsymbol{x}} = (x_{i_1}, x_{i_2}, \ldots, x_{i_M})$ with M tokens.
Detect the evaluator heads at layer l_e, $\mathcal{C}_f = \{l_j, h_j\}$ by the pilot experiments.
Get $\boldsymbol{A}^{l_j,h_j}$ during the pre-filling stage at layer l_e layer.
Get the scores by evaluator head: $\boldsymbol{s}_1 = \sum_{N_r \le i \le N} \frac{\boldsymbol{A}^{l_j,h_j}[i,:]}{N_o}$, where N_o is the observed window and $N_r = N - N_o$.
Calculate the pooled scores: $\boldsymbol{s}_2 = \mathbf{Pool}(\boldsymbol{s}_1,\ r)$
Select the indices with high scores: $J \leftarrow \mathsf{topk\_index}(\boldsymbol{s}_2, k = M)$
Sort the indices according to prompt order: $\hat{J} = \mathsf{sort}(J) = \{i_1, \cdots, i_M\}$.

Here, we provide the pseudocode flow of our method in Algorithms B.

C Background

Multi-head attention Transformer-based models autoregressively predict the next token based on the τ preceding tokens according to

$$\boldsymbol{h}^l = \boldsymbol{h}^{l-1} + \boldsymbol{\delta}^l + \boldsymbol{h}^l,\ m^l = \mathrm{FFN}(\boldsymbol{h}^{l-1} + \boldsymbol{\delta}^l)$$

where $\boldsymbol{h}^l, \boldsymbol{m}^l, \boldsymbol{\delta}^l \in \mathbb{R}^d$ are the hidden states of the l-th layer and $\mathrm{FFN}(\cdot)$ denotes the feed forward network. Transformer models typically utilize multi-head attention, so that the hidden state $\boldsymbol{\delta}^l$ at layer l is computed as

$$\boldsymbol{\delta}^l = \boldsymbol{W}_a^l \, \mathrm{ConCat}(\hat{\boldsymbol{h}}^{l1}, \cdots, \hat{\boldsymbol{h}}^{lH}),$$

where H is the number of heads in each layer, and $\hat{h}^{lh}$ denotes the hidden representation of the h-th head at layer l. The attention operation in each attention head is

$$\hat{\boldsymbol{h}}^{lh} = \mathrm{Softmax}(\frac{\boldsymbol{Q}^{lh}(\boldsymbol{K}^{lh})^T}{\sqrt{d_k}}) \cdot \boldsymbol{V}^{lh}, \tag{3}$$

where $\boldsymbol{Q}^{lh}, \boldsymbol{K}^{lh}, \boldsymbol{V}^{lh} \in \mathbb{R}^{\tau \times d_k}$, and $d_k = d/H$ is the dimension of each head. As shown in Eq. (3), the attention mechanism aggregates information and selects important tokens from input prompts. The weights calculated by the dot product between queries and keys determine which tokens of the values are considered within this attention block. The independent multi-head attention mechanism enables the model to capture information from previous tokens in multiple ways.

LLM inference and KV cache compression Contemporary decoder-only LLMs generating new tokens with a series of tokens as input involve two stages: the pre-filling stage and the decoding stage. During the pre-filling stage, the model processes compute the intermediate states (keys and values in attention operations), which are highly parallelized and computationally efficient. In the decoding stage, LLMs load the precomputed KV cache and generate each output token autoregressively through a forward pass, which is slower due to its serial nature Zhong et al. [2024].

When processing long text input, the size of the corresponding KV cache increases dramatically, significantly raising both computational costs and time. To address this issue, KV cache compressing methods have been proposed that eliminate unnecessary KV caches. Many KV cache compression methods are based on attention weights to propose a policy to determine which tokens to retain in memory. Let $\boldsymbol{A}^{lh} = \mathrm{Softmax}(\frac{\boldsymbol{Q}^{lh}(\boldsymbol{K}^{lh})^T}{\sqrt{d_k}})$ represent the attention matrix. The policy first averages the last rows to represent the scores of input tokens, as follows:

$$\boldsymbol{s} = \sum_{N_r \leq i \leq N} \frac{\boldsymbol{A}[i,:]}{N_o}, \tag{4}$$

where N denote the number of tokens in the prompt, N_o is the observed windows length and N_r is the length of remaining part, such that $N = N_r + N_o$. To further improve the contextual integrity, Li et al. [2024] apply the pooling operation to perform clustering:

$$\hat{\boldsymbol{s}} = \mathbf{Pool}(\boldsymbol{s}, k), \tag{5}$$

where $\mathbf{Pool}(\cdot,\cdot)$ represents a pooling operation such as $\max(\cdot)$ and $\mathrm{Average}(\cdot)$, and k is the kernel size. This trick ensures that the identified tokens are continuous rather than isolated, resulting in more coherent semantics.

D Hyper-parameters

In this section, we introduce the hyperparameter for reproduction. We first introduce our detected evaluator heads and then introduce the hyperparameter to use these heads.

Detected evaluator heads We conducted pilot experiments using the "Needle-in-a-Haystack" benchmark across three popular LLMs: `Llama-3.1-8B-Instruct`, `CodeLlama-7B`, and `Phi-3.5-mini-3.8B-Instruct`.

- For `Llama-3.1-8B-Instruct`, which has 32 layers and 32 heads, the selected layer is 13, and the chosen heads are $[18, 13, 21, 8, 11, 1, 4, 3]$.
- For `CodeLlama-7B`, also with 32 layers and 32 heads, the selected layer is 14, and the selected heads are $[24, 3, 18, 7, 29, 2, 9, 1]$.
- Finally, for `Phi-3.5-mini-3.8B-Instruct`, which features 32 layers and 32 heads, the selected layer is 17, and the chosen heads are $[7, 17, 30, 2, 6, 16, 25, 18]$.

Using evaluator heads The hyperparameters applied during the evaluation of heads include the size of the observed windows, the pooling operation, and the kernel size for pooling. In all experiments, we used the average pooling operation, as the difference between average pooling and maximum pooling was experimentally negligible. For `Llama-3.1-8B-Instruct`, we set the size of the observed windows and the kernel size for pooling to 16 and 32, respectively. For `Phi-3.5-mini-3.8B-Instruct`, the size of the observed windows and the kernel size for pooling were set to 4 and 32, respectively. A larger kernel size typically results in a more continuous compressed context, which is why we generally prefer to use a larger kernel size.

E Complexity

We further discuss the complexity of NMI in this section. Consider an LLM f with L layers, H attention heads per layer, and a hidden dimension $d = d_k H$. Suppose that the model f receives an input prompt of N tokens and generates t new tokens. In the pre-filling stage, each head computes $\mathrm{Softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})$ with $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N \times d_k}$, resulting in a complexity of $O(d_k N^2)$. Thus, the total

Table 7: Speedup Ratios for Different Sequence Lengths Compared to Direct Inference in Pre-filling and Decoding Stages

		Length			
Model	**Phase**	8k	16k	24k	32k
Llama	Pre-fill	1.6×	1.8×	1.9×	2.0×
-	Decode	2.6×	4.9×	4.0×	10.3×
Phi	Pre-fill	1.7×	1.9×	OOM	OOM
-	Decode	1.2×	1.5×	OOM	OOM

complexity for the pre-filling stage is $O(LHd_kN^2)$. During the decoding stage, the model generates t tokens based on the pre-cached $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N \times d_k}$. At the $i+1$ tokens generated, the softmax operation in each head involved $\boldsymbol{Q} \in \mathbb{R}^{1 \times d_k}, \boldsymbol{V} \in \mathbb{R}^{(N+i) \times d_k}$ and the complexity is $O(d_k(N+i))$. to calculate the total complexity for generating t tokens, we sum the complexities across all t tokens:

$$O(\sum_{i=1}^{t-1} d_k(N+i)) = O(d_k(Nt + \frac{(t-1)^2}{2})) = O(d_k(Nt + \frac{t^2}{2})). \tag{6}$$

Thus, the overall complexity for generating t tokens during the decoding stage is $O(LHd_k(Nt+t^2))$. Regarding the complexity of EMI, it remains similar if the inference model is still transformer-based.

F Speedup ratios

We present the EHPC speedup ratios compared to direct inference in Table 7. We evaluated the time taken for Needle-in-a-Haystack with sequence lengths ranging from 8k to 32k, running five examples for each length. We directly take the pre-filling time for pre-fill and take the time of decode per token for decode. We also compute the speed ratio of our method with direct inference. The results demonstrate that our method effectively improves the pre-filling stage. In the decoding stage, the speedup ratio increases with length in the Llama model. In the Phi model, the baseline encounters out-of-memory issues at 16k, while our method can successfully infer 32k sequences.

G Additional results for NMI (local model)

We provide the experiment results over ∞Bench [Zhang et al., 2024b] in Table 8, with Minference and SnapKV as baselines. Due to the Minference method encountering OOM errors with 32k long sequences on the Phi model, we primarily selected 32k as the truncation length. Additionally, we provide experimental results of our method on the Phi model with 128k length to demonstrate that our model can effectively infer such long sequences.

G.1 Extremely Long Benchmark

Table 8: Additional evaluation over ∞Bench compared with Minference and SnapKV

Model	Truncate	Method	Passkey	NumStr	LongBook Sum	CodeDebug	LongDialog QA	LongBook QA	LongBook Choice	MathFind	KV Retrieval	Avg
Llama	-	SnapKV	27.1	26.6	26.5	2.3	7.0	14.8	56.3	21.3	5.0	20.8
-	32k	Minference	27.1	27.1	24.7	17.0	11.0	9.2	43.2	27.7	17.8	22.8
-	-	EHPC	27.1	27.1	18.8	27.4	3.0	16.9	45.9	22.3	17.6	22.9
-	128k	EHPC	100.0	99.8	16.8	22.6	5.5	18.0	49.3	26.9	32.0	41.2
Phi	-	SnapKV	27.1	10.3	23.4	15.5	7.3	9.5	43.2	30.6	0.4	18.6
-	32k	Minference	27.1	27.1	29.0	3.6	7.5	14.5	56.3	36.0	25.2	25.1
-	-	EHPC	27.1	27.1	24.6	16.2	8.0	11.2	55.5	34.0	25.8	25.5

Table 9: Comparison of Minference and CritePrefill on LongBench Benchmark

Model	Method	NarrativeQA	Qasper	MFQA-en	HotpotQA	2WikiMHQA	MuSiQue	TREC	TriviaQA	SAMSum	PCount	PRe	LCC	Re
Phi	CritePrefill	22.83	23.33	29.66	4.72	31.70	20.42	66.50	78.70	-	7.0	57.00	33.70	30.10
	Minference	-	14.13	34.94	18.81	21.26	9.80	70.00	85.54	16.43	-	-	-	-
	Our (1024)	23.91	32.22	45.36	44.97	32.79	20.27	68.50	85.72	36.69	1.80	79.08	38.67	41.57

G.2 Comparison with additional baselines

We present the results of Minference [Jiang et al., 2024] and CritePrefill [Lv et al., 2025] on LongBench. Due to the extensive decoding time required for long tokens in CritePrefill, we have removed datasets that involve summaries.

H Datasets

We present the synthetic and real-world benchmarks for assessing the capability of LLMs in handling long contexts.

H.1 Synthetic benchmarks

Needle-in-a-Haystack Kamradt [2024] is a well-known synthetic dataset used to benchmark the ability to navigate long contexts. It involves randomly inserting a sentence into a variable-length long context, followed by querying a given LLM to retrieve that specific sentence from a long context.

Ruler Hsieh et al. [2024] is a synthetic long-context benchmark that extends "Needle-in-a-Haystack" by offering more complex tasks. RULER encompasses diverse types and quantities of needles and introduces new task categories, such as multi-hop tracing and aggregation, to evaluate behaviors beyond direct searching within the context.

H.2 Real-world benchmarks

LongBench Bai et al. [2024] is a widely used long-context benchmark that includes 21 datasets across six types of tasks. The six tasks are single-document question answering, multi-document question answering, summarization, few-shot learning, code completion, and synthetic tasks for retrieval and counting. In line with previous research Jiang et al. [2023a,b], we focus on English datasets and encompass six tasks across 16 datasets. We present a detailed introduction in the following context.

Single-Doc QA

- **NarrativeQA** [Kočiskỳ et al., 2018] is a standard question-answering dataset that includes texts from Project Gutenberg and movie screenplays sourced from various websites. It contains 200 entries and is evaluated using the F1 metric.
- **Qasper** [Dasigi et al., 2021] is a question-answering dataset focused on NLP publications, featuring abstractive, extractive, and yes/no questions. It consists of 200 entries and is evaluated using the F1 metric.
- **MultiFieldQA-en** [Bai et al., 2024] is created from diverse sources, including legal documents, government reports, encyclopedias, and academic publications. It includes 150 entries and is evaluated using the F1 metric.

Multi-Doc QA

- **HotpotQA** [Yang et al., 2018] features many 2-hop questions crafted by native speakers, based on two related paragraphs. It contains 200 entries and is evaluated using the F1 metric.
- **2WikiMultihopQA** [Ho et al., 2020] includes up to 5-hop questions systematically constructed with manual templates. Answering these questions requires reasoning paths that cannot be resolved by local content alone. It contains 200 entries and is evaluated using the F1 metric.

- **MuSiQue** [Trivedi et al., 2022] consists of up to 4-hop questions, eliminating shortcuts and questions about naturalness. Each question includes 2-4 supplementary paragraphs that outline the reasoning path and relevant content. It comprises 200 entries and is evaluated using the F1 metric. [2]

Summarization

- **GovReport** [Huang et al., 2021] collects comprehensive reports containing human-written summaries from the U.S. Government Accountability Office and Congressional Research Service, covering a wide array of national policy issues. It includes 200 entries and is evaluated using the Rouge-L metric.
- **QMSum** [Zhong et al., 2021] contains annotated pairs of meeting summaries across various domains, including product, academic, and committee meetings. It includes 200 entries and is evaluated using the Rouge-L metric.
- **MultiNews** [Fabbri et al., 2019] is a multi-document summarization dataset that clusters 2-10 news articles discussing the same event or topic, each paired with a human-written summary, thus forming a new long-text summarization task. It includes 200 entries and is evaluated using the Rouge-L metric.

Few-Shot Learning To construct few-shot learning with long text, [Bai et al., 2024] selected a range of training examples from the following datasets to concatenate the context in LongBench:

- **TREC** [Li and Roth, 2002] is a classification dataset featuring fine-grained class labels. It includes 200 entries and is evaluated using the accuracy metric.
- **TriviaQA** [Zhong et al., 2021] is another classification dataset that involves messenger-like conversations accompanied by human-written summaries. It contains 200 entries and is evaluated using the F1 metric.
- **SAMSum** [Fabbri et al., 2019] is a reading comprehension dataset consisting of question-answer pairs annotated with evidence passages. It includes 200 entries and is evaluated using the Rouge-L metric.

Code Completion Code completion is a critical yet challenging task utilized by auto-completion systems to assist users in predicting and completing code based on previous inputs and context.

- **LCC** [Guo et al., 2023]: Sampled from the Long Code Completion dataset, this dataset is constructed by filtering code based on length within individual GitHub files. It incorporates preceding lines of code as context, with the next line serving as the answer. This dataset includes 200 entries and is evaluated using the Exact Match (EM) metric.
- **RepoBench-P** [Liu et al., 2024b]: Collected from GitHub repositories, this dataset aggregates relevant cross-file code snippets based on module import statements. These snippets are combined with preceding lines of code in the current file to predict the next line of code, utilizing the most challenging XF-F setting. It comprises 200 entries and is evaluated using the Exact Match (EM) metric.

Synthetic Task Two synthetic datasets evaluate the ability to retrieve and count from long contexts.

- **PassageRetrieval-en**: Derived from English Wikipedia, this dataset randomly samples 30 passages and selects one for summarization, with the task of identifying the original paragraph corresponding to the summary. It includes 500 entries and is evaluated using the Edit Similarity metric.
- **PassageCount**: This dataset presents a more complex challenge by randomly selecting paragraphs from English Wikipedia, repeating, and shuffling them. The model is required to determine the number of unique passages among the provided set. It consists of 500 entries and is evaluated using the Edit Similarity metric.

[2] Complex query answering [Bai et al., 2025, Wang et al., 2023] is another kinds of reasoning tasks, including various types of questions [Yin et al., 2025, 2023] and context [Fei et al., 2024c].

ZeroSCROLLS [Shaham et al., 2023] is a well-known long-context benchmark that encompasses three types of tasks: summarization, question answering, and aggregation across ten datasets. In line with prior research Jiang et al. [2023a,b], we focus on the validation set for evaluation, as it is the only set providing ground truth data. Below, we introduce the ten datasets across the three tasks, using the same evaluation metric for each dataset.

Summarization

- **GovReport**: Contains long reports from the Congressional Research Service and U.S. Government Accountability Office, paired with expert-written summaries.
- **SummScreenFD**: Comprises episode scripts from TV shows with community-contributed recaps sourced from Wikipedia and TVMaze.
- **QMSum**: A query-based summarization dataset featuring meeting transcripts, including academic, industrial, and parliamentary discussions, with each instance accompanied by a specific query.
- **SQuALITY**: A question-focused dataset derived from Project Gutenberg stories, requiring summaries based on crowdsourced guiding questions.

Question Answering

- **Qasper**: Contains NLP papers from the Semantic Scholar Open Research Corpus, with questions based on abstracts answered by practitioners.
- **NarrativeQA**: Features questions and answers derived from books and movie scripts, with questions crafted from summaries provided by annotators.
- **QuALITY**: Comprises stories and articles requiring multiple-choice questions that necessitate reading substantial portions for accurate answers.
- **MuSiQue**: Focuses on multi-hop questions using Wikipedia paragraphs, including both answerable and unanswerable questions.

Aggregation

- **SpaceDigest**: A sentiment aggregation task using 50 hotel reviews per hotel from the Space dataset, focusing on strictly positive or negative reviews.
- **BookSumSort**: A task based on the BookSum dataset, requiring the reordering of shuffled chapter summaries from selected books to their original order.

∞Bench [Zhang et al., 2024b] We provide an overview of the various tasks included in our study, detailing their contexts, example counts, and average token statistics.

En.Sum This task involves summarizing a fake book created through core entity substitution. It consists of 103 examples, with an average of 171.5k input tokens and 1.1k output tokens.

En.QA This task focuses on free-form question answering based on the content of the fake book. It includes 351 examples, with an average of 192.6k input tokens and 4.8 output tokens.

En.MC This task consists of multiple-choice questions derived from the fake book. It contains 229 examples, averaging 184.4k input tokens and 5.3 output tokens.

En.Dia This task involves identifying speakers in partially anonymized scripts. It includes 200 examples, with an average of 103.6k input tokens and 3.4 output tokens.

Zh.QA This task entails question answering on a set of newly collected books. It consists of 175 examples, with an average of 2068.6k input tokens and 6.3 output tokens.

Code.Debug This task focuses on identifying which function in a code repository contains a crashing error, presented in multiple-choice format. It includes 394 examples, averaging 114.7k input tokens and 4.8 output tokens.

Code.Run This task simulates the execution of multiple simple, synthetic functions. It contains 400 examples, with an average of 75.2k input tokens and 1.3 output tokens.

Math.Calc This task involves performing calculations that include extremely long arithmetic equations. It consists of 50 examples, averaging 43.9k input tokens and 43.9k output tokens.

Math.Find This task focuses on finding special integers within a lengthy list. It includes 350 examples, with an average of 87.9k input tokens and 1.3 output tokens.

Retrieve.PassKey1 This task involves retrieving hidden keys in a noisy long context. It consists of 590 examples, averaging 122.4k input tokens and 2.0 output tokens.

Retrieve.Number This task focuses on locating repeated hidden numbers in a noisy long context. It includes 590 examples, with an average of 122.4k input tokens and 4.0 output tokens.

Retrieve.KV2 This task involves finding the corresponding value from a dictionary based on a given key. It consists of 500 examples, averaging 89.9k input tokens and 22.7 output tokens.

I Baselines of prompt compression

Selective-Context [Li et al., 2023] applies the logits of a causal language model to compute the self-information for each token, word, or sentence, subsequently eliminating unnecessary content based on this self-information. To the best of our knowledge, this is the first work to study compressed prompts using pre-trained language models.

LLMLingua [Jiang et al., 2023a] divides the target prompt into several segments and allocates different compression budgets according to the perplexity distribution of these segments. Compressing a long prompt is modeled as a scheduling problem, utilizing the perplexity of each segment to allocate budgets effectively.

LongLLMLingua [Jiang et al., 2023b] further incorporates task information (such as questions for document QA) and employs a Question-Aware strategy to enhance the density of key information in long contexts. LongLLMLingua introduces conditional perplexity based on task information and aims to improve information density relative to the task, building on the foundation established by LLMLingua [Jiang et al., 2023b].

LLMLingua-2 [Pan et al., 2024] is based on smaller, fine-tuned BERT models to improve efficiency, leveraging global information from an extractive text compression dataset annotated by ChatGPT. Using a smaller model for prompt compression, we have found that this is, to the best of our knowledge, the fastest prompt compression method available.

J Additional details for evaluator heads

In this section, we describe how we construct task-aware synthetic data to identify task-aware heads. We focus on two types of tasks: multi-hop reasoning and code completion. For multi-hop reasoning, we utilize the multi-hop tracing task from the Ruler, which is an extension of "Needle-in-a-Haystack". Multi-hop tracing involves randomly inserting several interconnected chains to assess how effectively the model tracks all the content of these chains in response to given questions. This forms the basis of a multi-hop reasoning task. Regarding the code completion task, we first created a long code dataset that is unrelated to LongBench to prevent data leakage. We then manually inserted the code that needs completion into the context, effectively generating the necessary evidence. These two types of data are designed to identify specific heads within their respective domains using synthetic data tailored to those domains.