FINCH: Prompt-guided Key-Value Cache Compression for Large Language Models

Giulio Corallo
SAP Labs, France
EURECOM, France
giulio.corallo@sap.com

Paolo Papotti
EURECOM, France
papotti@eurecom.fr

Abstract

Recent large language model applications, such as Retrieval-Augmented Generation and chatbots, have led to an increased need to process longer input contexts. However, this requirement is hampered by inherent limitations. Architecturally, models are constrained by a context window defined during training. Additionally, processing extensive texts requires substantial GPU memory. We propose a novel approach, FINCH, to compress the input context by leveraging the pre-trained model weights of the self-attention. Given a prompt and a long text, FINCH iteratively identifies the most relevant Key (K) and Value (V) pairs over chunks of the text conditioned on the prompt. Only such pairs are stored in the KV cache, which, within the space constrained by the context window, ultimately contains a compressed version of the long text. Our proposal enables models to consume large inputs even with high compression (up to 93x) while preserving semantic integrity without the need for fine-tuning.

1 Introduction

Large Language Models (LLMs), built upon the Transformer architecture, have delivered breakthroughs in numerous applications. With their generalization and reasoning capabilities, models such as ChatGPT have revolutionized fields where extensive input prompts are necessary for generating precise responses, such as Retrieval-Augmented Generation, Chain-of-Thought, conversational chatbots, and In-Context Learning (Lewis et al., 2020; Wei et al., 2022; Dong et al., 2022). However, the use of LLMs in production is limited by their increasing requests in terms of GPU memory (Dettmers et al., 2023). First, as the computational complexity grows along with the size of the models, their memory consumption increases. Second, this issue becomes more pronounced when LLMs process larger inputs, as demanded by their ever-increasing context size. Third, the Key-Value (KV) cache mechanism, typically employed by LLMs to speed up the generation process, prioritizes efficiency by retaining and reusing previously computed KV vectors during attention computation, bypassing re-calculations at each token generation step (Kaiser et al., 2017). Nevertheless, this solution comes with the trade-off of increased memory consumption.[1] To offer more efficient solutions to operate these models, it has been proposed to *compress* input prompts, exploiting the redundancy in natural language (Goyal et al., 2020). By preserving critical token information while compressing less crucial details, these models reduce the context in a compact description, without noticeably degrading the functional accuracy (Mu et al., 2023). Compression also enables the LLMs to process large inputs that do not fit the model's context size. However, most of these models require a training/fine-tuning process or a large number of calls to an external model for the compression (Jiang et al., 2023b). We revisit the LLMs' generative inference mechanism to deal with the memory constraint problem and the limitations of current solutions in processing large inputs. We propose a novel approach targeting the reduction of the KV cache memory footprint while avoiding resource-intensive retraining or fine-tuning processes. Drawing insights from the patterns inherent in attention modules, and guided by the understanding that not all attention modules engage with every token, our solution compresses the cached vectors, leading to a reduction in memory usage and efficient text generation.

Our approach, termed FINCH,[2] facilitates faster generative inference through adaptive KV cache

[1] It has been reported that OPT-175B (with batch size 128 and sequence length 2048) consumes 325 GB of memory, but its KV cache requires 950 GB (Liu et al., 2023b).

[2] *Finch* is a small and quick bird, known for its chirp—a complex language for a small animal.

Transactions of the Association for Computational Linguistics, vol. 12, pp. 1517–1532, 2024. https://doi.org/10.1162/tacl_a_00716
Action Editor: Tao Ge. Submission batch: 3/2024; Revision batch: 6/2024; Published 11/2024.

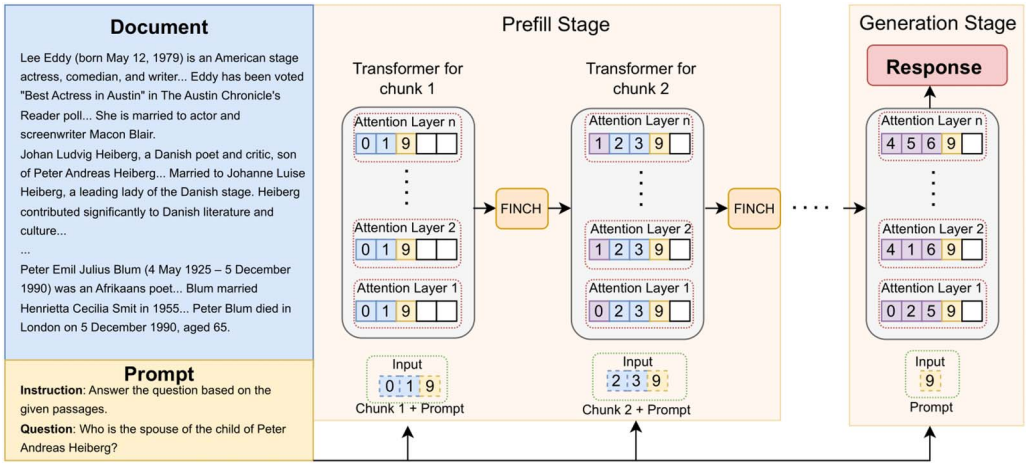

Figure 1: Overview of FINCH. An input document is larger than the model context and thus is processed in chunks. At each step in the Prefill stage, FINCH sequentially consumes a document chunk (two dashed border squares, blue background), alongside the input prompt (one dashed border square, yellow background) as depicted at the bottom. At each step, it processes the key, value pairs (solid squares in the transformer) and identifies the most relevant to the prompt. It then carries them to the cache processing the next chunk (where they appear with a violet background). In the Generation stage, the model synthesizes a response that is informed by the compressed cached information from the entire document. The white square is the space reserved for producing output tokens in the Generation stage.

compression in the Prefill stage. Figure 1 shows how a long document and the input prompt are processed with a model context size that cannot fit the entire input. At every step, a document chunk is processed. FINCH uses the attention information between the prompt and the document chunk to identify the most relevant KV pairs across different layers. This information then is stored in the KV cache for the processing of the next input chunk. Our approach dynamically selects what to keep in the KV cache's memory, effectively keeping its footprint constrained, until the Generation stage produces the response.

FINCH incrementally feeds the KV cache with the compressed context without any learning or external summarization module; it can be used in a plug-and-play fashion with any decoder-based model. The compression rate is specified by setting the target size of the KV cache as an input parameter constrained by the model context size. Even with high compression ratios, our method ensures that the correctness of the model response is preserved. We test FINCH on two popular benchmarks covering tasks in question answering, summarization, code completion, synthetic tasks and few-shot learning. Compared against the original LLM (without compression) over the SQuAD v2 benchmark (Rajpurkar et al., 2018), FINCH achieves comparable generation quality at 2.35x compression and 90% of the reference accuracy score at 3.76x compression, while being faster in terms of end-to-end execution times in most cases. When compared to the state-of-the-art compression method LongLLMLingua (Jiang et al., 2024) FINCH reports the best quality scores in most of the tasks in LongBench (Bai et al., 2024), both with Llama 2 and Mistral (Touvron et al., 2023b; Jiang et al., 2023a). Our method achieves a compression range of 2x to 93x across various tasks, consistently outperforming a truncation baseline in most experiments. Remarkably, FINCH even surpasses the performance of the LLMs operating with the full, uncompressed context in certain cases. Finally, in question answering tasks, we also include a RAG baseline, and our method outperforms it in 10 out of 12 experiments.

2 Related Work

We position our work w.r.t. two main topics. First, we discuss strategies for improving computational efficiency, i.e., making LLMs accessible for real-time applications or use on devices with limited resources. Second, we focus on attention patterns in LLMs, as our work shows that those contribute significantly towards optimizing the models to handle larger inputs in a limited context size.

Efficiency Improvements in LLMs. Methods targeting the reduction of inference and fine-tuning costs include model modification, such as quantization (Frantar et al., 2023; Dettmers et al., 2022) and model compression (Frantar and Alistarh, 2023). Other efforts enhance model efficiency for LLMs by eliminating redundant input words based on attention scores (Goyal et al., 2020) and compressing the input sequence by augmenting the encoding modules with pooling layers (Dai et al., 2020). Proposed solutions also involve learning to skip layers in the transformer architecture (Guan et al., 2022; Zhou et al., 2020) or to select the most critical tokens for performance (Huang et al., 2022). Other approaches pursue prompt compression, either by limiting the number of tokens that are processed in inference by learning special ''compressed'' tokens (Mu et al., 2023; Wingate et al., 2022; Ge et al., 2024b) or by pruning and merging tokens (Goyal et al., 2020; Modarressi et al., 2022), e.g., learning thresholds for pruning unimportant ones (Kim et al., 2022). However, some of these strategies require an additional re-training or fine-tuning phase and others have been designed for encoder models and are not well suited for auto-regressive LLMs such as ChatGPT and Llama (Touvron et al., 2023a,b). In contrast with such solutions, our approach condenses auto-regressive LLMs input contexts during the Prefill stage by using the caching mechanism without model re-training and even faster inference. Finally, recent methods focus on optimizing the generation stage to improve efficiency (Zhang et al., 2023; Xiao et al., 2024; Han et al., 2024; Oren et al., 2024; Ren and Zhu, 2024). We leave to future work the study of how to use our prompt-guided token selection strategy in such approaches.

The Role of Attention. Our work relies on self-attention to make the most relevant information in a context available in a concise manner. The development of transformer models provoked studies to unravel the underlying mechanisms of self-attention, e.g., heads prominently pay attention to separator and adjacent tokens (Clark et al., 2019). Our solution capitalizes on the attention mechanism structure to heighten inference efficiency by exploring the KV cache for the most important key, value pairs w.r.t. the given prompt. Related work evaluates the informativeness of lexical units using a language model and drops less informative content for compression (Li, 2023; Jiang et al., 2023b, 2024), for example, by regarding tokens with lower perplexity as more influential in the inference process. These techniques view LLMs as a compressor for world knowledge and work by further compressing information within prompts (Deletang et al., 2024). In contrast with these solutions, our approach instead optimizes the management of the KV cache during the Prefill stage without requiring a separate LLM. Other approaches look at how to select the most important tokens in the Prefill stage, but, differently from our method that dynamically identifies the most important tokens, they rely on manually defined policies for token selection (Ge et al., 2024a).

Finally, we focus on a plug-and-play solution for existing models, with an emphasis on limited computing resources. This is in contrast with other solutions that demand more devices to handle a very large input context (Liu et al., 2023a).

3 Background

Self-attention is foundational in transformer models (Vaswani et al., 2017), enabling language understanding and generation capabilities. Transformers learn the contextual relationships between words or subwords within a sentence. Central to this mechanism are three types of vectors—Queries ($\mathbf{Q}$), Keys ($\mathbf{K}$), and Values ($\mathbf{V}$)—that are learned from the input embeddings.

- **Queries ($\mathbf{Q}$)**: Represent the current word or token being processed, acting as a point of focus.
- **Keys ($\mathbf{K}$)**: Serve as identifiers, highlighting tokens in the sequence relevant to the query.
- **Values ($\mathbf{V}$)**: Correspond to the actual specific information carried by each token.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

The self-attention mechanism computes the dot product of the **Query** with all **Keys** to determine their similarity. A softmax function normalizes these scores, creating a distribution that determines how much attention to allocate to each token. The output is a weighted sum of the **Values**.

In several NLP tasks, transformers generate a response sequence from a given context/document and a user prompt. Consider a sequence of tokens representing the context $\mathbf{x}^{\text{cont}} \in \mathbb{R}^{n^{\text{cont}}}$ and a sequence of tokens representing the user prompt $\mathbf{x}^{\text{que}} \in \mathbb{R}^{n^{\text{que}}}$, which may also include instructions, the goal is to enable the model to generate a response sequence $\mathbf{y} \in \mathbb{R}^{a}$. This process can be divided into two stages.

Prefill Stage. As a first step, both the context and the prompt sequence are concatenated to form the input sequence $\mathbf{x} \in \mathbb{R}^{n}$, where:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}^{\text{cont}} \\ \mathbf{x}^{\text{que}} \end{bmatrix}, \quad \text{and} \quad n = n^{\text{cont}} + n^{\text{que}}$$

This sequence is then embedded into an embedding matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where d denotes the embedding dimension and processed through multiple layers of multi-head self-attention and feed-forward networks, which operate in parallel across the sequence length and attention heads. Each attention layer calculates and stores the corresponding Key and Value $\mathbf{K} \in \mathbb{R}^{n \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$ matrices in a cache for the sake of performance for the subsequent Generation stage. In the transformer architectures, the $\mathbf{K}$ and $\mathbf{V}$ matrices encapsulate historical token information. Unlike other components of the transformer (e.g., feedforward layer or layer norm), which process current inputs independently of their past tokens, the $\mathbf{K}$ and $\mathbf{V}$ matrices are the only matrices that retain information from previously encountered tokens. Caching the Key and Value matrices for every layer for the context eliminates the necessity to recompute them for each new token generated.

Generation Stage. In this step, the model iteratively generates new tokens. For each new token, the $\mathbf{q}^{\text{new}}, \mathbf{k}^{\text{new}}, \mathbf{v}^{\text{new}} \in \mathbb{R}^{d}$ are produced at each attention layer with the $\mathbf{k}^{\text{new}}$ and $\mathbf{v}^{\text{new}}$ vectors appended to the existing cache keys and values:

$$\mathbf{K} \leftarrow \begin{bmatrix} \mathbf{K} \\ \mathbf{k}^{\text{new}} \end{bmatrix}, \quad \mathbf{V} \leftarrow \begin{bmatrix} \mathbf{V} \\ \mathbf{v}^{\text{new}} \end{bmatrix}$$

Self-attention uses this updated cache to compute attention. Thanks to the stored $\mathbf{K}$ and $\mathbf{V}$ matrices, the computational complexity is just $O(nd)$ as opposed to the approach without cache, which has a computational complexity of $O(n^2 d)$. Finally, logits are generated and used to predict the next token in the vocabulary, e.g., with greedy decoding (Vijayakumar et al., 2016; Shao et al., 2017).

4 Problem Formulation

As discussed, K and V are the only matrices that retain information about previous tokens. We can therefore formulate the problem of compression as reducing the size of these two matrices during the Prefill stage and before the actual answer generation takes place. Specifically, we have to find $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{V}}$ where $\tilde{\mathbf{K}}, \tilde{\mathbf{V}} \in \mathbb{R}^{k \times d}$ such that two properties are satisfied:

- **Compression**: the *target tokens size* of the compressed $\tilde{\mathbf{K}}, \tilde{\mathbf{V}}$ matrices should be smaller than the sequence length n^{cont} of $\mathbf{K}^{\text{cont}}, \mathbf{V}^{\text{cont}} \in \mathbb{R}^{n^{\text{cont}} \times d}$.
- **Information retention**: the output $\mathbf{y} \in \mathbb{R}^{a}$ using $\mathbf{K}, \mathbf{V}$ matrices is similar to the output $\tilde{\mathbf{y}} \in \mathbb{R}^{a}$ obtained using $\tilde{\mathbf{K}}, \tilde{\mathbf{V}}$, expressed as:

$$\min_{\tilde{\mathbf{K}}, \tilde{\mathbf{V}}} f(\tilde{\mathbf{y}}, \mathbf{y}) \tag{1}$$

where f is a distance function and its choice depends on the task at hand. For example, in question answering, the difference between F1 scores for $\tilde{\mathbf{y}}$ and $\mathbf{y}$ might be used.

We also define the compression ratio σ as:

$$\sigma = \frac{n^{\text{cont}}}{k}$$

In this work, we compress the context $\mathbf{K}^{\text{cont}}$, $\mathbf{V}^{\text{cont}}$ matrices, according to the target tokens size k, while conditioning on the user prompt. This decision is driven by the recognition that the integrity of the user prompt—particularly its instructions for an instruction-tuned model—plays a significant role in the answer generation (Ouyang et al., 2022). Furthermore, in the tasks that we address in this work, the prompt is typically much shorter than the context, making its compression of limited value.

5 Method

Our approach aims at compressing contexts into a manageable form for LLMs, particularly when faced with extensive documents and the need to maintain computational efficiency. Our methodology is motivated by the following observation: The softmax of self-attention distributes attention across all elements to varying degrees, effectively capturing a spectrum of contextual relationships in the data. We hypothesize that the ''smooth'' distribution of attention may include superfluous information for the given prompt at hand.

5.1 Adaptive Key-Value Cache Compression

As depicted in Figure 1, FINCH iteratively processes a document segmented into chunks, each evaluated in conjunction with a user prompt, and uses the self-attention to identify which K,V pairs to keep in the cache. In analogy to the long-term memory involving the capacity to recall words, concepts, or numbers (Chauvet, 2024), we say that these pairs can act as the *semantic memory* for the model. The document is reduced to its significant elements and processed in the Generation stage.

Document Segmentation. The transformer input is constrained by a context window defined during training, denoted as n_{max}. Given the user specified target tokens size k for the KV cache, FINCH processes chunks using at most $m_{\text{max}} = n_{max} - k$ tokens.[3] The input document is partitioned into chunks of size m, which value is constrained by m_{max}. At every Prefill step i, for $i > 1$, the K,V pairs from the previous step $i - 1$ (the compressed chunk) are added into the tokens reserved for the k target tokens.

This process introduces a trade-off between granularity and throughput. Smaller chunks enable finer granularity in processing, which is beneficial for certain tasks as we highlight in Section 7. Conversely, larger chunks (up to m_{max}) enhance throughput by reducing the number of sequential operations required, thus speeding up the Prefill stage. This trade-off is crucial for optimizing performance and is examined in our ablation study.

Prompt-Guided Layer-wise top-r position selection. Our method for selecting the top r (relevant) positions is rooted in the analysis of the attention scores across its layers. We take into account the unique role of each layer for the representation of the input, i.e., early layers might focus on syntactic features, while deeper layers might capture more abstract, semantic relationships (Clark et al., 2019). As a consequence, for each layer of the transformer, we calculate attention scores (the scaled dot-product attention between Q and K) and determine the context per-token relevance of the chunk with respect to tokens in the user prompt. By acknowledging that relevance varies by layer, we ensure a more holistic compression of the document. For example, tokens that are relevant in early layers might be not relevant in deeper layers. This allows our method to preserve a wide spectrum of information without redundancy.

Our method also takes into consideration the inherent positional bias present in the attention mechanism. In particular, causal language models operate in the principle that each token in a sequence can only be influenced by preceding tokens, not by those that come after it. This is visually represented by a triangular matrix in attention mechanism, where the ability of tokens to ''attend'' to each other is constrained by their position in the sequence. As a result, early tokens in a sentence have a broader scope of attention compared to later tokens. For example, for the first token, its attention score is maximal since it only considers itself, leading to a score of 1. To address the issue that later tokens in the sequence, which could be equally or more relevant to the question, are not overlooked due to systemic bias, we incorporate a normalization step that adjusts the raw attention scores to mitigate positional bias, ensuring that each token's relevance is equally evaluated.

Consider $\mathbf{A}^{(l)} \in \mathbb{R}^{H \times M \times N}$ as the attention scores matrix at layer l, with H attention heads. Here, M and N are defined as:

$$M = m + n^{\text{que}}, N = m + n^{\text{que}} + c$$

where m is the chunk length and c is the current KV cache length. The compression process involves several steps as visualized in Figure 2.

- **Sum over Heads**: Every Head in a transformer attention layer captures various aspects of the

[3] We ignore the user prompt size in this discussion as we assume it to be much smaller than the input document size.

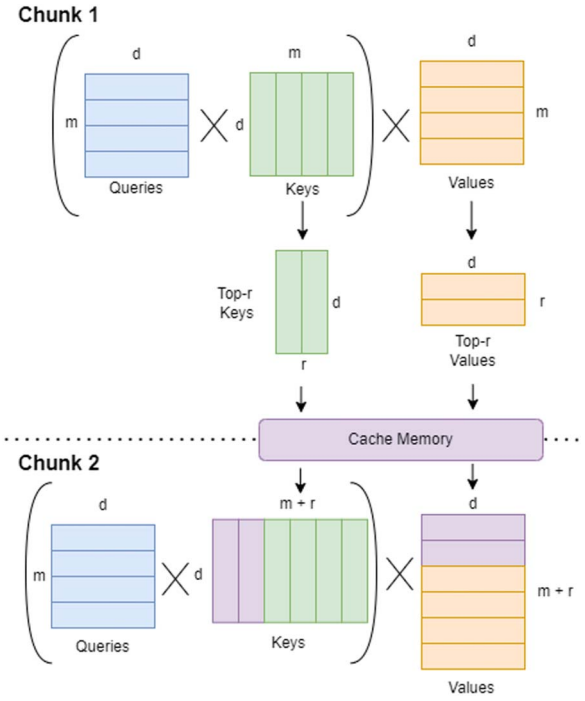

Figure 2: Our attention computation process. In the top portion, the initial chunk of length m is processed to identify the top r keys,values pairs through the dot product of queries and keys. The top r elements are then stored in cache memory. As the second chunk undergoes processing (bottom), new keys and values are generated and both the current chunk of length m and the top r elements of the previous iteration are considered for the subsequent top r selection.

data. We sum the attention scores over the heads to aggregate their contributions, The elements $A_{ij}^{(l)\text{sum}}$ of $\mathbf{A}^{(l)\text{sum}}$ are defined as:

$$A_{ij}^{(l)sum} = \sum_{h=1}^{H} A_{hij}^{(l)sum}$$
$$\forall i \in \{1,\ldots,M\}, j \in \{1,\ldots,N\}$$

- **Extract Prompt-guided Submatrix**: A submatrix is extracted to focus on the attention scores between prompt tokens and the current document chunk, this includes considering the tokens accumulated in the KV cache, which grows with each iteration:

$$A_{i,j}^{(l)\text{cont}} = A_{m+i,j}^{(l)\text{sum}}$$
$$\forall i \in \{1,\ldots,n^{\text{que}}\}, j \in \{1,\ldots,m+c\}$$

Figure 3 shows how attention scores for the last layer of Llama 2 evolve in the sequential operations.

- **Normalization**: Attention scores are normalized to mitigate positional bias, adjusting for non-zero attention scores:

$$A^{(l)\text{norm}} = A^{(l)\text{cont}} \cdot \left(\frac{\text{count}(A^{(l)\,\text{cont}} \neq 0)}{m+c} \right)$$

- **Selection of Top r Position**: The final step is to select the top r indices based on the aggregated attention scores over the prompt tokens.

$$A_i^{(l)\text{agg}} = \sum_{p=1}^{n^{\text{que}}} A_{p,i}^{(l)\text{norm}} \quad \forall i \in \{1,\ldots,m+c\}$$

$$\mathbf{t} = \text{top-r}(\mathbf{A}^{(l)\text{agg}}, r)$$

here, $\mathbf{t}$ is a vector containing indices of the top r positions with the highest attention scores. The parameter r dynamically updates at each iteration based on the chunk size m, cache length c, and compression rate σ. Specifically, the update rule is given by:

$$r_{it+1} = \frac{m_{it+1}}{\sigma} + c_{it}$$

where it denotes the iteration. At the final iteration, r corresponds to the target token size k.

Managing the Cache: The key, value pairs for the selected top r positions are preserved within the KV cache due to their significant relevance to the user prompt. This process involves an adjustment to their positional embeddings. To accurately reflect the tokens' relative positions, we draw inspiration from the mechanisms used in Attention sinks (Xiao et al., 2024). For example, given a cache sequence [0, 1, 2, 3, 4, 5] and a relevance ranking [3, 5, 0], we prioritize '3' by moving it three positions to the left, '5' by moving it four positions to the left, and '0' by shifting it two positions to the right, while the others are discarded. For Rotary Position Embeddings (Su et al., 2024), as in Llama 2, this repositioning involves calculating the cosine and sine required

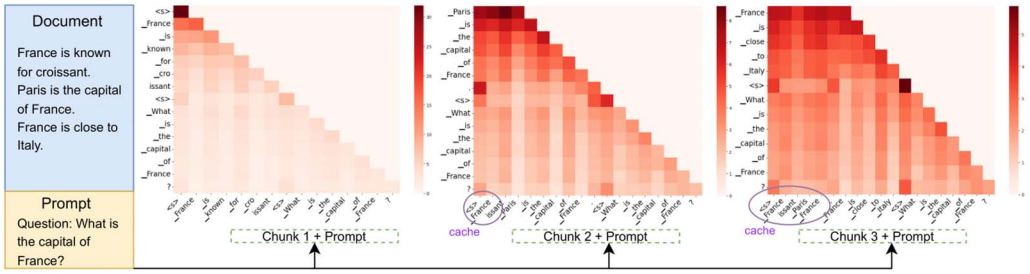

Figure 3: Attention distribution in the final layer of Llama 2, FINCH does the same analysis across all layers. The blue and yellow rectangles represent the document chunk and the user prompt, respectively. Initially, attention scores are evaluated between chunk 1 and the prompt, the most relevant tokens (circled) get stored in the cache in the next iteration. In successive iterations, the attention of the cached tokens together with the new chunk is measured w.r.t. the prompt. The final step involves only the cache and the prompt, leading the model to generate the response ‘‘Paris’’ based on the cached information.

Method	Complexity per Layer	Sequential Ops	Cache Growth/Op.
Vanilla	$O(n^2d)$	$O(1)$	$\Delta c = n$
FINCH	$O(mcd + m^2d)$	$O\left(\frac{n}{m}\right)$	$\Delta c = \frac{m}{\sigma}$

Table 1: Complexity comparison between the Vanilla transformer and FINCH in the Prefill stage.

for rotating to earlier or later positions in the sequence.

Compression Output: The final cache, composed of $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{V}}$, represents the compressed document, which encapsulates its essence in a condensed form and is used in the Generation stage.

5.2 Complexity Analysis

To illustrate the computational benefit of our approach, we report a comparative analysis of complexity metrics between the attention-based Vanilla transformer and FINCH. We consider Complexity per Layer according to n (total number of tokens), m (chunk size), d (model's embedding dimension), a (output sequence length), Sequential Operations as the number of times the model is invoked sequentially, Cache Growth per Operation as the increment in cache size c with each sequential operation, and Initial Cache Size at the beginning of the Generation stage (0 at the beginning of the Prefill stage). Table 1 shows complexities for the Prefill stage. For large n, the Vanilla method has a higher computational complexity due to quadratic relations, while FINCH introduces sequential operations that scale according to m, hence demonstrating enhanced efficiency and potential for scalability in processing large sequences ($m \ll n$). Table 2 shows complexities in the Generation stage, comparing the resource usage when synthesizing the final output. Also in this stage, the benefit for FINCH come from the reduced size of the initial cache according to the compression ratio σ.

5.3 Encoder-decoder

Our presentation of the methods is focused on a decoder-only architecture, as it is increasingly prevalent in NLP applications. While our methodology is experimented with decoder-only models, it is equally viable for encoder-decoder models that employ a KV cache mechanism. In such scenarios, during the Prefill stage, we can prefill the KV cache enabling the concise representation of context within the decoder. Subsequently, in the Generation stage we can feed the question or instructions to the encoder. The decoder then utilizes cross-attention mechanisms to access this information, along with the compressed context stored in the KV cache to generate the answer.

6 Experimental Setup

We evaluate FINCH using a variety of datasets and NLP tasks, with a focus on its application to the Llama 2 7B-chat (Touvron et al., 2023b) and the Mistral 7B-Instruct-v0.2 (Jiang et al., 2023a) models. Experiments are conducted with 4-bit NormalFloat Quantization and Double Quantization (Dettmers et al., 2023).

Method	Compl. per Layer	Seq. Ops	Initial Cache Size	Cache Growth/Op.
Vanilla	$O(cd)$	$O(a)$	$c = n$	$\Delta c = 1$
FINCH	$O(cd)$	$O(a)$	$c = \frac{n}{\sigma}$	$\Delta c = 1$

Table 2: Complexity comparison between the Vanilla transformer and FINCH in the Generation stage.

Unless otherwise noted, the experiments are conducted in a zero-shot setting.[4] Experiments are structured around three public datasets and four baseline methods.[5]

SQuADv2: For an assessment of FINCH's ability to preserve quality when compressing according to Equation 1, we use short texts that let us run the entire document as input. We use SQuAD v2 (Rajpurkar et al., 2018), a benchmark which includes both questions that can and cannot be answered with the given documents. We measure how our model maintains or improves its accuracy, despite having reduced context, against two baselines. First, we report for *Vanilla*, the standard model configuration which has access to the full context. Second, a *Truncate* strategy that reduces the input to the same size used by FINCH. Given a budget, we truncate the input after a number of tokens equal to half the reduced context both from the start and from the end, i.e., we take the beginning and the end of the document.

LongBench: To assess the robustness of our method with long documents and a variety of tasks, we also evaluate on the LongBench benchmark (Bai et al., 2024). This is a suite of tasks that involve extended contexts, including single-document question answering (QA), multi-document QA, document summarization, few-shot learning, code completion, and a synthetic task. The tasks span 16 datasets and presents a challenge due to the length of the input texts; for the size of the output, we use the original values in the dataset. For this dataset, our model is also compared against a third baseline, LongLLMLingua (Jiang et al., 2024), a state-of-the-art method for compression of long input texts. For LongLLMLingua, we use phi-2 (Li et al., 2023) as the compressor and Llama 2 7B-chat (or Mistral 7B-Instruct-v0.2), quantized at 4 bits with double quantization, as the generator. Unlike LongLLMLingua, our method does not use an external model for compression. For question answering tasks, a natural baseline is a Retrieval Augmented Generation (RAG) solution (Lewis et al., 2020). In our implementation of RAG, we segment the long text into chunks of 256 tokens each. To identify the most relevant chunks, we calculate the cosine similarity between the embeddings of these chunks and the embedding of the prompt. We use the *all-mpnet-base-v2* model from Sentence Transformers (Reimers and Gurevych, 2019) for generating these embeddings.

Lost in the Middle: A critical challenge for LLMs is the ''lost in the middle'' issue (Liu et al., 2024), where models exhibit degraded performance if relevant information is situated in the middle of long contexts. We evaluate the robustness of our compression technique also in their dataset.

7 Results and Discussion

We discuss five questions over our results.

1. Does FINCH's compression preserve the relevant information? Our evaluation on SQuAD v2 measures how FINCH retains pertinent information in a compressed format. We compare the Vanilla approach (Llama 2 provided with full documents), FINCH constrained to target tokens size k, and the truncation strategy. We choose five values of target tokens sizes, corresponding to different average compression ratios; we obtain the latter by dividing the average number of tokens in the SQuAD tests (document and prompt) by the average number of tokens that FINCH uses according to the given target tokens size. Specifically, 384 target tokens corresponds to an average σ of 1.1x, 256 tokens to 1.53x, 192 tokens to 2.35x, 160 to 3.03x and 144 tokens to 3.76x.

The results in Figure 4 show that FINCH not only consistently outperforms the truncation strategy

[4] FINCH's code and datasets are available at `https://github.com/giulio98/context-compression/`.

[5] Details on the inference hyperparameters and on the chunk size m per every dataset are provided in the Appendix of the extended version of this paper (Corallo and Papotti, 2024).

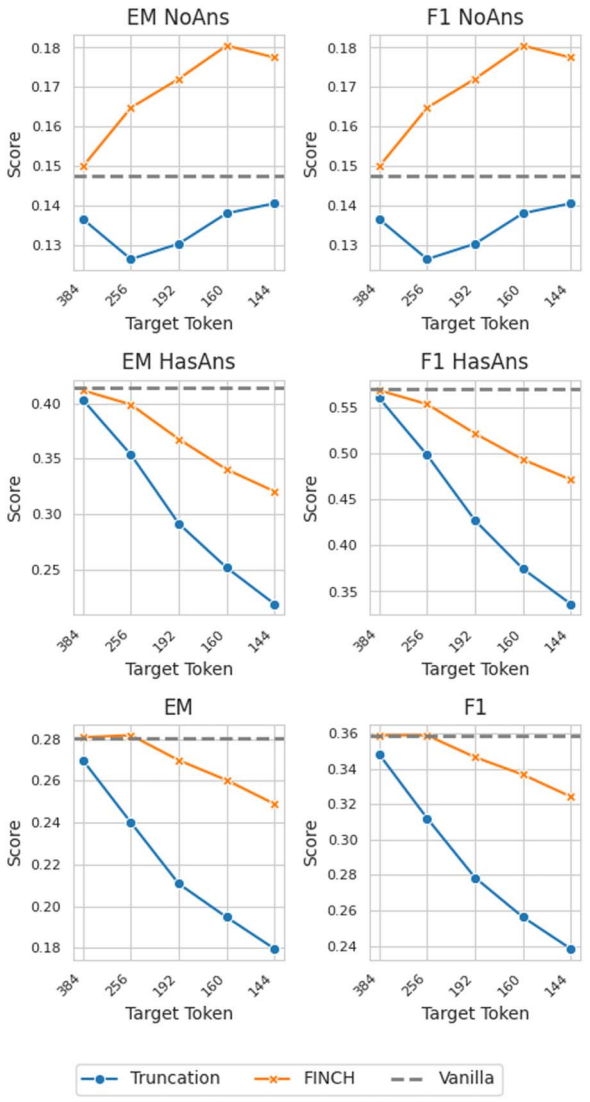

Figure 4: Performance results for SQuAD v2 for the Llama 2 Vanilla model, FINCH, and the truncation baseline. We report Exact Match (EM) and F1 scores for tests without answers (top), tests with answers (middle) and average across all tests.

across all token lengths but also, in certain cases, exceeds the quality performance of the Vanilla approach. This is evident in the F1 NoAns and Exact Match (EM) NoAns scores, where FINCH's ability to prevent responses based on irrelevant or non-existent evidence suggests that it eliminates extraneous content that could potentially mislead the model.

The overall EM and F1 scores indicate that FINCH maintains the integrity of the context as it is compressed. Even as the target tokens size k decreases, FINCH holds onto essential information, enabling the model to generate accurate responses with significantly less input data. In this dataset,

Model	Idx 0	Idx 4	Idx 9	Idx 14	Idx 19
Vanilla	24.7%	25.2%	28.2%	29.7%	40.0%
FINCH	38.0%	36.4%	38.2%	41.1%	46.2%

Table 3: ''Lost in the middle'' comparison of FINCH and Vanilla (Llama 2). Accuracy of returning the correct answer when the position of the document containing it varies across the model's input ($n = 4096$, $m = 256$). FINCH's $\sigma = 4$.

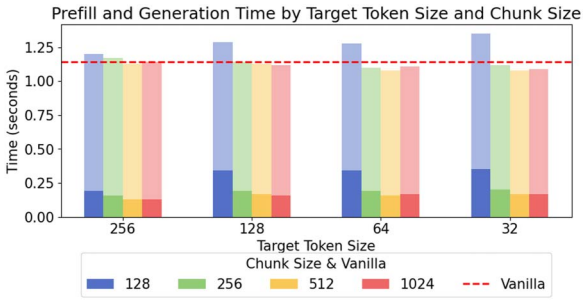

Figure 5: Impact of chunk and target tokens size k on decoding time for SQuAD v2; Finch's prefill (dark color) and generation (light color) times vs Llama 2 Vanilla (dotted red line: prefill+generation).

the loss of quality compared to the full context becomes more significant starting with an average compression of 3.7x.

To further illustrate the impact of our compression, we run the ''lost in the middle'' experiment, where the position of the information to answer the user question changes within the input document. It has been shown that this position has a significant impact on the model's accuracy (Liu et al., 2024). We compare again our solution against the original Vanilla model on the dataset from the paper reporting this problem. Results in Table 3 show that FINCH significantly outperforms the baseline across the different positions, with up to 13.3 absolute points gain when the correct answer is in the first document (Idx 0) and the compression ratio is 4x. The results also show that our method mitigates the original ''lost in the middle'' issue with 9.8 absolute points difference between the best and worst accuracy for FINCH, rather than 15.3 points for Vanilla.

2. How fast is FINCH compared to Vanilla self attention? Analysis of FINCH's efficiency, detailed in Figure 5, highlights a reduction on the overall time w.r.t. the Vanilla when the chunk

Task (metric)	Dataset	Vanilla	512 target tokens				1000 target tokens				2000 target tokens			
			Truncate	FINCH	LINGUA	avg(σ)	Truncate	FINCH	LINGUA	avg(σ)	Truncate	FINCH	LINGUA	avg(σ)
Single-Doc QA (F1 score ↑)	NarrativeQA	21.64	9.84	17.85	9.13	77.50x	11.28	20.38	9.13	37.16x	14.72	17.60	9.24	18.1x
	Qasper	24.93	9.23	19.59	9.71	13.68x	12.52	22.18	12.36	6.51x	16.50	23.19	15.62	3.40x
	MultiFieldQA	45.13	29.56	37.47	23.31	16.7x	36.8	42.11	24.60	8.52x	41.44	44.13	29.70	4.42x
	Overall	30.57	*16.21*	**24.97**	14.05		*20.20*	**28.22**	15.36		*24.22*	**28.30**	18.18	
Multi-Doc QA (F1 score ↑)	HotpotQA	17.15	19.20	29.89	18.28	34.38x	22.62	33.41	18.91	16.81x	26.43	33.21	25.01	8.42x
	MultihopQA	21.65	13.62	16.17	12.51	19.63x	14.79	18.42	13.74	9.85x	16.26	25.28	14.15	5.20x
	MuSiQue	19.25	7.58	12.43	6.09	39.96x	9.23	15.7	6.47	19.40x	11.94	17.86	8.23	9.64x
	Overall	19.35	*13.47*	**19.49**	12.29		*15.55*	**22.51**	13.09		*18.21*	**25.45**	15.80	
Summarization (Rouge-L ↑)	GovReport	24.24	18.70	19.05	18.16	25.1x	20.07	20.12	18.46	12.64x	21.36	21.05	19.03	6.50x
	QMSum	20.52	17.95	19.86	18.20	33.84x	18.86	20.04	18.03	16.72x	18.80	20.08	18.43	8.52x
	MultiNews	18.58	16.85	16.95	16.39	7.32x	17.94	17.79	16.89	3.89x	18.47	18.31	18.40	2.16x
	Overall	21.11	*17.83*	**18.62**	17.58		*18.96*	**19.31**	17.79		*19.54*	**19.81**	18.62	
Few-shot Learn (Accuracy ↑)	TREC	29.79	**40.39**	*36.75*	17.17	16.47x	*43.14*	**43.68**	10.08	8.37x	*44.43*	**47.41**	16.62	4.47x
Synthetic Task (Accuracy ↑)	PassageCount	0.96	0.25	*1.35*	**3.00**	41.61x	0.96	**2.41**	*2.00*	19.08x	*2.25*	**2.81**	2.21	9.33x
Code Complete (Edit Sim ↑)	LCC	26.01	18.97	31.93	15.08	9.78x	22.74	33.34	15.55	5.20x	24.31	34.59	18.56	2.91x
	RepoBench-p	25.65	18.51	24.19	15.64	28.65x	21.21	25.26	16.46	14.62x	23.34	25.63	18.60	7.55x
	Overall	25.83	*18.74*	**28.06**	15.36		*21.98*	**29.30**	16.01		*23.83*	**30.11**	18.58	

Table 4: Mistral results' comparison for the full context (Vanilla), truncation (Truncate), FINCH and LongLLMLingua (LINGUA) compression for different target tokens sizes (512/1000/2000) across datasets for six tasks. Best result per task and target tokens size in **bold**, second best in *italic*.

size is greater than 128 on Llama 2. This observation aligns with the complexity study in Section 5.2. Although FINCH introduces additional sequential operations in the Prefill stage, these are offset by the reduced complexity per layer, which is contingent on the chunk size m rather than the full context size n. This approach allows FINCH to handle each chunk with a complexity of $O(mcd + m^2d)$ as opposed to the Vanilla complexity per layer $O(n^2d)$. With larger chunk sizes, FINCH demonstrates improved speed over Vanilla self-attention. In the generation phase, the distinction in performance becomes more pronounced, as in Table 2. FINCH benefits from a smaller initial cache size, which is a function of the compression ratio σ. Such a configuration is advantageous in real-world applications where the response time is key and the volume of text to be processed is substantial.

3. How does FINCH perform on documents larger than the model context? To study how our method handles long input documents, we focus on the LongBench benchmark. As for the SQuADv2 experiment, we set the target tokens sizes and we feed the input document in chunks, while reserving space for the prompt and the output generation. We compare FINCH also against the state-of-the-art compression model LongLLMLingua.[6] As shown in Table 4 and Table 5, FINCH outperforms LongLLMLingua across five of the six tasks on Mistral and four out of six on Llama 2. The benefit of our solution is clear with different datasets and compression ratios, with a boost up to 8.8 absolute points of accuracy for question answering w.r.t. the best baseline (Truncate) on Mistral. Experiments on Llama 2 reports similar patterns, with a an improvement up to 6.3 points over the best QA baseline.

FINCH outperforms also the Vanilla baseline using the full document as input in the model context in 12 of the 18 experiments (overall results across 6 tasks and 3 target tokens sizes) on Mistral and in 15 over 18 on Llama 2. This is remarkable when considering that the compression ratio varies between 2.23x and 93.17x.

The baselines beat our method in 4 out of 6 experiments in the Synthetic task, where all methods report very low results. We explain this by the limits of the LLM with 7B parameters, since the tasks demands deep contextual understanding. FINCH shows better performance according to increasing target tokens sizes (512, 1000, 2000). In the question answering tasks, FINCH with a

[6]Results for LongLLMLingua are lower than those reported in their paper, where they use larger models such as ChatGPT (Jiang et al., 2024).

Task (metric)	Dataset	Vanilla	512 target tokens				1000 target tokens				2000 target tokens			
			Truncate	FINCH	LINGUA	avg(σ)	Truncate	FINCH	LINGUA	avg(σ)	Truncate	FINCH	LINGUA	avg(σ)
Single-Doc QA (F1 score ↑)	Narrative	16.69	11.14	19.10	10.56	93.17x	14.15	18.15	10.51	40.92x	15.45	19.45	11.68	19.37x
	Qasper	12.53	11.81	19.39	12.10	15.62x	12.27	20.25	11.82	7.00x	12.78	22.95	12.70	3.46x
	MultiField	34.50	30.26	33.47	21.87	17.86x	32.67	33.88	23.18	8.85x	38.43	34.67	27.35	4.50x
	Overall	21.24	*17.74*	**23.99**	14.84		*19.70*	**24.09**	15.17		*22.22*	**25.69**	17.24	
Multi-Doc QA (F1 score ↑)	Hotpot	30.46	25.31	36.75	26.13	38.64x	29.47	36.48	27.29	17.90x	30.07	34.29	28.32	8.71x
	Multihop	26.47	22.04	28.81	25.34	21.07x	22.90	27.96	24.64	10.24x	26.78	30.22	25.72	5.13x
	MuSiQue	10.54	9.41	14.12	9.43	45.97x	9.41	13.93	9.61	20.66x	8.25	12.58	10.21	10.03x
	Overall	22.49	18.92	**26.56**	*20.30*		*20.59*	**26.12**	20.51		*21.70*	**25.10**	21.42	
Summarization (Rouge-L ↑)	GovReport	18.02	17.79	18.20	17.27	28.30x	18.61	18.41	17.32	13.73x	19.19	18.79	17.86	6.84x
	QMSum	19.29	18.41	19.80	19.01	37.02x	18.47	19.63	18.86	17.38x	19.56	19.99	19.37	8.74x
	MultiNews	16.70	16.89	16.57	15.97	7.82x	17.29	17.22	16.61	4.11x	17.62	17.52	17.57	2.23x
	Overall	18.00	*17.70*	**18.19**	17.42		*18.12*	**18.42**	17.60		**18.80**	*18.77*	18.26	
Few-shot Learn (Accuracy ↑)	TREC	15.00	**24.25**	*23.75*	6.50	17.75x	*25.00*	**26.00**	6.50	8.78x	**32.50**	*29.00*	8.00	4.57x
Synthetic Task (Accuracy ↑)	P. Count	4.25	**5.17**	2.45	*4.50*	43.58x	**3.17**	2.32	*3.00*	19.65x	**2.60**	1.67	*2.00*	9.52x
Code Complete (Edit Sim ↑)	LCC	21.16	25.52	26.02	25.02	10.21x	25.06	25.79	22.14	5.32x	24.64	24.64	20.45	2.98x
	R. Bench	23.00	24.23	25.88	26.73	29.84x	23.33	24.67	24.11	14.97x	23.34	23.46	21.14	7.65x
	Overall	23.28	24.88	**25.95**	*25.88*		*24.20*	**25.23**	23.13		*24.00*	**24.05**	20.80	

Table 5: Llama 2 results' comparison for the full context (Vanilla), truncation (Truncate), FINCH and LongLLMLingua (LINGUA) compression for different target tokens sizes (512/1000/2000) across datasets for six tasks. Best result per task and target tokens size in **bold**, second best in *italic*.

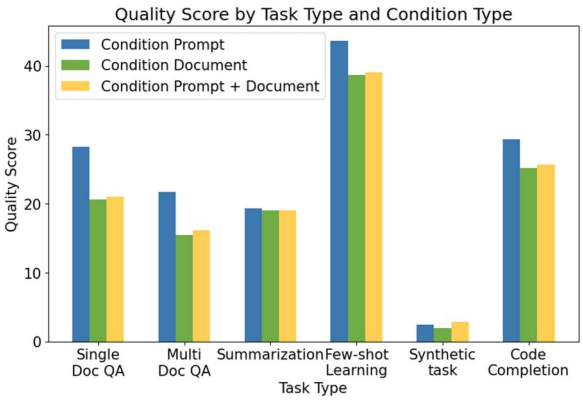

Figure 6: Impact of three types of condition in FINCH in all LongBench tasks on Mistral.

compression at 512 target tokens beats Truncate and LongLLMLingua with 1000 and 2000 target tokens, both with Llama 2 and Mistral.

We use the LongBench datasets also to validate our idea that conditioning the compression guided by the prompt is more effective than analyzing the self attention scores on the entire input (prompt and document) or on the document only. Results in Figure 6 show that over all the six tasks, the prompt guided solution leads to the best quality.

Finally, Figure 7 shows how FINCH outperforms the RAG baseline both on Mistral and Llama 2 at different compression rates in 10 over 12 question answering experiments. Compressing with FINCH, using the LLM KV cache, offers superior reliability w.r.t. a RAG solution, which suffers from increased latency and fragility due to its dependency on external retrieval mechanisms.

4. What is the effect of the chunk size? Figure 8 shows the impact of the chunk size m, i.e., the number of tokens into which the input context is divided for sequential processing by the model. Results show nuanced effects on quality performance. Larger chunk sizes (1024) yield better performance in single-document question answering, while smaller sizes (256) are more effective in multi-document settings. This can be attributed to the compression algorithm of retrieving a fixed number of top r tokens per iteration. In noisy multi-document contexts, a smaller chunk size enables better discrimination between relevant and irrelevant content, enhancing overall model performance. Chunk size has also an impact on the execution times. As expected, larger chunks lead to faster end-to-end execution because of the smaller number of iterations. These positive results are especially important for use cases that require longer outputs generated by the LLMs. As the user requires a bigger output, the space available for input processing gets smaller, thus reducing the size of the chunks in the Prefill stage.

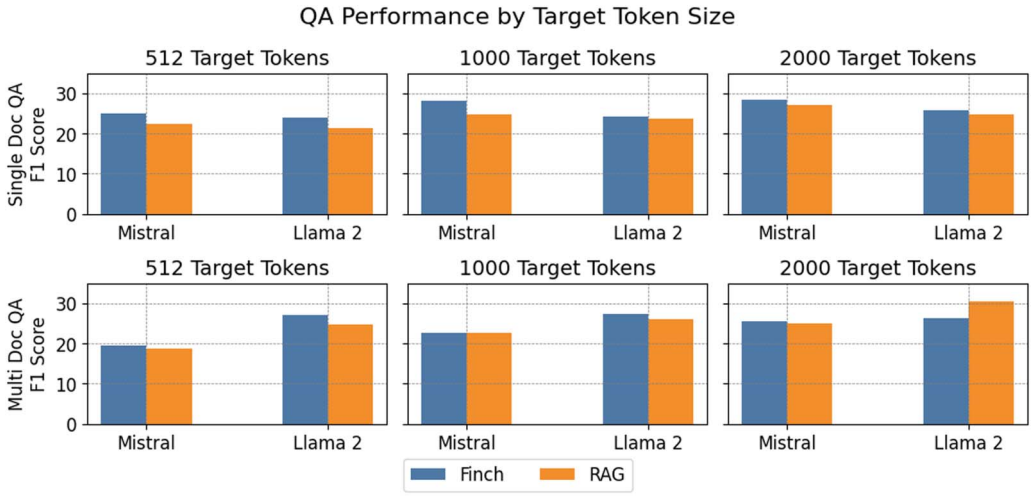

Figure 7: Comparison of FINCH and RAG in Mistral and Llama 2 for the QA tasks of LongBench.

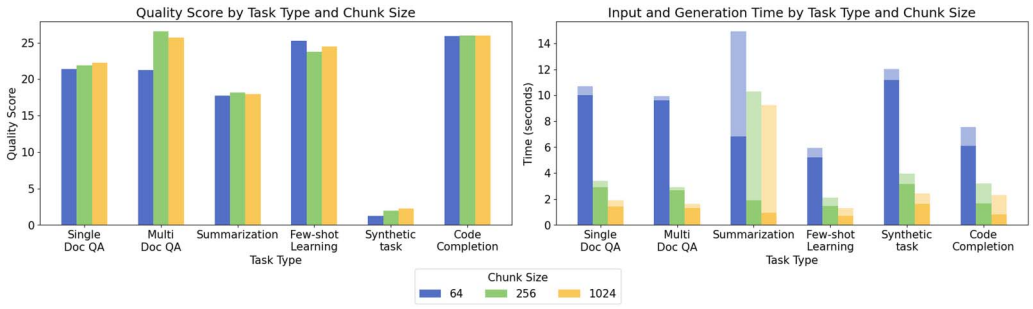

Figure 8: Ablation study for the impact of three chunk sizes in FINCH in all LongBench tasks on Llama 2 with a target tokens size of 512. Left: quality score. Right: inference Prefill (dark color) and Generation (light color) execution times.

Method	Model (GB)	KV Cache (GB)
Vanilla	4.33	4.52
FINCH ($\sigma = 2$)	4.33	2.38
FINCH ($\sigma = 4$)	4.33	1.30
FINCH ($\sigma = 8$)	4.33	0.60

Table 6: Memory consumption of Vanilla and FINCH at the beginning of the Generation stage.

5. What is the benefit in terms of GPU memory? Table 6 reports the memory consumed by FINCH (different compression rates) and the Vanilla model for the NarrativeQA (LongBench) dataset (truncated at $n = 4096$). Results show that our approach delivers a significant reduction in the initial KV cache size at the beginning of the Generation stage. Unlike the Vanilla model, FINCH achieves substantial memory savings by reducing the required cache size in proportion to the compression ratio, confirming the results in Table 2. This benefit enhances model scalability and makes FINCH a practical choice for deployment in resource-constrained environments.

8 Conclusion and Future Work

We have shown how attention can be used to identify and prioritize important information within the input data, effectively reducing the need for truncation. FINCH tackles the limitations of LLMs in processing large inputs, offering a balance between computational efficiency and maintaining high language model quality. Our solution leverages the pre-trained model weights of the self-attention mechanism to provide an economically feasible method for operating LLMs.

As future work, we envision a dynamic threshold mechanism to avoid that a fixed amount of KV states are selected in every chunk of the Prefill stage, exploiting the fact that some chunks are not relevant and can be compressed more. Another interesting research question is about the use of

the proposed method to compress the generated output tokens. This extension would be especially valuable in settings where the LLM is requested to generate long outputs, such as chain-of-thought reasoning. Our approach could be used to identify the important tokens to preserve in the generation step - this is aligned with results showing that preserving a fraction of the original context is sufficient to obtain high quality generated outputs (Xiao et al., 2024; Han et al., 2024).

Finally, we are interested in studying how cache compression techniques can be extended to structured data, e.g., for replacing the current data retrieval and filtering solution in table question answering (Badaro et al., 2023).

Acknowledgments

We thank the action editor and the reviewers for their comments which helped us improve the content of this work. We also thank Riccardo Taiello for the insightful discussion on complexity analysis.

References

Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics*, 11:227–249. https://doi.org/10.1162/tacl_a_00544

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. 2024. 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. Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.acl-long.172

Jean-Marie Chauvet. 2024. Memory GAPS: Would LLM pass the Tulving Test? *arXiv:2402.16505*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics. https://doi.org/10.18653/v1/W19-4828

Giulio Corallo and Paolo Papotti. 2024. Finch: Prompt-guided key-value cache compression. *arXiv:2408.00167*.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. 2020. Funnel-Transformer: Filtering out sequential redundancy for efficient language processing. In *Advances in Neural Information Processing Systems*, volume 33, pages 4271–4282. Curran Associates, Inc.

Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. 2024. Language modeling is compression. In *The Twelfth International Conference on Learning Representations*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv:2301.00234*.

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate quantization for generative pre-trained transformers.

In *The Eleventh International Conference on Learning Representations*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024a. Model tells you what to discard: Adaptive KV cache compression for llMs. In *The Twelfth International Conference on Learning Representations*.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2024b. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3690–3699. PMLR.

Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, and Minyi Guo. 2022. Transkimmer: Transformer learns to layer-wise skim. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7275–7286, Dublin, Ireland. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.502

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. LM-Infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008, Mexico City, Mexico. Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.naacl-long.222

Xin Huang, Ashish Khetan, Rene Bidart, and Zohar Karnin. 2022. Pyramid-BERT: Reducing complexity via successive core-set based token selection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8798–8817, Dublin, Ireland. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.602

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023a. Mistral 7b. *arXiv:2310.06825*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023b. 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, Singapore. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.825

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*. https://doi.org/10.18653/v1/2024.acl-long.91

Lukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. 2017. Learning to remember rare events. In *International Conference on Learning Representations*.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, pages 784–794, New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/3534678.3539260

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In

Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks are all you need II: phi-1.5 technical report. *arXiv:2309.05463*.

Yucheng Li. 2023. Unlocking context constraints of LLMs: Enhancing context efficiency of LLMs with self-information-based content filtering. *arXiv:2304.12102*.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023a. Ring attention with blockwise transformers for near-infinite context. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173. https://doi.org/10.1162/tacl_a_00638

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023b. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Ali Modarressi, Hosein Mohebbi, and Mohammad Taher Pilehvar. 2022. AdapLeR: Speeding up inference by adaptive length reduction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–15, Dublin, Ireland. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.1

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. *arXiv:2401.06104*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-2124

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1410

Siyu Ren and Kenny Q. Zhu. 2024. On the efficacy of eviction policy for key-value constrained generative language model inference. *arXiv:2402.06262*.

Yuanlong Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope, and Ray Kurzweil. 2017. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2210–2219, Copenhagen, Denmark. Association for Computational Linguistics. https://doi.org/10.18653/v1/D17-1235

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063. https://doi.org/10.1016/j.neucom.2023.127063

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasath R. Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv:1610.02424*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. 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. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.findings-emnlp.412

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc.