LoCoCo: Dropping In Convolutions for Long Context Compression

Ruisi Cai [1] **Yuandong Tian** [2] **Zhangyang Wang** [1] **Beidi Chen** [2 3]

Abstract

This paper tackles the memory hurdle of of processing long context sequences in Large Language Models (LLMs), by presenting a novel approach, Dropping In Convolutions for **Lo**ng **Co**ntext **Co**mpression (**LoCoCo**). LoCoCo employs only a fixed-size Key-Value (KV) cache, and can enhance efficiency in both inference and fine-tuning stages. Diverging from prior methods that selectively drop KV pairs based on heuristics, LoCoCo leverages a data-driven adaptive fusion technique, blending previous KV pairs with incoming tokens to minimize the loss of contextual information and ensure accurate attention modeling. This token integration is achieved through injecting one-dimensional **convolutional kernels** that dynamically calculate mixing weights for each KV cache slot. Designed for broad compatibility with existing LLM frameworks, LoCoCo allows for straightforward "drop-in" integration without needing architectural modifications, while incurring minimal tuning overhead. Experiments demonstrate that LoCoCo maintains consistently outstanding performance across various context lengths and can achieve a high context compression rate during both inference and fine-tuning phases. During inference, we successfully compressed up to 3482 tokens into a 128-size KV cache, while retaining comparable performance to the full sequence - an accuracy improvement of up to 0.2791 compared to baselines at the same cache size. During post-training tuning, we also effectively extended the context length from 4K to 32K using a KV cache of fixed size 512, achieving performance similar to fine-tuning with entire sequences. Codes are available at: https://github.com/VITA-Group/LoCoCo.

[1]University of Texas at Austin [2]Meta AI (FAIR) [3]Carnegie Mellon University. Correspondence to: Ruisi Cai <ruisi.cai@utexas.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Large Language Models (LLMs) (Radford et al., 2018; 2019; Brown et al., 2020) excel across a variety of linguistic tasks, including text generation (Goyal & Durrett, 2020; Yuan et al., 2022), program synthesis (Chen et al., 2021; Li et al., 2022), question answering (Kamalloo et al., 2023), and mathematical problem-solving (Lewkowycz et al., 2022). These tasks typically involve processing extensive sequences, often requiring the analysis of thousands of tokens to derive outcomes based on comprehensive contextual information. For example, the task of summarizing extensive government reports, as seen in the GovReport section of SCROLLS (Shaham et al., 2022), demands that LLMs efficiently sift through and distill key information from vast textual data, highlighting the need for models capable of handling long token sequences effectively.

Yet, transformers (Vaswani et al., 2017) struggle to process extensive token sequences due to their quadratic memory demands, which exceed the capacity of contemporary hardware. Attention computations are performed in blocks (Dai et al., 2019), with key and value states cached for subsequent encoding or decoding steps to mitigate this. However, this approach results in a Key-Value (KV) cache size that increases linearly with context length, quickly depleting GPU memory (Zhang et al., 2023b). Recently, StreamingLLM (Xiao et al., 2023) attempted to reduce KV cache size by limiting each token's receptive field and incorporating "attention sinks". Concurrently, H_2O (Zhang et al., 2023b) prunes tokens based on lower accumulated attention scores to stabilize KV cache size. Despite these efforts, both methods fail to leverage full-sequence information and adequately extend the context window. StreamingLLM's exclusion of all tokens in the context middle could significantly impair the model's ability to utilize the full long context (even completely ignore), known as "lost in the middle" (Liu et al., 2023), while H_2O struggles to extrapolate to longer sequences than the training context length (Han et al., 2023).

Enhancing the context length in LLMs also necessitates increasing the block size during fine-tuning (Press et al., 2021; Chen et al., 2023a), introducing a significant memory challenge. While attention approximation methods like (Choromanski et al., 2020; Kitaev et al., 2020; Xiong et al., 2021) reduce training expenses, they do not alleviate memory de-

mands at inference, as the KV cache continues to explode with longer predictions. Another representative work LongLoRA (Chen et al., 2023b) leveraged locally grouped attention alongside LoRA (Hu et al., 2021) for quick adaptation to longer-context data. However, implementing LongLoRA necessitates several modifications to the architecture of pre-trained LLMs for fine-tuning, hence not yet a hassle-free "drop-in" option. Besides, the LongLoRA-tuned model may compromise its performance if we still use smaller context sizes (see our experiments).

In our study, we address the challenge of efficiently managing long contexts in **both inference and fine-tuning** phases. We introduce a novel method, **Lo**ng **Co**ntext Compression by Dropping-In **Co**nvolutions, abbreviated as **LoCoCo**. This technique employs a **static-size KV cache** for segment-level attention processes, ensuring peak memory usage remains unchanged. LoCoCo departs from traditional methods of dropping KV pairs based on pre-defined or ad-hoc rules (Zhang et al., 2023b; Xiao et al., 2023), instead adopting a data-driven **adaptive fusion** approach that merges prior KV pairs with new tokens. This fusion minimizes the loss of the whole context and achieves accurate attention modeling. Specifically, LoCoCo utilizes one-dimensional **convolutional kernels** to calculate mixing weights for each KV cache slot, integrating incoming tokens efficiently (Kim, 2014; Poli et al., 2023; Massaroli et al., 2023). This strategy is informed by the **insight** that autoregressive generation benefits from the continuity provided by shifting windows, and introducing the shift-invariant operation of convolutions can reinforce the sequence's stationary inductive bias. It counters potential discontinuities that might arise from excluding tokens during the generative process (e.g., one token in the middle might contribute to the current generation, but "suddenly" be dropped when generating the next token).

It is important to note that LoCoCo is designed to be **universally compatible** with existing LLM architectures, allowing for seamless integration without necessitating any modifications to the original model designs. It requires merely "**dropping in**" a few extra convolution layers, incurs a small tuning overhead, yet can achieve consistently effective performance across various context lengths, with **high context compression rates** for both inference and fine-tuning.

Our contributions could be summarized as follows:

- We introduced the novel (LoCoCo) method to manage long contexts efficiently at both inference and fine-tuning, employing a static-size KV cache and data-driven adaptive fusion of context information.
- LoCoCo utilized one-dimensional convolutional kernels for dynamic weight calculation in the KV cache, enhancing accurate attention modeling while addressing the challenges of sequence continuity and the stationary inductive bias for autoregressive generation.
- LoCoCo highlights universal compatibility with existing LLM architectures, enabling easy "drop-in" integration without extra design modifications, and achieving high context compression rates across different context lengths with minimal tuning overhead.
- During inference, we successfully compress up to 3482 tokens into a 128-size KV cache, while retaining comparable performance to the full sequence - an accuracy improvement of up to 0.2791 compared to baselines at the same cache size. During post-training tuning, we extended the context length from 4K to 32K using a KV cache of fixed size 512, achieving performance similar to fine-tuning with entire sequences.

2. Related Work

2.1. Long-Context Inference

Generating long contexts necessitates a KV cache for preceding tokens and incurs a significant memory overhead. For memory-efficient inference, Zhang et al. (2023b) proposes mitigating KV cache demands during long-context generation through auto-regressive token eviction. Furthermore, Ribar et al. (2023) optimizes memory usage by selectively fetching from the cached history. Approaching differently, Jiang et al. (2023) focuses on prompt compression techniques to create concise yet expressive prompts. Meanwhile, Xiao et al. (2023) achieves infinite-length context generation by only storing tokens within a local window plus "attention sink" tokens, and rolling position embeddings. However, they fall short of utilizing full-sequence information and extending the context window.

2.2. Long-Context Fine-tuning

The limited sequence length of pre-trained LLMs and their inability to handle long-context data effectively are major concerns for practitioners. To address this, strategies such as extending the context length through fine-tuning have been explored Xiong et al. (2023). The work of Dai et al. (2019) introduces a segment-level recurrence mechanism using fixed-length training segments. Other approaches include positional interpolation (Chen et al., 2023a), NTK-aware embedding (ntk, 2023), Yarn (Peng et al., 2023), positional skipping (Zhu et al., 2023), self-extension (Jin et al., 2024), stabilized attention entropy (Zhang et al., 2024), and so on. Additionally, landmark attention (Mohtashami & Jaggi, 2023a) introduces a gating mechanism based on landmark tokens, each representing a block of tokens. This method selectively retains "landmarks" in memory, utilizing other memory resources (e.g., CPU memory or disk) for storing the remaining tokens. Tworkowski et al. (2023) employs contrastive learning, LongLoRA (Chen et al., 2023b) introduces shifted sparse attention and parameter-efficient

fine-tuning. Zhang et al. (2023a) investigates the necessity of attending to long-context tokens in a layer-wise manner.

2.3. Attention Approximation

Efforts to mitigate the quadratic complexity of transformers primarily focus on attention approximation. A comprehensive review of the rich literature can be found in (Tay et al., 2022). Specifically, Child et al. (2019); Kitaev et al. (2020); Roy et al. (2021) leverages sparsity, and Choromanski et al. (2020); Katharopoulos et al. (2020); Wang et al. (2020) utilizes low-rank approximation. Beltagy et al. (2020); Zaheer et al. (2020) approximated the full attention with both local and global attention. Nevertheless, none of these approaches eliminate the memory bottleneck for the KV cache.

2.4. Language Model Design with Built-In Convolutions

(Dauphin et al., 2017) introduced the first convolutional language model that rivaled strong recurrent models on large-scale language tasks. More recently, (Poli et al., 2023; Arora et al., 2023) proposed using long convolutions to completely replace attention mechanisms in transformers. Additionally, state-space models (SSMs) can be computed as either convolutions or recurrences, achieving sub-quadratic training and constant inference complexity (Gu et al., 2021a). Architectures utilizing implicit convolutional filters (Poli et al., 2023) can be converted to SSMs via a simple distillation step (Poli et al., 2023; Massaroli et al., 2023). These designs inspired our research; however, our work has a different focus of providing "drop-in" components to enhance the long-context capability of pre-trained LLMs.

3. Methodology

3.1. Segment-Level Attention with Long Sequences

The attention mechanism (Vaswani et al., 2017) plays as a crucial component in transformers. Suppose the sequence length is L and the hidden dimension is d. In causal language modeling, an attention block receives query, key, and value matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{d\times L}$, and computes outputs as:

$$\mathrm{Attn}(\boldsymbol{K}, \boldsymbol{Q}, \boldsymbol{V}) = \boldsymbol{V}\mathrm{softmax}\left(\boldsymbol{K}^\top \boldsymbol{Q} \odot \boldsymbol{M}/\sqrt{d}\right), \quad (1)$$

where $\boldsymbol{M}$ is a lower-triangular causal mask. Intuitively, causal attention only allows tokens to aggregate information from past tokens. Given a sequence with L tokens $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_1 & \cdots & \boldsymbol{x}_L\end{bmatrix} \in \mathbb{R}^{d\times L}$, the query, key, value matrices are computed as the linear projections of $\boldsymbol{X}$:

$$\boldsymbol{K} = \boldsymbol{W}_K\boldsymbol{X}, \boldsymbol{Q} = \boldsymbol{W}_Q\boldsymbol{X}, \boldsymbol{V} = \boldsymbol{W}_V\boldsymbol{X}. \quad (2)$$

As illustrated in the Figure 1 (a), acquiring full attention matrix $\mathrm{softmax}(\boldsymbol{K}^\top\boldsymbol{Q})$ requires $\mathcal{O}(L^2)$ peak memory cost. When L is large, i.e. handling long sequential data, full attention computation tend to run out of GPU memory rapidly.

Algorithm 1 Segment-level Attention (Training Time)

Input: A full sequence of length L: $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_L$, block size B, the number of segments N.
Initialize an empty cached KV pairs as $\widetilde{\boldsymbol{K}}$, $\widetilde{\boldsymbol{V}}$.
for $n = 1, \cdots, N$ **do**
  Step 1 - Let $\boldsymbol{X}_n = \begin{bmatrix}\boldsymbol{x}_{nB} & \cdots & \boldsymbol{x}_{(n+1)B-1}\end{bmatrix} \in \mathbb{R}^{d\times B}$ collect a sequence of the n-th segment.
  Step 2 - Calculate key, query, values: $\boldsymbol{Q}_n = \boldsymbol{W}_Q\boldsymbol{X}_n$, $\boldsymbol{K}_n = \boldsymbol{W}_K\boldsymbol{X}_n$, and $\boldsymbol{V}_n = \boldsymbol{W}_V\boldsymbol{X}_n$.
  Step 3 - Perform attention as:
    $\boldsymbol{O}_n \leftarrow \mathrm{Attn}([\widetilde{\boldsymbol{K}}, \boldsymbol{K}_n], \boldsymbol{Q}_n, [\widetilde{\boldsymbol{V}}, \boldsymbol{V}_n])$
  Step 4 - Update cached KV pairs:
    $\widetilde{\boldsymbol{K}} \leftarrow [\widetilde{\boldsymbol{K}}, \boldsymbol{K}_n]$, $\widetilde{\boldsymbol{V}} \leftarrow [\widetilde{\boldsymbol{V}}, \boldsymbol{V}_n]$.
end for
Return $\begin{bmatrix}\boldsymbol{O}_1 & \cdots & \boldsymbol{O}_N\end{bmatrix}$

Context chunking is a common practice for reducing peak memory usage during training. Owing to the causality, Transformer-XL (Dai et al., 2019) introduces a recurrent computation mechanism by caching and reusing the hidden states to extend the context length for both training and inference. Specifically, the whole sequence is divided into a couple of segments, each then processed sequentially. The intermediate key and value states will be stored in the memory. Previously cached KV pairs will be used for computing the token representation in the subsequent segments.

Algorithm 1 presents the detailed procedure for the training-time attention computation. Suppose the input sequence of length L can be divided into N segments, where each block has B tokens, i.e. $L = NB$. The symbol $[\cdot, \cdot]$ therein denotes the concatenation of two matrices' columns.

Note that auto-regressive generation is a special case of segment-level attention at $B = 1$. This is, tokens come in sequel and attention is only computed between the incoming query and past KV pairs. The cached KV pairs $\widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}}$ are known as *KV cache* for short in the inference mode [1].

As illustrated in Figure 1(b), by performing context chunking, the memory used by attention for the r-th block is $\mathcal{O}(B^2r)$, and the memory to store past KV pairs is $\mathcal{O}(Br)$. Hence, the peak memory usage for computing the full attention is reduced to $\mathcal{O}(LB)$, which occurs at the N-th round when the last token needs to attend all previous key and value blocks $\{\boldsymbol{K}_n, \boldsymbol{V}_n, n \in \{1, ..., N\}\}$. The deduction of peak memory usage comes at the cost of increased caching memory, which grows linearly with the sequence length.

[1] With a slight ambiguity, we also refer to the training-time saved KV pairs as the KV cache since their functionality is identical to their test-time counterparts.

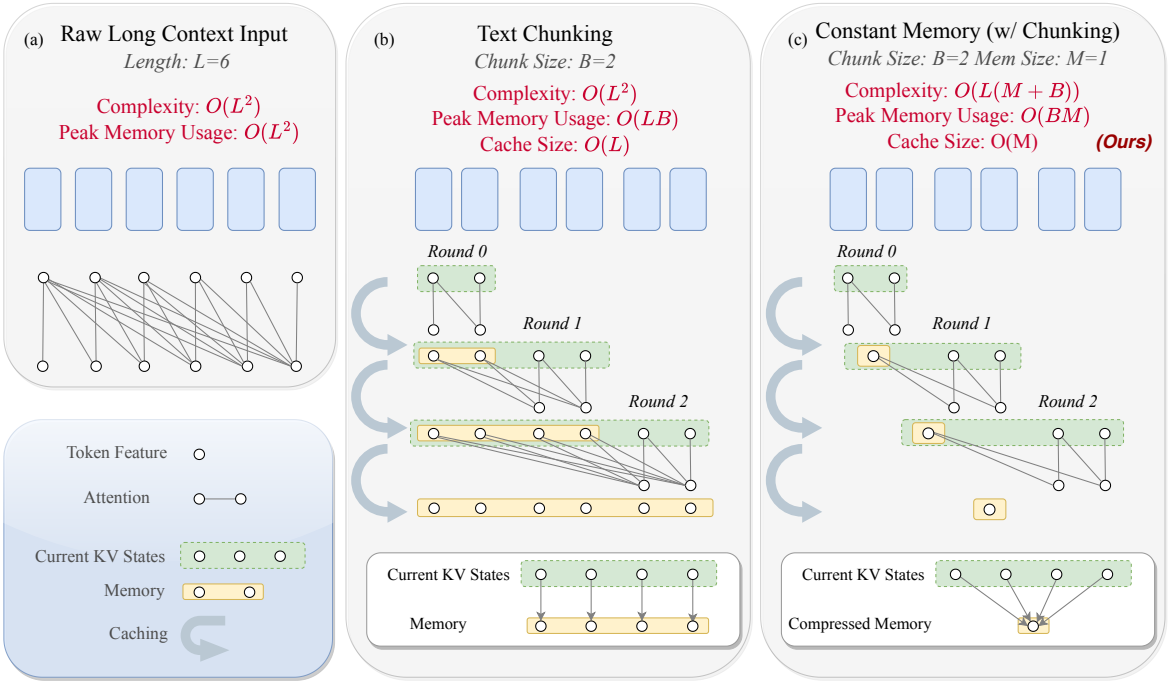

Figure 1. Overview of our pipeline. We process the long sequences block-wisely and maintain a fixed-size compressed memory.

3.2. Convolution as a Context Compression Operator

So far, the peak memory footprint has been reduced from quadratic to linear concerning sequence length. However, this linear growth of the KV cache can still lead to excessive memory usage as the sequence length increases (Zhang et al., 2023b). Early attempts using k-NN lookup (Wu et al., 2022) and gating mechanisms (Mohtashami & Jaggi, 2023b) enable sparse token selection to save memory but still require caching all previous tokens, resulting in a cache size of $\mathcal{O}(L)$. In this section, we introduce a framework that further optimizes this linear complexity to a constant size.

Compressing past token information using a fixed-size hidden space is well-documented in the literature. Notably, State Space Models (SSMs) utilize a fixed-dimension latent vector to represent all prior tokens, showing great promise for long-sequence modeling (Gu et al., 2021b;a; 2020; 2022; Gupta et al., 2022; Fu et al., 2022; Gu & Dao, 2023). This hidden vector interacts with incoming tokens on behalf of all previous tokens.

Inspired by this, we propose allocating at most M slots to store past KV pairs, allowing subsequent sequence blocks to attend to these compressed KV states. We replace the simple concatenation in Step 4 with a KV compression operator $\mathcal{C}$:

$$\widetilde{\boldsymbol{K}} \leftarrow \mathcal{C}([\widetilde{\boldsymbol{K}}, \boldsymbol{K}_n]), \quad \widetilde{\boldsymbol{V}} \leftarrow \mathcal{C}([\widetilde{\boldsymbol{V}}, \boldsymbol{V}_n]), \tag{3}$$

where $\mathcal{C}$ maps a longer sequence to a sequence of length M. We next elaborate on our instantiation of $\mathcal{C}$.

3.2.1. Convolutional Token Compressor

There are various ways to implement the sequence function $\mathcal{C}$ to meet the above definition. In this paper, we propose modeling the update rule of the KV cache as a weighted fusion between existing cache entries and newly input tokens. Formally, for all $\forall i \in [M]$:

$$\widetilde{\boldsymbol{k}}_i \leftarrow \sum_{j=1}^{B} w_{i,j}\boldsymbol{k}_j + \sum_{j=1}^{M} \widetilde{w}_{i,j}\widetilde{\boldsymbol{k}}_j \tag{4}$$

$$\widetilde{\boldsymbol{q}}_i \leftarrow \sum_{j=1}^{B} w_{i,j}\boldsymbol{q}_j + \sum_{j=1}^{M} \widetilde{w}_{i,j}\widetilde{\boldsymbol{q}}_j, \tag{5}$$

where $w_{i,j}$ denotes the contribution of the j-th token in the input block to the i-th entry in the cache, and similarly $\widetilde{w}_{i,j}$ the contribution of the j-th token in the existing cache to the i-th entry in the updated cache. Here weights for keys and queries are shared to preserve token correspondence.

We further identify three key properties desired for $\{w_{i,j}\}$ and $\{\widetilde{w}_{i,j}\}$: ***1) Efficiency***: computing these weights is an intermediate step of performing attention, and hence its overheads should be negligible - otherwise we beat our purpose. ***2) Learnability***: Ad-hoc $\{w_{i,j}\}$ and $\{\widetilde{w}_{i,j}\}$, such as averaging (i.e., uniform weights) or heuristic-based token dropping (i.e., many zero weights) (Zhang et al., 2023b), may not be flexible enough or introduce extra bias (e.g., locality (Chen et al., 2023b) or "lost in the middle" (Liu et al., 2023)). ***3) Stationarity***: the compression policy must

be globally informed and stable concerning token position, ensuring that compressed KV states update continuously as tokens are processed. This addresses the potential disruptions in KV states caused by dropping tokens during the generative process (Zhang et al., 2023b).

It has not escaped our notice that convolutional kernels fulfill all the aforementioned requirements. Therefore, we propose using convolutional layers to generate $w_{i,j}$ and $\widetilde{w}_{i,j}$. Specifically, a 1D convolution will process all the pairs existing in the KV cache and the newly incoming segment, assigning each token an M-dimensional output that indicates its importance for each slot in the updated cache. Formally, we denote $\boldsymbol{g} : \mathbb{R}^{2d\times(M+B)} \to \mathbb{R}^{M\times(M+B)}$ as a Convolutional Neural Network (CNN) with $2d$ input channels and M output channels. Then weights $\{w_{i,j}\}$ and $\{\widetilde{w}_{i,j}\}$ are computed as:

$$\boldsymbol{W} \leftarrow \boldsymbol{g} \circledast \begin{bmatrix} \boldsymbol{K}_n & \widetilde{\boldsymbol{K}} \\ \boldsymbol{V}_n & \widetilde{\boldsymbol{V}} \end{bmatrix} \in \mathbb{R}^{M\times(M+B)}, \tag{6}$$

$$w_{i,j} \leftarrow \frac{\boldsymbol{W}_{i,j}}{\sum_{k=1}^{M+B} \boldsymbol{W}_{i,k}}, \quad \widetilde{w}_{i,j} \leftarrow \frac{\boldsymbol{W}_{i,j+B}}{\sum_{k=1}^{M+B} \boldsymbol{W}_{i,k}}, \tag{7}$$

where $\circledast$ denotes multi-channel convolution operation along columns of two operands. Here we normalize the prediction from the CNN kernel $\boldsymbol{g}$ as the final blending weights. Convolution parameters are trained end-to-end with a small set of calibration data.

We name our approach as *Long Context Compression by Dropping-In Convolutions*, or **LoCoCo** for short. We summarize the outline of LoCoCo in Algorithm 2, where the major differences from Algorithm 1 are highlighted in Steps 4 and 5. When the number of KV entries to be stored $\#(\widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}}) + B$ surpasses the number of slots M, we apply convolution-based compression between cached entries and newly added KV pairs. Otherwise, we preserve all KV pairs in the memory. In our implementation, we adopt a shallow CNN for each attention layer. The convolutional head consists of a single convolution layer with kernel size 21. In addition, we prepend a ReLU as the activation function.

3.2.2. Complexity Analysis

With negligible computational overhead, LoCoCo achieves **constant memory** regardless of sequence length. In Step 3, the attention is computed between a group of M tokens and a group of B tokens. The memory cost for this step is maintained as $\mathcal{O}(MB)$. Step 4 synthesizes fusion weights via convolution, whose computation complexity can be as cheap as $\mathcal{O}(L \log L)$ by Fourier transformation. Afterward, Step 5 leads to a constant-size cache, which guarantees the computation in the next round does not require more memory. Therefore, the total peak memory cost is $\mathcal{O}(MB + M)$ with an extra $\mathcal{O}(L \log L)$ computation overhead.

Algorithm 2 LoCoCo Attention (Training Time)

Input: A full sequence of length L: $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_L$, block size B, the number of segments N, the number of total cached KV entries M.
Initialize an empty cached KV pairs as $\widetilde{\boldsymbol{K}}$, $\widetilde{\boldsymbol{V}}$.
for $n = 1, \cdots, N$ **do**
  Step 1 - Let $\boldsymbol{X}_n = \begin{bmatrix} \boldsymbol{x}_{nB} & \cdots & \boldsymbol{x}_{(n+1)B-1} \end{bmatrix} \in \mathbb{R}^{d\times B}$ collect a sequence of the i-th segment.
  Step 2 - Calculate key, query, values: $\boldsymbol{Q}_n = \boldsymbol{W}_Q \boldsymbol{X}_n$, $\boldsymbol{K}_n = \boldsymbol{W}_K \boldsymbol{X}_n$, and $\boldsymbol{V}_n = \boldsymbol{W}_V \boldsymbol{X}_n$.
  Step 3 - Perform attention as:
    $\boldsymbol{O}_n \leftarrow \mathrm{Attn}([\widetilde{\boldsymbol{K}}, \boldsymbol{K}_n], \boldsymbol{Q}_i, [\widetilde{\boldsymbol{V}}, \boldsymbol{V}_n])$
  if $\#(\widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}}) + B \le M$ **then**
    Fill KV cache: $\widetilde{\boldsymbol{K}} \leftarrow [\widetilde{\boldsymbol{K}}, \boldsymbol{K}_n]$, $\widetilde{\boldsymbol{V}} \leftarrow [\widetilde{\boldsymbol{V}}, \boldsymbol{V}_n]$.
  else
    Step 4 - Compute fusion weights as: $\{w_{i,j}\}$ and $\{\widetilde{w}_{i,j}\} \leftarrow$ Equations 6 and 7.
    Step 5 - Update cached KV pairs: $\forall i \in [M]$, $\widetilde{\boldsymbol{k}}_i \leftarrow$ Equation 4, $\widetilde{\boldsymbol{q}}_i \leftarrow$ Equations 5.
  end if
end for
Return $\begin{bmatrix} \boldsymbol{O}_1 & \cdots & \boldsymbol{O}_N \end{bmatrix}$

3.2.3. Connection with Token Dropping

Zhang et al. (2023b) proposes to use accumulated attention scores to determine the importance of tokens. The method then auto-regressively keeps tokens with the top scores and discards others. That can be viewed as a special instance of operator $\mathcal{C}$ in Equation 3. However, the heuristic-based method is less expressive compared to our learnable framework. Specifically, as detailed in Section 4.2 and Section 4.3, LoCoCo, empowered by the general learnable token compression paradigm, demonstrates superior performance compared to prior arts in token eviction. In addition, our method can be executed on top of other token eviction methods, as to be discussed in Section 5.2.

3.3. Dropping-In Integration of LoCoCo

In this section, we introduce how our technique can be easily integrated to pre-trained LLMs, for both long-context inference and long-context training purposes.

Long-Context Efficient Inference Standard LLMs cache all previous KV pairs, resulting in high memory usage that limits their applicability in memory-constrained inference. To address this, we "drop in" a compressor on top of the pre-trained weights. The compressor is optimized using Algorithm 2 with a minimal fraction of the training data (e.g., 104 million tokens, or 0.0052% of the 2 trillion tokens used for Llama-2 pre-training (Touvron et al., 2023)).

During the pre-filling stage, prompts are split into segments of size B before being fed into the LLM. These segments sequentially pass through the LLM, generating and compressing KVs via Equation 3, resulting in compressed KVs of length M that encapsulate the context information. In the generation stage, the segment length is set to 1. Detailed results are provided in Section Section 4.2.

As our "dropping-in" term implies, *the pre-trained weights remain unchanged*, allowing users to switch back to the uncompressed mode simply by removing the compressor heads, when sufficient resources are available for a linearly scaled KV cache.

Long-Context Extension Our method also supports long context extension through post-training tuning, allowing pre-trained LLMs to handle longer contexts without incurring the excessive memory costs. We achieve this by leveraging positional interpolation (Chen et al., 2023a), inserting compressor heads, and adding LoRA adapters to fine-tune the pre-trained model, following Chen et al. (2023b)'s practice. The fine-tuning procedure is detailed in Algorithm 2.

4. Experiment

We first describe our experimental settings in Sec 4.1. Then, we demonstrate our proposed convolutional head as a plug-in tool for pre-trained LLMs, that enables memory-efficient inference, in Sec 4.2. Additionally, in Sec 4.3, we apply the proposed method to long context fine-tuning, enabling training with long sequences under fixed-size memory.

4.1. Experimental Settings

Base Models We select Llama2-7B and Llama2-13B (Touvron et al., 2023) as our base models, each with a maximum context length of 4096 tokens. For inference with context lengths shorter than 4096 tokens (in Sec 4.2), we retain the original model weights and fine-tune only the convolutional heads. To extend the context window to 32768 tokens during long context fine-tuning (in Sec 4.3), we utilize positional interpolation for initialization (Chen et al., 2023a).

Convolutional Heads We insert convolutional heads layer-wise to capture the diverse token relationships across layers. Each convolutional head possesses one layer of 1-D convolutional kernels: its input feature dimension is the dimension of key and value matrices, while the output feature dimension is the target memory size. We set the kernel size to be 21 by default, as more choices will be validated in Sec 5.3. Within the same layer, all attention heads would share the same set of convolutional kernel parameters. Thus, for Llama2-7b, a 32-layer model with 256-dimension KV states, we only add 22 million parameters for compressing raw KV states to a memory of 128 tokens.

Compression Details For post-hoc compression without modifying pre-trained LLMs, we experiment compressing the sequence of length up to 4096, to fit in the memory size of 128, 256, 512, leading to the compression ratio of $32:1$.

For experiments on context length extending, we by default set 512 as the memory size. We will validate more choices ranging from 128 to 1024 in Sec 5.1.

Training Details We use RedPajama (Computer, 2023) as our training dataset. For post-hoc compression experiments, we only tune compression heads for 200 steps without modifying the pre-trained LLM. For context length extending, we fine-tune the convolutional heads and LoRA adapters (rank 8), and also allow modifying the embedding and normalization layers, all following Chen et al. (2023b).

For all experiments, we use the learning rates of 5×10^{-5} for LoRA adapters, embedding and normalization layers and 5×10^{-2} for convolutional heads, with linear learning rate schedule. We use the batch size of 128, and chunk size of 512. All experiments are run on A6000 (48GB memory) to intentionally test our efficacy with small-memory GPUs, and we use per-device batch size as 1.

4.2. Post-hoc Token Compression of Pre-trained Models

At inference, we validate LoCoCo on representative downstream tasks, under target memory sizes varying from 128 to 512. We select the reading comprehension dataset RACE (Lai et al., 2017) (2, 4, 6 shots), the closed-book question answering dataset TriviaQA (Joshi et al., 2017) (50 shots), and the common sense reasoning dataset: HellaSwag (Zellers et al., 2019) (10, 20, 40 shots), WinoGrande (Sakaguchi et al., 2021) (70 shots), and ARC easy and challenge (Clark et al., 2018) (40 shots). Note that we deliberately keep the sequence length of each task within the maximum sequence length of the pre-trained Llama-2 (Touvron et al., 2023).

Using the Llama-2-7b (Touvron et al., 2023) as the base model, we compare our approach with H_2O (Zhang et al., 2023b), a recent token dropping method. As in Figure 2, LoCoCo shows exceptional performance on various tasks, especially on tasks whose average sequence length is long.

LoCoCo be further applied onto any long-context model. We insert convolutional heads on the top of ChatGLM3-6B-32k (Du et al., 2021), a representative long-context pre-trained model. We evaluate the model on SCROLLS (Shaham et al., 2022), a popular long-context dataset, and Table 1 again demonstrates our effectiveness over H_2O.

4.3. Extending Context Length with Limited Memory

In this section, we set the memory size to 512 and extend the pre-trained context length of Llama-2 (Touvron et al., 2023) from 4096 to 8192, 16384, and 32768 using the Red-

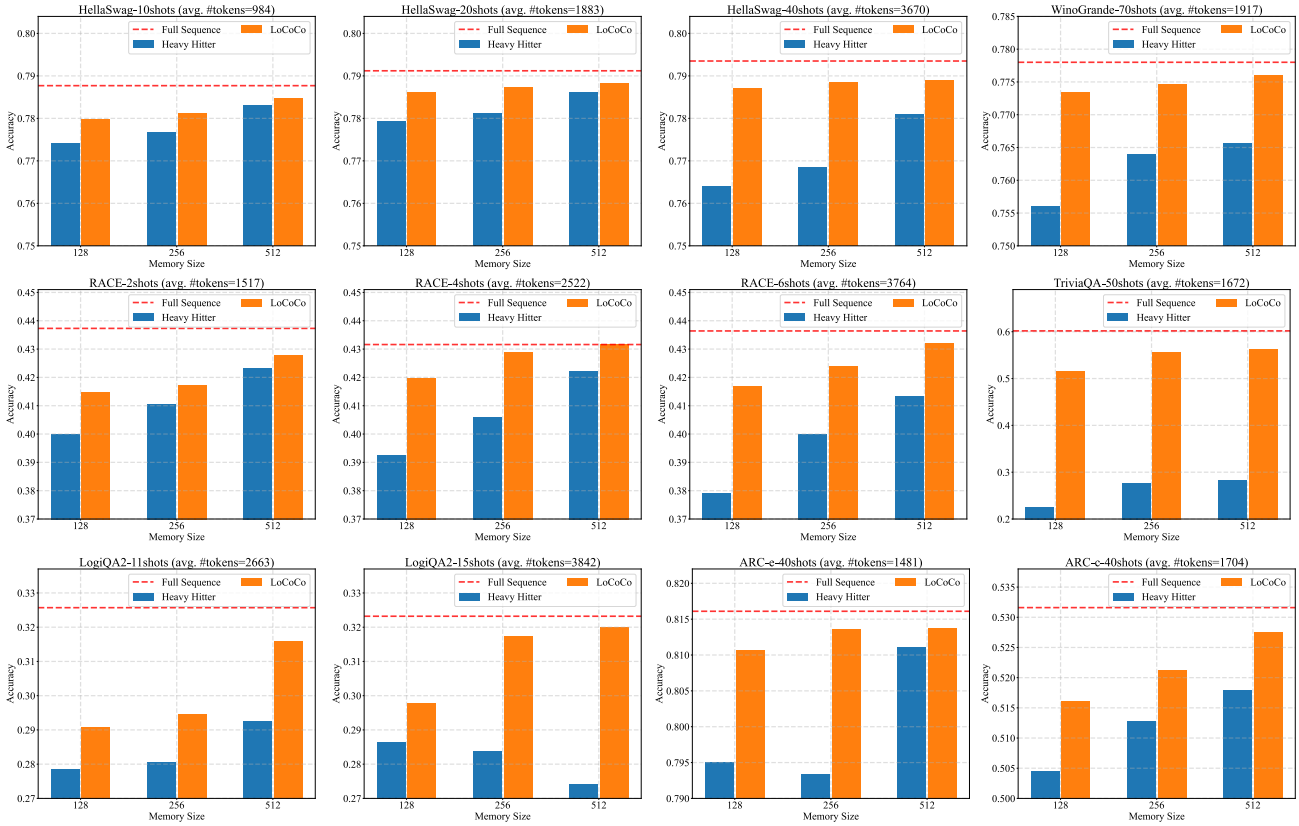

Figure 2. Token merging via convolutional kernels as the drop-in" integration without modifying the original weights. Based on Llama-2-7B (Touvron et al., 2023), we inserted the convolutional heads on the top of self-attention, and tested the model performance on various few-shot downstream tasks. The input sequence typically consists of about 2000 tokens. We compare our method with Zhang et al. (2023b), a token eviction strategy. We also provide the uncompressed case, where the model uses the full sequence.

Table 1. LoCoCo applied to the ChatGLM3-6B-32k (Du et al., 2021) base model, and validated on SCROLLS (Shaham et al., 2022).

SCORLLS Task	QuALITY	Qasper	SummScreen	GovReport	QMSum	NarrativeQA
H_2O	0.4351	0.3919	0.2498	0.3411	0.2137	0.2433
ours	0.4689	0.4284	0.2611	0.3617	0.2310	0.2576
full sequence	0.4769	0.4314	0.2636	0.3669	0.2378	0.2605

Pajama pre-training dataset (Computer, 2023). We conduct experiments on the 7B and 13B models and report perplexity on Proof-Pile-2 (Azerbayev et al., 2023). We also validate the model performance under shorter context lengths.

The results are provided in Table 2. Besides Zhang et al. (2023b), we also compare with StreamingLLM (Xiao et al., 2023), a method handling contexts longer than the pre-trained length in a zero-shot manner. Additionally, we compare with LongLoRA (Chen et al., 2023b), which utilizes only local tokens without considering global information. Finally, we evaluate the model tuned with uncompressed full sequence length. When combining our proposed token merging with eviction, our method demonstrates superior performance over the aforementioned methods, and shows comparable performance with the uncompressed scenario.

To further validate our effectiveness, we report our results on LongBench (Bai et al., 2023) in Table 4. We adopt Llama2-13b (Touvron et al., 2023) and extend the maximum context length to 32K. Compared to LongLoRA (Chen et al., 2023b) and H_2O (Zhang et al., 2023b), our method again achieves superior performance.

4.4. Memory and Throughput Measurement

We first test our GPU memory usage during training (tuning): the memory is measured when extending the context length of Llama2-7B to 16k. As shown in Table 5, performing training directly on the full sequence will exhaust all GPU memory (resulting in "OOM"). In contrast, our method

Table 2. Perplexity evaluated on Proof-Pile-2(Azerbayev et al., 2023). We fine-tuned Llama-2-7B (Touvron et al., 2023) to extend the context length from 4K to 8K, 16K, and 32K, respectively. Additionally, we fine-tuned Llama-2-13B, extending the 4K context length to 8K. T denotes the sequence length of the training data, whereas L indicates the chunk size.

Size	Training Length (T)	Method	Attention Complexity	Evaluation Context Length				
				2048	4096	8192	16384	32768
7b	8192	StreamingLLM	$O(L \times (L+8))$	4.0373	4.0174	4.0551	-	-
		LongLoRA	$O(L^2)$	4.0526	3.8111	3.6877	-	-
		H_2O	$O(L \times (L+512))$	3.9653	3.7043	3.5706	-	-
		Ours	$O(L \times (L+512))$	3.9411	3.6775	3.5414	-	-
		Full Sequence	$O(L \times T)$	3.9325	3.6558	3.5070		
	16384	StreamingLLM	$O(L \times (L+8))$	4.0373	4.0174	4.0551	4.0334	-
		LongLoRA	$O(L^2)$	4.0704	3.8125	3.6928	3.6279	-
		H_2O	$O(L \times (L+512))$	3.9842	3.7173	3.5974	3.5458	-
		Ours	$O(L \times (L+512))$	3.9628	3.6958	3.5763	3.5058	-
		Full Sequence	$O(L \times T)$	3.9491	3.6619	3.5094	3.4801	-
	32768	StreamingLLM	$O(L \times (L+8))$	4.0373	4.0174	4.0551	4.0334	4.0171
		LongLoRA	$O(L^2)$	4.0891	3.8348	3.7161	3.6276	3.5916
		H_2O	$O(L \times (L+512))$	4.0564	3.8179	3.6570	3.5634	3.5102
		Ours	$O(L \times (L+512))$	4.0253	3.8078	3.5807	3.5145	3.4408
		Full Sequence	$O(L \times T)$	3.9803	3.7703	3.5011	3.4836	3.4012
13b	8192	StreamingLLM	$O(L \times (L+8))$	3.6979	3.7013	3.7022	-	-
		LongLoRA	$O(L^2)$	3.7153	3.5902	3.4511	-	-
		H_2O	$O(L \times (L+512))$	3.6823	3.5482	3.4073	-	-
		Ours	$O(L \times (L+512))$	3.6798	3.4953	3.3697	-	-
		Full Sequence	$O(L \times T)$	3.6412	3.4506	3.3421	-	-

Table 3. Performance on representative long-context task SCROLLS. (Shaham et al., 2022)

SCORLLS Task	QuALITY	Qasper	SummScreen	GovReport	QMSum	NarrativeQA
LongLoRA	0.3395	0.2421	0.1712	0.2891	0.1792	0.1754
H_2O	0.3461	0.2659	0.1885	0.2924	0.1913	0.1849
LoCoCo	0.3528	0.2813	0.1903	0.3113	0.2089	0.1902
full sequence	0.3600	0.2828	0.1945	0.3125	0.2125	0.1942

Table 4. Evaluation on LongBench (Bai et al., 2023).

Method	LongLoRA	H_2O	LoCoCo
LongBench	34.7%	36.9%	37.4%

Table 5. Comparison on memory usage (during training) and throughput (during inference).

Method	LongLoRA	H_2O	LoCoCo	Full Sequence
Memory Usage	49GB	50GB	50GB	OOM
Throughput (Token/s)	25	32	33	11

only requires an additional 1GB of memory compared to LongLoRA (Chen et al., 2023b) and uses the same amount of memory as H_2O (Zhang et al., 2023b).

We then measure the throughput during inference, at the pre-filling stage. The pre-filling length is set to be 16k. As shown in Table 5, our method achieves superior throughput compared to all baselines at inference. For all aforementioned experiments, we set the batch size to 1, and the block size and the KV cache memory size to both 512. We use Flash Attention v2 (Dao, 2023) and DeepSpeed Stage 2 by default. The measurements are conducted on the NVIDIA A100 80GB GPU, confirming our inference efficiency.

5. Ablation

5.1. Effectiveness under Different Memory Sizes

We vary the memory size during fine-tuning, ranging from 128 to 1024, and compare our method with (Zhang et al., 2023b). Based on the pre-trained model Llama-2 (Touvron et al., 2023) whose maximal context length is 4096, we extend it to the length of 8192, on dataset RedPajama (Computer, 2023). We evaluate the models on Proof-Pile-2 (Azerbayev et al., 2023) in terms of perplexity. As in Figure 3, our method shows exceptional performance especially at large compression ratios, indicating that LoCoCo could generate more expressive compressed tokens compared to heavy-hitters.

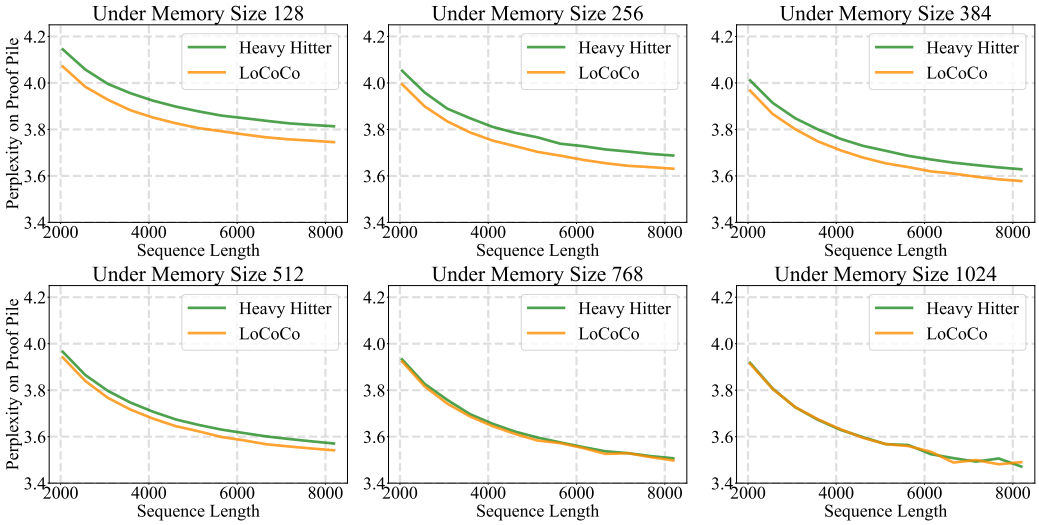

Figure 3. Varying memory sizes during fine-tuning, evaluated on Proof-Pile-2 (Azerbayev et al., 2023). Compared to (Zhang et al., 2023b), our method shows exceptional performance at large compression ratios, indicating the expressiveness of the merged token.

Table 6. Our method could be performed solely or combined with multiple token eviction methods.

Method	Perplexity
H_2O	3.5714
LoCoCo	3.5451
LoCoCo w. StreamingLLM	3.5439
LoCoCo w. H_2O	3.5414

5.2. Combination with Different Eviction Policies

Our method could either work alone or be integrated with any token eviction policy. In Table 6, to extend the maximum context length of Llama-2-7B to 8192 tokens, we showed our core idea of token merging via convolutional heads (1) works well alone; (2) could be combined with StreamingLLM (Xiao et al., 2023), by additionally storing the initial tokens, as known as "attention sink"; and (3) could be further augmented by heavy hitters(Zhang et al., 2023b), the "important tokens" identified by accumulated attention scores. All variants of our methods show superior performance to solely using the previous token eviction method (Zhang et al., 2023b).

5.3. Effectiveness under Different Kernel Sizes

Longer convolutional kernels may also present challenges in optimization. With the Llama-2-7B model (Touvron et al., 2023), we extend the context length to 8192, employing kernel sizes ranging from 3 to 21. We evaluate the fine-tuned model on Proof-Pile-2 (Azerbayev et al., 2023), using a context length of 8192. The results are summarized in Table 7. We observe stable performance for most size choices, although there are degradations with extremely small kernel sizes. That suggests LoCoCo can work well with moderately sized convolutions, without visible optimization hurdles.

Table 7. Ablation with different kernel sizes.

Kernel	3	7	17	21	31	41	51	61
PPL	3.68	3.57	3.53	3.53	3.54	3.53	3.57	3.58

6. Conclusions

This paper introduces LoCoCo, designed to improve both computation and memory efficiency when dealing with long-context inputs, through the use of a fixed-size KV Cache. We propose a data-driven adaptive token fusion technique, characterized by learnable convolutional kernels. LoCoCo is compatible with any pre-trained Language Models (LLMs), enabling seamless integration with low overhead. Experiments demonstrate that LoCoCo achieves a compression ratio of up to $32:1$ and outperforms baseline methods by up to 27.91% in accuracy.

Acknowledgements Portions of this research were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing[2]. The work is in part supported by the gift funding from `https://moffett.ai` (B. Chen) and the National AI Institute for Foundations of Machine Learning (Z. Wang).

Impact Statement

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

[2]https://hprc.tamu.edu/aces/

References

Ntk-aware scaled rope. https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/, 2023.

Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., and Ré, C. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

Azerbayev, Z., Schoelkopf, H., Paster, K., Santos, M. D., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. Llemma: An open language model for mathematics, 2023.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023a.

Chen, Y., Qian, S., Tang, H., Lai, X., Liu, Z., Han, S., and Jia, J. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023b.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Computer, T. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

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

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., and Tang, J. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*, 2021.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022.

Goyal, T. and Durrett, G. Evaluating factuality in generation with dependency-level entailment. *arXiv preprint arXiv:2010.05478*, 2020.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021a.

Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34: 572–585, 2021b.

Gu, A., Goel, K., Gupta, A., and Ré, C. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35: 35971–35983, 2022.

Gupta, A., Gu, A., and Berant, J. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

Han, C., Wang, Q., Xiong, W., Chen, Y., Ji, H., and Wang, S. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023.

Jin, H., Han, X., Yang, J., Jiang, Z., Liu, Z., Chang, C.-Y., Chen, H., and Hu, X. Llm maybe longlm: Self-extend llm context window without tuning. *arXiv preprint arXiv:2401.01325*, 2024.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Kamalloo, E., Dziri, N., Clarke, C. L., and Rafiei, D. Evaluating open-domain question answering in the era of large language models. *arXiv preprint arXiv:2305.06984*, 2023.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Kim, Y. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Massaroli, S., Poli, M., Fu, D. Y., Kumbong, H., Parnichkun, R. N., Timalsina, A., Romero, D. W., McIntyre, Q., Chen, B., Rudra, A., et al. Laughing hyena distillery: Extracting compact recurrences from convolutions. *arXiv preprint arXiv:2310.18780*, 2023.

Mohtashami, A. and Jaggi, M. Landmark attention: Random-access infinite context length for transformers, 2023a.

Mohtashami, A. and Jaggi, M. Random-access infinite context length for transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., et al. Scrolls: Standardized comparison over long language sequences. *arXiv preprint arXiv:2201.03533*, 2022.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6): 1–28, 2022.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Tworkowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Miłoś, P. Focused transformer: Contrastive training for context scaling. *arXiv preprint arXiv:2307.03170*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Xiong, W., Liu, J., Molybog, I., Zhang, H., Bhargava, P., Hou, R., Martin, L., Rungta, R., Sankararaman, K. A., Oguz, B., et al. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039*, 2023.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 14138–14148, 2021.

Yuan, A., Coenen, A., Reif, E., and Ippolito, D. Wordcraft: story writing with large language models. In *27th International Conference on Intelligent User Interfaces*, pp. 841–852, 2022.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, Q., Ram, D., Hawkins, C., Zha, S., and Zhao, T. Efficient long-range transformers: You need to attend more, but not necessarily at every layer. *arXiv preprint arXiv:2310.12442*, 2023a.

Zhang, Y., Li, J., and Liu, P. Extending llms' context window with 100 samples. *arXiv preprint arXiv:2401.07004*, 2024.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023b.

Zhu, D., Yang, N., Wang, L., Song, Y., Wu, W., Wei, F., and Li, S. Pose: Efficient context window extension of llms via positional skip-wise training. *arXiv preprint arXiv:2309.10400*, 2023.