CaM: Cache Merging for Memory-efficient LLMs Inference

Yuxin Zhang [1 2 *] **Yuxuan Du** [1 *] **Gen Luo** [1] **Yunshan Zhong** [1] **Zhenyu Zhang** [3] **Shiwei Liu** [4 5] **Rongrong Ji** [1 6]

Abstract

Despite the exceptional performance of Large Language Models (LLMs), the substantial volume of key-value (KV) pairs cached during inference presents a barrier to their efficient deployment. To ameliorate this, recent works have aimed to selectively eliminate these caches, informed by the attention scores of associated tokens. However, such cache eviction invariably leads to output perturbation, regardless of the token choice. This perturbation escalates with the compression ratio, which can precipitate a marked deterioration in LLM inference performance. This paper introduces Cache Merging (CaM) as a solution to mitigate this challenge. CaM adaptively merges to-be-evicted caches into the remaining ones, employing a novel sampling strategy governed by the prominence of attention scores within discarded locations. In this manner, CaM enables memory-efficient LLMs to preserve critical token information, even obviating the need to maintain their corresponding caches. Extensive experiments utilizing LLaMA, OPT, and GPT-NeoX across various benchmarks corroborate CaM's proficiency in bolstering the performance of memory-efficient LLMs. Code is released at `https://github.com/zyxxmu/cam`.

1. Introduction

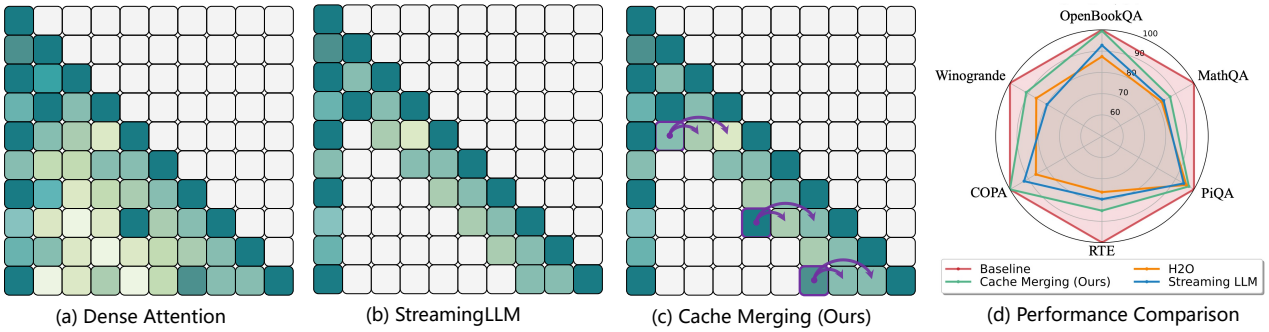
The unparalleled efficacy of Large Language Models (LLMs) across varied application domains has paved an unprecedented trajectory toward the realization of Artificial General Intelligence (AGI) (Brown et al., 2020; Touvron et al., 2023; Bommarito II & Katz, 2022). Nevertheless, deploying LLMs at inference time poses considerable financial and energetic burdens, chiefly attributed to their substantial scale that requires massive computational resources and GPU memory (Zhao et al., 2023). Consequently, the generative procedure of LLMs frequently integrates a Key-Value (KV) Cache mechanism intended to streamline the generation process. During each step of generation, the key-value embeddings from the attention module are retained in memory, thereby obviating the need for redundant key-value calculations in subsequent steps. Despite its utility, the memory footprint of the KV cache swells with the enlargement of the model dimensions and lengths of generation sequences, imposing significant demands on device memory capacity. Consider LLaMA-65B (Touvron et al., 2023) as an illustrative example: for a batch size of 128 and a sequence length of 2048, approximately 365GB of KV cache memory is required, which is nearly threefold that of the model's parameters (around 130GB). Therefore, advancing the memory efficiency of the KV cache, while simultaneously preserving the generative efficiency and quality of LLMs, represents a critical task.

A promising group of methods has recently emerged where a proportion of KV cache can be evicted at inference time to reduce memory consumption and accelerate token generation (Zhang et al., 2023b; Xiao et al., 2023b; Ge et al., 2023; Liu et al., 2023). Nevertheless, such token selection constitutes a challenging task since the eviction of salient tokens might cause critical information loss, thereby leading to notable performance declines (Zhang et al., 2023b). To date, the bulk of research is grounded on the attention scores respective to tokens, which ostensibly mirrors their influence on the output after cache eviction—expressed, *w.r.t*, $O = A \cdot V$, where O, A, and V symbolize the attention output, attention weights, and value cache, respectively. For instance, Zhang et al. (2023b) suggested a more adaptive selection method by preserving KV caches of tokens that accumulate higher attention scores throughout the generative process. Xiao et al. (2023b) contended the existence of "sink tokens" at the beginning of the sequence exhibits pronounce attention scores, thereby proposing StreamingLLM to simply keep the initial 4 tokens and the current window's KV cache.

Despite the ongoing efforts to substantially diminish the KV cache memory burden throughout LLM inference (Han

[*]Equal contribution [1]Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China, Xiamen University [2]Peng Cheng Laboratory [3]University of Texas at Austin [4]University of Oxford [5]Eindhoven University of Technology [6]Institute of Artificial Intelligence, Xiamen University. Correspondence to: Rongrong Ji <rrji@xmu.edu.cn>.

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



(a) Dense Attention (b) StreamingLLM (c) Cache Merging (Ours) (d) Performance Comparison

Figure 1: (a) Attention maps using dense attention. (b) StreamingLLM (Xiao et al., 2023b) mitigates the memory overhead by evicting token caches considered non-essential, determined by attention scores. (c) CaM merges caches of to-be-evicted tokens with higher attention score (purple frame) into subsequent tokens, hence (d) resulting in smaller output perturbations caused by cache eviction and leading to superior inference performance (LLaMA-7B with 20% memory budget).

et al., 2023; Xiao et al., 2023b; Zhang et al., 2023b), a conspicuous constraint persists. Fundamentally, regardless of the chosen token selection criteria, cache eviction invariably induces inferential discrepancies—a consequence inevitably linked to the essential linear dot product calculations underpinning the attention mechanism in LLMs. Such discrepancy will be greatly magnified with a large compression ratio, particularly for long sequence generation.

In response to this issue, this paper presents **Cache Merging (CaM)**, a new approach that strategically merges the intended-to-be-eliminated KV cache into the preserved cache, instead of permanently evicting them. The impetus behind CaM is rooted in the fundamental processes of attention computation: by establishing the merge rate in proportion to the attention score ratio between the token sets designated for eviction and those selected for merging, the output remains theoretically unchanged, *i.e.*, *LLM inference outcomes do not differ even cache memory is reduced*. Nevertheless, such lossless merging heavily relies on assigning merge ratios to reflect the attention proportion of tokens being evicted and those being merged. Such a ratio pertinent to future generation steps is unpredictable, thereby rendering the efficacy of merging difficult to ascertain.

Fortunately, we empirically observe that the mean attention spanning across a sequence of tokens exhibits significant constancy within future generation steps. This observation has prompted us to propose a strategy that evenly merges the cache destined for eviction into the remaining caches. Building on this, we further employ CaM a sampling decision based on the cumulative attention scores accrued by the to-be-evacuated token in comparison to other tokens, to ascertain its merging necessity. Through theoretical substantiation and empirical verification, we demonstrate that such an adaptive process for even cache merging invariably leads to less output perturbation than direct cache eviction. Utilizing CaM, we observed that the inferential performance of memory-efficient LLMs is markedly enhanced across diverse testing tasks, encompassing question-answering, text summarization, and extensive language modeling tasks, as illustrated in Fig 2 (d). For example, CaM enhances StreamingLLM (Xiao et al., 2023b) by 5.1% in zero-shot accuracy on the RTE (Wang et al., 2018) benchmark while retaining 20% of the cache within LLaMA-7B (Touvron et al., 2023). The contributions of this paper are delineated as follows:

- We propose CaM to merge caches scheduled for eviction back into the residual caches to reduce the output discrepancies caused by cache eviction. This represents the first concept of cache merging within the domain of LLM inference, to our best knowledge.
- Extensive experimental results demonstrate that CaM effectively bolsters the performance of memory-efficient LLMs over a wide spectrum of tasks and sequence length conditions.
- We provide comparative studies to substantiate the strategic rationale behind cache merging. Hopefully, these results can serve as both a useful tool for future research in cache merging for memory-efficient LLMs.

2. Related Work

LLMs Inference Breakdown. The memory load of LLMs at inference is attributed by three primary components: the volume of model parameters, the size of the activation buffer, and the size of the KV cache. Model compression techniques, inclusive of parameter quantization (Lin et al., 2023; Frantar et al., 2022; Dettmers et al., 2022; Xiao et al., 2023a) and network pruning (Frantar & Alistarh, 2023; Sun et al., 2023; Zhang et al., 2023a; Yin et al., 2023), have demonstrated impressive progress in addressing the former two components. Analogously, the memory overhead of KV

caches in LLMs is linked to the models' hidden dimensions, yet with a greater dependency on sequence lengths and batch sizes during inference. The requisite memory for the KV cache proportionally increases as the previous key-value embeddings in attention are stored persistently to generate new tokens. In scenarios involving extended sequences, such as those spanning 2048 tokens, memory consumption typically escalates to 2.5 to 5 times the size of the model itself (Ge et al., 2023; OpenAI, 2023).

KV Cache Compression. Efforts to alleviate the KV cache burden in LLM inference have been broadly divided into two categories: introducing sparsity to discard non-essential tokens and reducing the bit requirements for each KV embedding. For the first category, notable works like H_2O (Zhang et al., 2023b) and Scissorhands (Liu et al., 2023) identifying important tokens based accumulated attention scores (Wang et al., 2021), while TOVA (Oren et al., 2024) suggests utilizing the recent attention pattern. StreamingLLM (Xiao et al., 2023b) and LM-Infinite (Han et al., 2023) propose the concept of 'sink tokens', *i.e.*, the beginning of the sequences, suggesting the removal of KV caches for intermediate tokens. Additionally, (Ge et al., 2023) investigates attention importance across different attention heads and layers, and prosposes an adaptive KV cache eviction policy. The second category aims to reduce the bits needed for storing KV embeddings, such as FlexGen (Sheng et al., 2023) and KIVI (Liu et al.) that recommend quantizing keys per channel and values per token. Meanwhile, recent approaches like landmark tokens (Mohtashami & Jaggi, 2023) or Gist tokens (Mu et al., 2023) compress KV cache costs by setting special tokens, though they require extra training and cannot be seamlessly integrated into existing LLMs. Distinct from previous methods, our Cache Merging (CaM) method presents a novel approach by merging caches to minimize information loss during compression. A closely related work, SparQ (Ribar et al., 2023), enhances KV cache eviction with a global embedding that is averaged across all tokens. However, averaging all tokens risks compromising the original output of the current token. Our CaM method, therefore, offers a unique and potentially more effective solution to the challenges of KV cache compression.

Token Prune and Merge in ViTs. Within the Vision Transformer (ViT) domain, compressing superfluous image tokens has been an extensively explored research topic (Rao et al., 2021; Yin et al., 2022; Bolya & Hoffman, 2023; Bolya et al., 2023). Early inquiries aimed at creating metrics for pinpointing and then discarding redundant tokens (Rao et al., 2021; Yin et al., 2022). The advent of Token Merge (ToMe) (Bolya et al., 2023; Bolya & Hoffman, 2023) introduced a pivotal advancement, effectively minimizing the performance degradation associated with token pruning by merging said visual tokens. Chen et al. (2023) delved into optimizing hyperparameters and calculating merge ratios. Different from previous work, CAM for the first time explores the possibility of merging the KV cache produced in the generation step, serving as a starting point for future research in KV cache merging for memory-efficient LLMs.

3. Methodology

3.1. Preliminary

We first give a basic preliminary about the memory usage for storing attention keys and values in LLMs inference. LLMs are typically built with transformer layer (Vaswani et al., 2017). Here we only consider one attention head within a specific layer for simplicity. Let the attention module weights be $W_q \in \mathbb{R}^{d\times d}$, $W_k \in \mathbb{R}^{d\times d}$, $W_v \in \mathbb{R}^{d\times d}$, where d represents the hidden dimension of the model. LLM inference follows an autoregressive fashion, generating one token at each step conditioned on the previous steps. At each step, the key-value embedding in attention is stored in memory to avoid repetitive key-value projection computation at future steps. Denote the the key and value cache as K, V, given prompt input of a specific layer in LLMs as $X_{\text{prompt}} = [x_1, ..., x_p]$, where p is the prompt length, the KV cache is initialized as

$$K = X_{\text{prompt}} W_k, V = X_{\text{prompt}} W_v. \tag{1}$$

Let the input of the attention module at generation step t be $x_t \in \mathbb{R}^d$, where we omitted the batch dimension for simplicity. x_t is firstly mapped into a set of queries, keys, and values $\{Q_t, K_t, V_t\}$ as

$$Q_t = x_t W_q, K_t = x_t W_k, V_t = x_t W_v. \tag{2}$$

Then, the KV cache gets updated as $K = [K, K_t], V = [V, V_t]$. Finally, the t-th step outputs O^t is computed by first calculating the attention weights at t-th step $A^t \in \mathbb{R}^t$ as

$$A^t = \text{softmax}(Q_t^\top K), \tag{3}$$

where softmax is the softmax function, and O is derived as

$$O^t = \sum_{k=1}^{t} A_k^t V_k. \tag{4}$$

Despite the effectiveness of KV caching mechanism in circumventing redundant computation of key-value projections in LLMs inference, it incurs considerable memory consumption—markedly in instances with extended sequences that may encompass thousands of tokens (Rae et al., 2019). Consequently, devising methods to diminish the cache's memory footprint without compromising generative quality remains an imperative challenge.

3.2. Revisiting KV Cache Eviction

Extensive research endeavors have been undertaken to reduce the burden of Key-Value (KV) cache without com-

promising the quality of inference (Zhang et al., 2023b; Xiao et al., 2023b; Liu et al., 2023; Ge et al., 2023). The fundamental goal of KV cache compression lies in preserving the fidelity of outputs obtained with a full cache when certain KV pairs are excised from the cache. Formally, upon the eviction of a selected KV cache pair K_i and V_i, the resulting perturbation Δ^t in output can be quantified using Eq. (4) as:

$$\Delta^t = -A_i^t V_i. \tag{5}$$

Remark 3.1. Prevailing studies (Zhang et al., 2023b; Liu et al., 2023; Xiao et al., 2023b) predominantly utilize the magnitude of the attention scores as a criterion for determining which caches should be discarded. These approaches are theoretically supported by Eq. (5), as removing caches corresponding to tokens with lower attention scores ostensibly reduces variations in the output.

Despite the ongoing advancements, we contend that the current methods employed for the eviction of the KV cache exhibit an obvious limitation. Independent of the metric used for token selection, the eviction of entries from the cache inherently causes fluctuations in the output, as inferred from Eq. (5). Concomitantly, an increase in the compression ratio of the KV cache exacerbates output perturbations, consequently precipitating a marked deterioration in the performance of LLM during inference, as shown in Figure 1 (d).

3.3. Cache Merging

In response to this issue, we introduce Cache Merge (CaM) to merge caches that is designed to be evicted into the remaining caches, instead of the customary eviction process found in prior works. The impetus for CaM originates from the theorem delineated hereunder.

Theorem 3.2. *Consider indices i and j corresponding to tokens whose associated KV cache pairs are designated for eviction and retention, respectively. The perturbation in output at the t-th generative step, denoted by Θ^t, is nullified when the value cache V_i is merged into V_j, via an updated cache $\bar{V}_j = V_j + \frac{A_i^t}{A_j^t} V_i$.*

Proof. We can express the output perturbation as follows:

$$\begin{aligned}
\Theta^t &= \sum_{k=1}^{t} A_k^t V_k - \left(\sum_{\substack{k=1 \\ k\neq i,j}}^{t} A_k^t V_k + A_j^t \bar{V}_j \right) \\
&= \sum_{k=1}^{t} A_k^t V_k - \left(\sum_{\substack{k=1 \\ k\neq i,j}}^{t} A_k^t V_k + A_j^t \left(V_j + \frac{A_i^t}{A_j^t} V_i \right) \right) \\
&= 0. \qquad \blacksquare
\end{aligned} \tag{6}$$

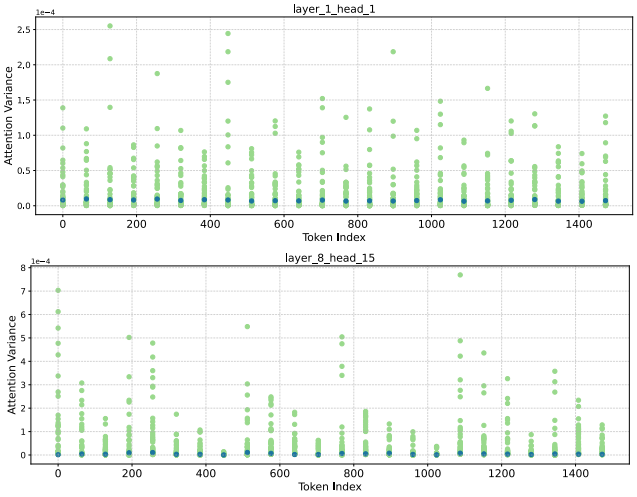


Figure 2: Variance on attention weights over sequence generation. (Llama-7b on the first sample of the XSUM testset). We divided tokens into multiple consecutive groups for visualization, each group includes 64 tokens. Take x-axis of 0 for example, the blue point represents the variance of mean attention across consecutive 64 tokens within [0,64] indexes, and the green points draw the attention variance of those individual tokens. The average attention weights of consecutive tokens hold significantly smaller variances.

This theorem demonstrates that it is feasible to alleviate the output perturbation engendered by cache eviction—*even achieving a state of zero loss.* The updating strategy for V_j assumes a prerequisite comprehension of the proportional distribution of attention scores among the tokens undergoing merging. This understanding appears to be absent in subsequent generative processes after we merge V_i. In real cases, the perturbation of the output at the $(t+1)$-th generation step, denoted as Θ^{t+1}, can be inferred from Equation (6) as follows:

$$\begin{aligned}
\Theta^{t+1} &= A_j^{t+1}(V_j + rV_i) - A_j^{t+1} V_j - A_i^{t+1} V_i \\
&= (rA_j^{t+1} - A_i^{t+1}) V_i,
\end{aligned} \tag{7}$$

where r represents the merge ratio of V_i into V_j. It can be referred that Θ^{t+1} equates to zero if and only if $r = A_i^{t+1}/A_j^{t+1}$. Conversely, lets say the worst situation, the output perturbation may even exceed the magnitude of the perturbation caused by the direct eviction of V_i, as demonstrated by

$$|\Theta^{t+1}| - |\Delta^{t+1}| = |(rA_j^{t+1} - A_i^{t+1}) V_i| - |-A_i^{t+1} V_i| > 0, \tag{8}$$

with Δ^{t+1} representing the perturbation resulting from the direct eviction of V_i, according to Eq. (5). To encapsulate, while cache merging serves as a potential strategy to drastically reduce the output perturbation caused by cache eviction, it equally holds the propensity to amplify output

discrepancies. Consequently, we engineer CaM to not only minimize Θ^{t+1} but also ensure that $|\Theta^{t+1}| - |\Delta^{t+1}| < 0$, thereby perpetually endowing the advantageous of cache merging for memory-efficient LLM inference.

In particular, Eq.(7) suggests that the crux of minimizing the output perturbations hinges upon the precise estimation for r that equalizes the proportions between A_j^{t+1} and A_i^{t+1}. However, precisely predicting the evolution of a given token's future attention as the inferential sequence advances proves empirically arduous. Fortuitously, our observation, typified in Figure 2, indicates that when considering the mean attention across an extent of m continuous tokens, there manifests a stable trend. This insight has propelled us to transition from innovate a shift from merging into a single cache V_j to merging into an expanse of multiple continuous tokens' value caches, encapsulated by $V_j, V_{j+1}, \ldots, V_{j+m}$, with m characterizing the span of local tokens. The merging strategy of CaM is thus formulated as follows:

$$\bar{V}_k = V_k + \frac{V_i}{m}, \quad \text{for } k = j, \ldots, j+m, \tag{9}$$

where $\bar{V}_k$ represents the merged token. Consequently, the perturbation of the output at the $(t+1)$-th generation step is given by:

$$\begin{aligned} \Theta^{t+1} &= \sum_{k=j}^{j+m} \frac{V_i}{m} A_k^{t+1} - A_i^{t+1} V_i \\ &= \left(\text{avg}\left(A_{j:j+m}^{t+1}\right) - A_i^{t+1}\right) V_i. \end{aligned} \tag{10}$$

Minimizing Eq.(10) entails aligning A_i^{t+1} with $\text{avg}\left(A_{j:j+m}^{t+1}\right)$, yet A_i^{t+1} still exhibits inherent variability and may fluctuate considerably. We contend, however, that it is feasible to selectively merge tokens to circumvent situations wherein the perturbations resulting from cache merges surpass those from evictions. This hypothesis is supported by the following theorem.

Theorem 3.3. *Cache merging induces a smaller output perturbation than cache eviction, that is,* $\left|\Theta^{t+1}\right| < \left|\Delta^{t+1}\right|$ *provided that* $\text{avg}(A_{j:j+m}^{t+1}) < 2A_i^{t+1}$.

Proof. The inequality $\left|\Theta^{t+1}\right| < \left|\Delta^{t+1}\right|$ can be inferred from Eq.(10) and Eq. (5) as follows:

$$|(avg(A_{j:j+m}^{t+1}) - A_i^{t+1})V_i| < |-A_i^{t+1}V_i|. \tag{11}$$

Dividing both sides of the equation by $|V_i|$ yields:

$$\left|\text{avg}(A_{j:j+m}^{t+1}) - A_i^{t+1}\right| < \left|A_i^{t+1}\right|. \tag{12}$$

Given that the attention scores in a Softmax function are exclusively positive, we can drop the absolute values, obtaining:

$$0 < \text{avg}(A_{j:j+m}^{t+1}) < 2A_i^{t+1}. \tag{13}$$

Algorithm 1 Cache Merging at t-th Generation Step

Input: Attention weights A, V-Cache V, $i, j, m \in \mathbb{N}$
Let i denote the index of to-be-evicted cache
Let j denote the first index of local tokens
Let m denote the number of to-be-merged tokens
$\bar{A} = \sum_{k=1}^{t} A^k$
$M = \text{Bernoulli}(\text{clamp}(\frac{\bar{A}_i}{\text{avg}(\bar{A}_{j:j+m})}, 0, 1))$ ◁ Eq. (14)
for $k = j$ to $j+m$ **do**
  $\bar{V}_k = V_k + M\frac{V_i}{m}$ ◁ Eq. (10)
  $V_k = \bar{V}_k$
end for
Output: Updated V-Cache V

With $\text{avg}(A_{j:j+m}^{t+1}) > 0$, the proof is complete. ■

Building upon this theorem, we advance CaM to use a binary merging mask M for ascertaining whether a cache should be amalgamated by employing the subsequent probabilistic model:

$$M = \text{Bernoulli}(\text{clamp}(\frac{\bar{A}_i}{\text{avg}(\bar{A}_{j:j+m})}, 0, 1)), \tag{14}$$

where $\text{Bernoulli}(\cdot)$ represents the Bernoulli sampling function and $\text{clamp}(\cdot)$ signifies the clamp operation, correspondingly. The term $\bar{A} = \sum_{k=1}^{t} A^k$ denotes the cumulative attention score across previous generation steps, which supersedes the conventional individual attention at a single step and has been demonstrated to be more robust in predicting future attention as detailed in (Zhang et al., 2023b). The process of cache merging is subsequently realized through the following operation:

$$\bar{V}_k = V_k + M\frac{V_i}{m}, \quad \text{for } k = j, \ldots, j+m. \tag{15}$$

The sampling of M affords a judicious enhancement of merge choices, particularly when $2\bar{A}_i^t$ surpasses $\text{avg}(\bar{A}_{j:j+m}^t)$, thereby attenuating perturbations in the output than cache eviction. The ideal circumstance is manifested when $\bar{A}_i = \text{avg}(\bar{A}_{j:j+m})$, where Eq.(10) intimates the realization of lossless compression. Conversely, a diminutive $\bar{A}_i$ severely curtails the likelihood of merging, thus eschewing errors that could potentially exceed those entailed by eviction.

The workflow of our proposed CaM method is delineated in Algorithm 1. It is worth noting that CaM does not constitute a selection algorithm for token eviction; instead, it integrates the value cache of evicted tokens exhibited in prevailing methods (Xiao et al., 2023b; Zhang et al., 2023b; Liu et al., 2023). Consequently, the index of the evicted token is incorporated as an input to CaM. Moreover, CaM uniformly amalgamates the cache with local tokens, *w.r.t*, tokens proximal to the current generated token. Since preserving local

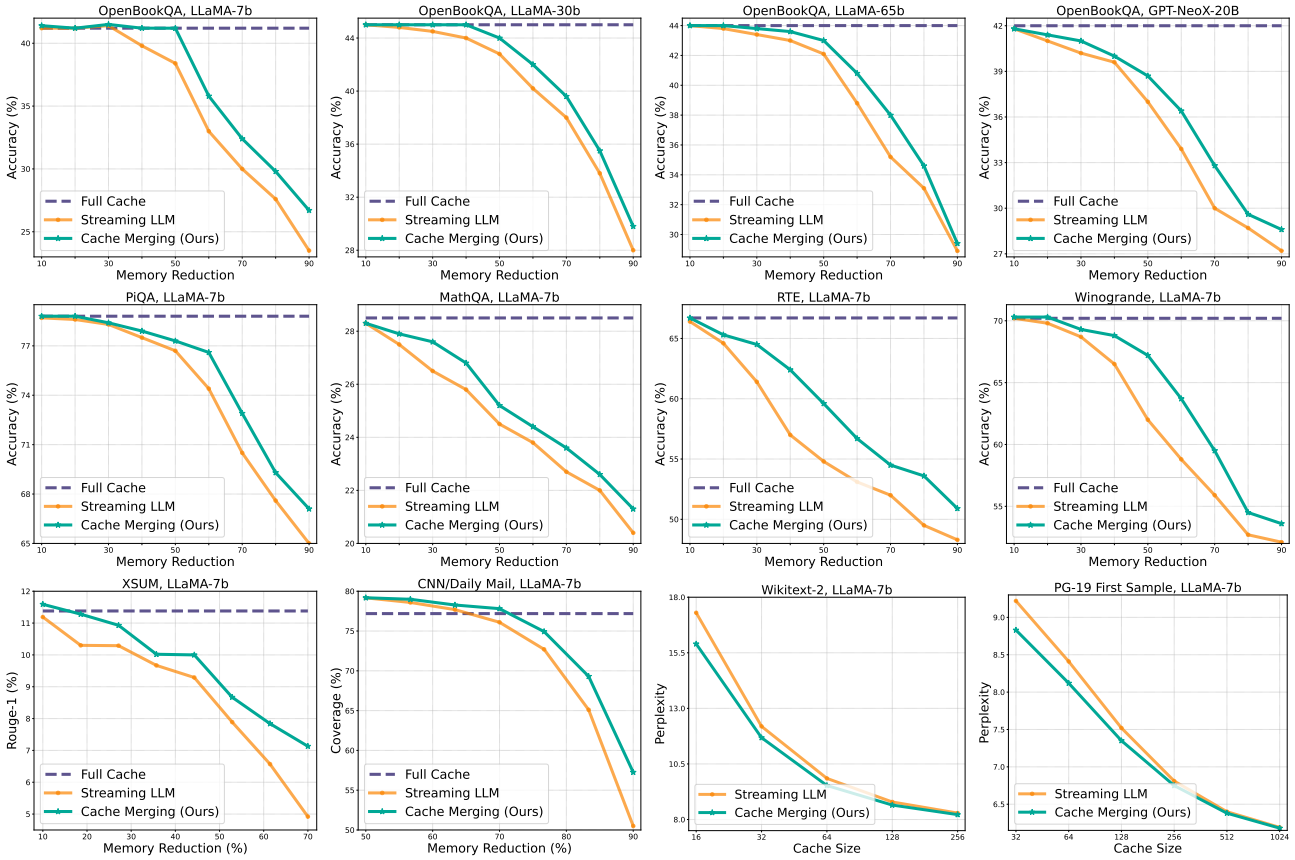


Figure 3: Performance comparison between full KV cache memory, Streaming LLM, and our proposed CaM for compressing KV cache of mainstream LLMs on multiple language tasks.

tokens is a consistent feature in all extant methods, CaM has considerable adaptability, facilitating its application across a spectrum of existing cache compression methods, and markedly augmenting their efficacy for memory-efficient LLMs inference. In addition, CaM is also highly efficient as it does not involve any matrix multiplication operations, as we demonstrate in the subsequent section.

4. Empirical Evaluation

In this section, we quantitatively examine the effectiveness of the proposed CaM method for enhancing the performance of memory-efficient LLMs. Initially, Section 4.1 summarily delineates the experimental settings. Subsequently, in Section 4.2, we employ the Streaming LLM (Xiao et al., 2023b)—a pioneer profiling method in the domain of memory-efficient LLMs that preserves the initial and recent token caches—to sweep the efficacy of CaM over a wide spectrum of LLMs and benchmarks. In Sec 4.3, we further quantitatively compare the performance of CaM when amalgamated with other KV cache compression methods (Zhang et al., 2023b; Liu et al., 2023) and show that it can achieve state-of-the-art performance. Conclusively, we undertake performance analysis to ablate the strategic merge decisions embodied in CaM, coupled with efficiency assessment.

4.1. Experimental settings

Tasks. We conduct experiments on representative tasks for LLM evaluation including question-answering, text summarization, and language modeling. Respectively, for question answering, we test six tasks using lm-eval-harness (Gao et al., 2021) framework: COPA (Roemmele et al., 2011), MathQA (Amini et al., 2019), OpenBookQA (Mihaylov et al., 2018), PiQA (Bisk et al., 2020), RTE (Wang et al., 2018), Winogrande (Sakaguchi et al., 2021). For text summarization, we use HELM framework to evaluate the XSUM (Narayan et al., 2018), and CNN/DailyMail (Nallapati et al., 2016) tasks. For language modeling, we evaluate the Perplexity performance of LLMs on Wikitext-2 (Merity et al., 2016) and PG-19 (Rae et al., 2019) datasets.

Model and baselines. Our evaluation is based on two representative model families of LLMs, including the OPT (Zhang et al., 2022), LLaMA (Touvron et al., 2023) and GPT-NeoX (Black et al., 2022) with model sizes ranging from 7 billion to 65 billion. We choose three baseline

Table 1: Performance comparison between different methods *w.* or *w.o.* CaM. Experiments are conducted with zero-shot evaluation under 20% KV cache budget.

Model	Method	OpenBookQA	PiQA	Winogrande	COPA	RTE	MathQA
LLaMA-7B	StreamingLLM	30.0	67.6	52.7	64.0	49.5	21.9
LLaMA-7B	***w.* CaM**	**31.8±0.2**	**69.8±0.2**	**54.5±0.1**	**68.0±0.0**	**54.6±0.2**	**22.6±0.1**
LLaMA-7B	H2O	22.8	66.9	50.3	47.0	48.3	22.1
LLaMA-7B	***w.* CaM**	**27.6±0.1**	**67.2±0.2**	**50.9±0.1**	**56.0±0.0**	**49.2±0.1**	**22.4±0.0**
LLaMA-7B	Scissorhands	22.0	50.3	51.6	58.0	51.6	22.1
LLaMA-7B	***w.* CaM**	**25.4±0.2**	**51.0±0.1**	**52.1±0.2**	**60.0±0.0**	**53.4±0.1**	**22.7±0.2**
OPT-13B	StreamingLLM	29.4	68.9	52.4	67.0	51.6	21.1
OPT-13B	***w.* CaM**	**31.4±0.1**	**69.4±0.1**	**53.5±0.2**	**71.0±0.0**	**56.6±0.3**	**21.7±0.1**
OPT-13B	H2O	24.0	60.6	47.9	61.0	53.0	22.2
OPT-13B	***w.* CaM**	**25.2±0.2**	**61.7±0.1**	**49.4±0.0**	**64.0±0.0**	**54.2±0.1**	**22.6±0.1**
OPT-13B	Scissorhands	31.0	53.3	54.2	65.0	51.6	21.5
OPT-13B	***w.* CaM**	**32.1±0.1**	**55.2±0.1**	**55.7±0.2**	**66.0±0.0**	**52.1±0.1**	**22.1±0.2**

method for comparison, including StreamingLLM (Xiao et al., 2023b), H2O (Zhang et al., 2023b), and Scissorhands (Liu et al., 2023).

Implementation details. For the implementation of CaM, we evenly merge the to-be-evicted value cache into all local tokens across all transformer layers. Given that all extant KV cache compression approaches conserve local tokens, CaM can be ubiquitously applied to them in a consistent manner. All experiments were conducted on NVIDIA Tesla A800 GPUs, utilizing five distinct random seeds, from which we report both the mean and variance of the results.

4.2. Model and Task Sweep

We first evaluate the efficacy of CaM when merging caches evicted by StreamingLLM (Xiao et al., 2023b). Figure 3 lists the performance comparisons for compressing KV cache memory at ratios ranging from 10% to 90%. With different KV cache budgets, CaM substantially enlarges the performance of StreamingLLM over a wide range of mainstream LLMs with model sizes ranging from 6.7 billion to 65 billion. Such performance improvements persist across a myriad of task types and sequence lengths, ranging from hundreds (*question-answering*), to thousands (*text summarization*), and up to tens of thousands (*language modeling on PG-19*). These results prove the effectiveness of CaM in enhancing the performance of memory-efficient LLMs.

Analysis. It is discernible that the enhancements conferred by CaM are more pronounced in challenging tasks such as text summarization and question-answering, exemplified by benchmarks like XSUM and MathQA, whereas in comparatively simpler tasks such as language modeling, StreamingLLM inherently maintains good performance. This phenomenon appears logical: language modeling primarily requires the retention of local tokens to support the generation of subsequent tokens. In contrast, for text summarization and question-answering, critical information is dispersed throughout the entire sequence, making the eviction of token caches from intermediate positions result in significant output perturbation. This elucidates the particular effectiveness of CaM for these tasks as it reintegrates caches containing vital information, which would have otherwise been discarded by Streaming LLM, thereby mitigating the performance decline caused by cache eviction.

4.3. Comparison with Other Works

Table 1 further elucidates the results of applying the proposed CaM to an expanded set of existing baselines, including H2O (Zhang et al., 2023b) and Scissorhands (Liu et al., 2023). These methods selectively preserve preceding caches of tokens by looking at their attention scores. Since these strategies inherently retain local tokens, the application policy for CaM remains consistent. Empirical evidence illustrates that CaM invariably elevates the inferential performance of all methods over a spectrum of tasks. For instance, CaM manages to augment the performance of H2O in removing the memory budget of LLaMA-7b, yielding improvements by 4.8%, 3.3%, and 9.0% on OpenBookQA, PiQA, and COPA datasets, respectively. These results imply that CaM is a versatile and scalable approach, complementary to current techniques aimed at ameliorating the output fluctuations engendered by cache eviction.

4.4. Performance Analysis

In Table 2, we ablate the merge choices made in CaM. Specifically, *w.o. Avg Merge* means that the evicted cache is merged to a randomly selected local token, whereas *w.o.*

Table 2: Ablation studies of CaM. We use StreamingLLM as baseline and respectively remove each components of CaM to investigate the performance change.

Method	OpenBookQA	COPA	RTE
Baseline	30.0	64.0	49.5
w.o. Avg Merge	30.4±0.3	66.0±0.0	52.1±0.3
w.o. Merge Mask	30.2±0.0	63.0±0.0	48.1±0.0
w.o. Acc. Attention	31.5±0.4	67.0±1.0	54.5±0.2
CaM	31.8±0.2	68.0±0.0	54.6±0.2

Table 3: Accuarcy comparison of setting different clamp intervals when sampling the merging mask. Experiments are conducted with 20% KV cache budget on LLaMA-7b, utilizing StreamingLLM as the baseline of CaM.

Clamp Interval	OpenBookQA	COPA	RTE
Baseline	30.0	64.0	49.5
$[0.5, 1]$	30.3±0.1	65.0±0.0	52.7±0.1
$[0, 0.5]$	30.8±0.2	64.0±0.0	51.9±0.1
$[0, 0.75]$	32.0±0.3	67.0±0.0	53.5±0.2
$[0, 1]$	31.8±0.2	68.0±0.0	54.6±0.2

Merging Mask implies that the merging operation is performed on all caches marked for eviction. In the case of *w.o. Acc. Attention*, only the attention from the current step is utilized for deriving the merging mask. The results reveal that both average merging and accumulating attention confer performance benefits by stabilizing the estimation of attention errors. Notably, the absence of merging mask significantly diminishes the inferential performance, even degrading it below the baseline. Such a phenomenon aligns with our analysis from Theorem 3.3 that indiscriminately merging all tokens earmarked for removal could introduce an output fluctuation that surpasses the impact of their direct eviction. In summary, it can be deduced that each step within CaM plays a pivotal role in enhancing the performance of memory-efficient LLMs.

As the merging mask serves as a pivotal factor in CaM, we delve further into the performance impacts exerted by varying clamping intervals within Eq. (14). Table 3 draws the results across three downstream tasks. Intuitively, elevating the lower limit of the interval results in an increased number of merged tokens, and vice versa. It is observable that adjusting the upper limit of the clamp range can enhance performance in certain tasks. However, setting a high lower limit invariably precipitates a decline in performance. These results suggest that selectively merging the removable cache is vitally important. In this paper, we adopt a consistent clamping interval of [0,1] to ensure that CaM remains a flexible and easy-to-use approach. Undoubtedly, these findings highlight the design of the merge mask as a highly promising avenue for future research.

Table 4: Generation throughput and GPU memory consumption on an NVIDIA 3090 GPU. We use the first sample of the PG-19 test set for evaluation, where all methods are under memory budget of 1024 tokens. "OOM" means out-of-memory. We use StreamingLLM as the baseline of CaM.

Method	Throughput	GPU memory
Full cache	-	OOM
StreamingLLM	26.1 ms/token	19.1GB
CaM	27.3 ms/token	19.3GB

Finally, we turn our attention to the efficiency of CaM. As delineated in Table 4, applying CaM in StreamingLLM does not notably reduce the generation speed nor increase memory usage. This is intuitive, given that CaM does not introduce any complex matrix operations or iterative loops. Coupled with previous experimental findings, we posit that CaM stands as a plug-and-play approach that can effectively enhance the performance of existing methods while realizing memory-efficient LLMs.

5. Limitation

We further discuss unexplored limitations of our proposed CaM method, which will be our future focus. First, cache merging entails both benefits and risks for preserving LLM inference performance as discussed before. While the proposed adaptive merge strategy based on attention score sampling has been theoretically and experimentally validated to be robust in diminishing the output perturbation, devising more advancing selection of being merged tokens and merging ratio remains an avenue worthy of exploration. Meanwhile, since the attention distribution and variance of tokens vary across different layers of LLMs, there is also substantial scope for investigation into layer-wise cache merge adaptability techniques.

6. Conclusion

Deploying LLMs in a memory-efficient manner is urgently needed but comes with challenges due to diminished performance consequent to the direct eviction of cached key-value pairs. In this paper, we introduce Cache Merging (CaM), which integrates caches slated for eviction with extant ones to bolster the efficacy of prevailing key-value cache compression methodologies. The potency of CaM has been corroborated by comprehensive experiments encompassing question-answering, text summarization, and language modeling, achieving state-of-the-art results in each task. We anticipate that CaM will not only equip practitioners with a robust LLM inference tool but also lay the groundwork for subsequent explorations into cache merge for memory-efficient LLMs.

Acknowledgement

This work was supported by National Key R&D Program of China (No.2022ZD0118202), the National Science Fund for Distinguished Young Scholars (No.62025603), the National Natural Science Foundation of China (No. U21B2037, No. U22B2051, No. 62176222, No. 623B2088, No. 62176223, No. 62176226, No. 62072386, No. 62072387, No. 62072389, No. 62002305 and No. 62272401), and the Natural Science Foundation of Fujian Province of China (No.2021J01002, No.2022J06001).

Impact Statement

This paper presents a CaM method to boost the inference performance of memory-efficient Large Language Models (LLMs). The LLMs community may benefit from our research. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Amini, A., Gabriel, S., Lin, S., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL https://aclanthology.org/N19-1245.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 34, pp. 7432–7439, 2020.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

Bolya, D. and Hoffman, J. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4598–4602, 2023.

Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. Token merging: Your vit but faster. *International Conference on Learning Representations (ICLR)*, 2023.

Bommarito II, M. and Katz, D. M. Gpt takes the bar exam. *arXiv preprint arXiv:2212.14402*, 2022.

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 (NeurIPs)*, 33:1877–1901, 2020.

Chen, M., Shao, W., Xu, P., Lin, M., Zhang, K., Chao, F., Ji, R., Qiao, Y., and Luo, P. Diffrate: Differentiable compression rate for efficient vision transformers. *arXiv preprint arXiv:2305.17997*, 2023.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems (NeurIPs)*, 2022.

Frantar, E. and Alistarh, D. Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning (ICML)*, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training compression for generative pretrained transformers. In *International Conference on Learning Representations (ICLR)*, 2022.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 2021.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

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.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. Kivi: Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Mohtashami, A. and Jaggi, M. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*, 2023.

Mu, J., Li, X. L., and Goodman, N. Learning to compress prompts with gist tokens. *arXiv preprint arXiv:2304.08467*, 2023.

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*, 2018.

OpenAI. Gpt-4 technical report. 2023.

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

Rae, J. W., Potapenko, A., Jayakumar, S. M., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., and Hsieh, C.-J. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

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.

Roemmele, M., Bejan, C. A., and Gordon, A. S. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pp. 90–95, 2011.

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.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D. Y., Xie, Z., Chen, B., Barrett, C., Gonzalez, J. E., et al. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*, 2023.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 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, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, H., Zhang, Z., and Han, S. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 97–110. IEEE, 2021.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, pp. 38087–38099. PMLR, 2023a.

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

Yin, H., Vahdat, A., Alvarez, J. M., Mallya, A., Kautz, J., and Molchanov, P. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10809–10818, 2022.

Yin, L., Wu, Y., Zhang, Z., Hsieh, C.-Y., Wang, Y., Jia, Y., Pechenizkiy, M., Liang, Y., Wang, Z., and Liu, S. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *arXiv preprint arXiv:2310.05175*, 2023.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, Y., Zhao, L., Lin, M., Sun, Y., Yao, Y., Han, X., Tanner, J., Liu, S., and Ji, R. Dynamic sparse no training: Training-free fine-tuning for sparse llms. *arXiv preprint arXiv:2310.08915*, 2023a.

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.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.