MAT: Unlocking KV Cache Efficiency via Managing Anchor Tokens

Anonymous ACL submission

Abstract

Large language models (LLMs) are powerful but require massive memory to cache the key/value vectors (KV cache) for efficient inference. To reduce the memory burden, we propose MAT, a novel KV cache eviction strategy tailored to heterogeneous attention patterns observed in shallow and deep layers of LLMs. Through a detailed analysis of attention patterns in LLMs, we observe that, for deeper layers, the anchor tokens, which consistently receive high attention logits from subsequent tokens, exhibit notably low attention logits between one another. These observations motivate us to prioritize retaining anchor tokens based on their attention logits to the first token for deep layers. For shallow layers, we retain the first few tokens of inputs as well as a sliding window to preserve local context. Extensive experiments conducted on end-to-end, language modeling, and open-ended generation tasks demonstrate that MAT achieves superior performance compared with existing methods when using the same memory budgets.

1 Introduction

Large Language Models (LLMs) have achieved great success in various domains, e.g., text generation, machine translation, and question answering (Touvron et al., 2023a,b; Achiam et al., 2023). Despite their effectiveness, they require substantial computational resources to recompute the key and value vectors (KV) for all previous tokens at every generation step. To avoid redundant computation, KV cache technique (Pope et al., 2023) stores previous key/value vectors in the memory and thus speeds up the decoding efficiency. However, the additional memory required by KV cache grows linearly with the length of the input prompt (Fu, 2024), causing a heavy burden for memory-sensitive devices like smartphones or laptops.

To reduce memory usage during generation progress, many KV cache compression methods, including KV cache quantization (Sheng et al., 2023; Liu et al., 2024b; Hooper et al., 2024) and KV cache eviction (Xiao et al., 2024; Zhang et al., 2023; Han et al., 2023; Cai et al., 2024), have been proposed. Cache quantization methods reduce the bit-width used for each key and value stored in the memory, but still require a large cache size. This work focuses on the KV cache eviction technique.

Cache eviction methods retain important tokens and evict redundant tokens in the decoding process to reduce the memory footprint of storing key/value vectors. For example, the recent state-of-the-art StreamingLLM (Xiao et al., 2024), identifies the *first* few tokens as important tokens (called sink tokens), which, together with the tokens in the sliding window, are retained in the cache. However, *intermediate* tokens can also be important in the generation progress (Sun et al., 2024a), evicting them leads to performance degradation in StreamingLLM. A crucial question in KV cache eviction is how to identify the important intermediate tokens to be retained.

We start with a property (Xiao et al., 2024; Zhang et al., 2023) in various LLMs where attention is consistently concentrated on a few tokens for deeper layers. These tokens, called **anchor tokens** in this work, consistently receive significant attention scores from the subsequent tokens. Building on this, we further analyze the attention patterns and reach two key observations.

(i) *Attention logits between anchor tokens are low*. We empirically find that the attention logits from anchor tokens to other anchor tokens are significantly lower than the attention logits from normal tokens to anchor tokens. This property is useful for seeking intermediate anchor tokens based on existing anchor tokens.

(ii) *Attention shows diffused patterns for shallow layers*. The attention logits in shallow layers are relatively sparse and diffused; that is, no few tokens dominate the attention scores. This observation

aligns with recent findings (Chen et al., 2024) that LLMs need to gather diverse information in shallow layers.

Using the above observations, we propose a novel method called MAT to unlock the KV cache efficiency via Managing Anchor Tokens. We divide the KV cache into two parts: an ***anchor cache*** to store anchor tokens and a window cache to store tokens within the sliding window. For deep layers, to identify the anchor tokens, based on the above Observation (i), we select the tokens with low attention logits to the first token (which is definitely an anchor token). For shallow layers, since no intermediate token has a dominant influence on the layer's output, the anchor cache stores the first few tokens of inputs.

Extensive experiments conducted on various tasks, including end-to-end generation, language modeling, and open-ended generation, demonstrate that MAT consistently achieves state-of-the-art performance for different models. Specifically, MAT outperforms StreamingLLM (Xiao et al., 2024) by a large margin of 1.93% on the average accuracy of eight zero-shot tasks, while maintaining the same throughput and retaining only 30% of the cache in LLaMA-2 (7B) (Touvron et al., 2023b). Moreover, MAT performs comparably to LLM with full cache within less than 70% cache budgets, validating the effectiveness of caching anchor tokens.

Our contributions are summarized as follows:

- We empirically delve into the attention patterns of LLMs, revealing an important property that the attention logits between anchor tokens are relatively smaller compared with those between anchor tokens and normal tokens. To the best of our knowledge, we are the first to observe this property and leverage it to improve the KV cache efficiency.
- Building on this property, we propose a novel method called MAT to manage the KV cache by seeking and retaining anchor tokens while evicting redundant tokens.
- Extensive experimental results on various tasks for different models demonstrate that MAT is effective in preserving performance while reducing memory usage of KV cache.

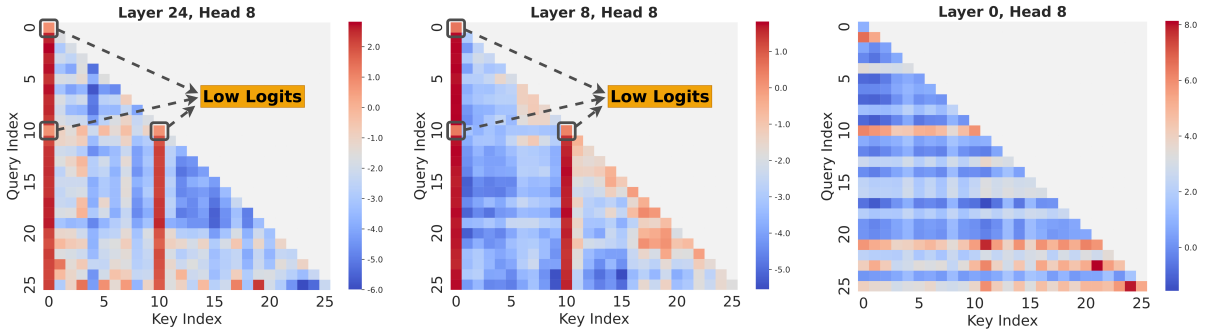
2 Related Work

Interpretation of Attention Patterns. LLMs have achieved great success in natural language processing based on attention mechanisms (Vaswani, 2017). Many prior works (Chen et al., 2025; Xiao et al., 2024; Zhang et al., 2023) have shown that attention matrices in LLMs are typically sparse, focusing disproportionately on a few special tokens. For instance, Xiao et al. (2024) and Zhao et al. (2023) show that LLMs assign most of the attention to the starting word token. Some works (Clark, 2019; Kovaleva, 2019; Bondarenko et al., 2021) discover that attention in BERT (Devlin, 2018) tends to focus on "separate" tokens. Ge et al. (2023) finds that high attention scores are usually allocated towards a set of fixed tokens. To explore the reason behind sparsity attention, Sun et al. (2024a) identifies the "massive activation" phenomenon to provide an in-depth analysis. Different from these existing findings, we concentrate on the relationships between tokens that are consistent with massive attention logit values.

KV Cache Compression. Many recent efforts have been devoted to reducing the running-time memory of LLM by compressing the KV cache. They can mainly be classified into two categories: (i) Reducing memory requirement for storing each KV embedding and (ii) Introducing sparsity to discard redundant tokens during the decoding process.

To reduce the memory footprint of KV embeddings, Cache Quantization (Sheng et al., 2023; Liu et al., 2024b; Hooper et al., 2024) converts higher-precision representations (such as 32-bit floating-point) of cache to lower-precision ones (such as 8-bit integers). For example, FlexGen (Sheng et al., 2023) and KIVI (Liu et al., 2024b) propose to quantize keys per channel and values per token. Moreover, Chang et al. (2024) reduces inference-time memory usage by low-rank techniques.

For the second category, recent works like StreamingLLM (Xiao et al., 2024) and LM-Infinite (Han et al., 2023) propose to retain initial tokens as "sink tokens", suggesting the removal of KV caches for intermediate tokens. H2O (Zhang et al., 2023) and Scissorhands (Liu et al., 2024a) identify the important tokens based on high passed attention score and remove the lowest one for each step. In addition to selecting which tokens to be evicted, PyramidKV (Cai et al., 2024) allocates different KV cache budgets across different layers. CaM (Zhang et al., 2024b) merges the to-be-evicted caches into the remaining ones. Our proposed MAT falls into the second category. However, unlike existing approaches, we introduce a novel method for evicting tokens based on the behavior of anchor tokens.



(a) Layer 24. (b) Layer 8. (c) Layer 0.

Figure 1: Attention patterns for different layers in LLaMA-2(7B).

3 Observations

To gain a deeper understanding of the attention mechanism in large language models (LLMs), we study the attention patterns they exhibit. Figure 1 visualizes the attention logits (i.e., the raw scores before the application of softmax) of LLaMA2-7B for an illustrative input example from the ARC-Challenge dataset (Clark et al., 2018): "Question: Plants and animals need food for growth. What happens to most of the food that plants produce?\nAnswer:". Due to the page limit, additional visualizations of attention logits for diverse inputs are provided in Appendix A.

Figure 1(b) and Figure 1(a) show the attention logits in two deeper layers, i.e., layer 8 and layer 24, respectively. As shown, we can observe a similar phenomenon as mentioned in (Zhang et al., 2023; Xiao et al., 2024) that most queries have a much higher attention logit on a small set of specific tokens than on other tokens. For example, 0-th and 10-th tokens consistently attract significant attention logits for all subsequent queries. We define the type of tokens that consistently receive massive attention logits from the subsequent tokens as ***anchor tokens***. Figure 2 compares the testing accuracy between LLaMA2-7B and versions without random tokens and anchor tokens. We can see that the performance significantly drops when anchor tokens are removed, demonstrating that these tokens play an important role in shaping the generation of the final output. Moreover, Figure 7 in Appendix A shows the attention patterns of different prompts, we find that the first token is consistently an anchor token. Then, we introduce two interesting observations for these anchor tokens.

3.1 Attention Logits Between Anchor Tokens are Low

We observe that the attention logits from anchor tokens to other anchor tokens are significantly lower

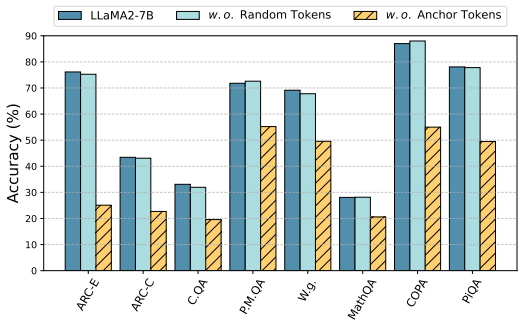


Figure 2: Performance comparison between the LLaMA2-7B model with full KV cache, the model *w.o.* random tokens and the model *w.o.* anchor tokens.

Table 1: Average attention logits between anchor tokens and other tokens. Rows correspond to queries, while columns correspond to keys. The values are average attention logits from the query (row) to the key (column).

	Anchor token 1	Anchor token 2
Anchor token 1	0.4266	-
Anchor token 2	1.3677	0.0792
Other Tokens	2.3608	1.2246

than the attention logits from normal tokens to anchor tokens. For example, consider an anchor token (column 0) in Figures 1(b) and 1(a). The attention logits from query 0-th and 10-th tokens are consistently lower than those from other tokens across different layers. To further validate this observation, for each sample in ARC-C dataset, we extract two anchor tokens sorted by their indices. Then, we compute two types of attention logits including (i) anchor tokens to anchor tokens, and (ii) other tokens to anchor tokens. These attention logits are averaged for all the samples and reported in Table 1. As shown, the attention logits from anchor tokens (first two rows) are significantly lower than those from other tokens (last row).

Our observation reveals the relationship between anchor tokens and provides a potential to find other anchor tokens based on the existing anchor tokens. This observation will motivate us to design a KV cache management strategy in Section 4.

3.2 Attention Shows Diffused Patterns for Shallow Layers

While anchor tokens play a crucial role in generation within deep layers, we observe that they are less influential in shallow layers. Figure. 1(c) shows the attention patterns for different heads in layer 0. Additional experiments for the other shallow layer (i.e. layer 1) can be found in Figure 8 in the Appendix. As can be seen, the attention logits appear relatively sparse and diffused, with no clear dominant focus. This indicates that the model broadly attends to a wide range of tokens without strong preferences. Interestingly, different heads demonstrate subtle but distinct patterns. For instance, head 8 displays weak diagonal structures, indicating a slight preference for attending to nearby tokens, which reflects a focus on local context. In contrast, head 16 exhibits a more distributed attention pattern, with attention spread across a wider range of tokens. This may indicate an early attempt to capture broader semantic relationships within the input. Overall, the diffuse attention patterns observed in layer 0 align with the model's need to gather diverse contextual information during the initial stages of processing (Sun et al., 2024a; Cai et al., 2024).

These attributes are also supported by other models (i.e., LLaMA-1 (Touvron et al., 2023a), Mistral (Jiang et al., 2023)), the details can be found in the Appendix A.

4 Methodology

Building on the insights discussed above, we propose **MAT**, a novel method to unlock KV cache efficiency via Managing Anchor Tokens. An overview is illustrated in Figure 3. We formally introduce the problem of KV cache eviction in Section 4.1 and present MAT in Section 4.2.

4.1 Problem Formulation

For simplicity of presentation, we focus on one attention head with weight matrices $\mathbf{W}_Q \in \mathbb{R}^{d_i \times d}$, $\mathbf{W}_K \in \mathbb{R}^{d_i \times d}$ and $\mathbf{W}_V \in \mathbb{R}^{d_i \times d}$, where d_i and d are the input and hidden dimension, respectively. At each decoding step t, the LLM generates the next token based on the current KV cache stored in the memory. Specifically, an input $\mathbf{x}_t \in \mathbb{R}^{d_i}$ of the attention module is mapped into query, key, and value vectors respectively by:

$$\mathbf{q}_t = \mathbf{W}_Q^\top \mathbf{x}_t, \mathbf{k}_t = \mathbf{W}_K^\top \mathbf{x}_t, \mathbf{v}_t = \mathbf{W}_V^\top \mathbf{x}_t. \quad (1)$$

Let $\mathbf{K} \in \mathbb{R}^{d \times n}$ and $\mathbf{V} \in \mathbb{R}^{d \times n}$ be the current cached key and value matrices, where n is the cache size. During inference, the cache size may exceed the memory limit c either in the pre-filling stage or during autoregressive generation. To tackle this problem, many KV cache methods (Xiao et al., 2024; Zhang et al., 2023; Liu et al., 2024a) have been proposed to update the KV cache by inserting the latest token and evicting redundantly cached tokens, whose update rule generally follows:

$$\mathbf{K} \leftarrow \text{update}(\mathbf{K}, \mathbf{k}_t), \mathbf{V} \leftarrow \text{update}(\mathbf{V}, \mathbf{v}_t). \quad (2)$$

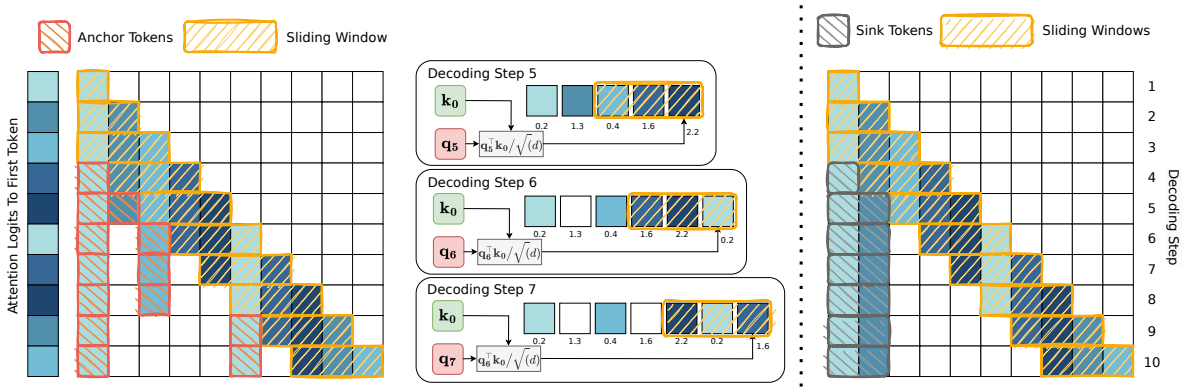
Evicting redundant tokens ensures that the key and value caches remain within the memory limit. Based on the updated KV cache, the attention score $\mathbf{a}_t \in \mathbb{R}^n$ between the query vector $\mathbf{q}_t$ and the key matrix $\mathbf{K}$ is computed as $\mathbf{a}_t = \text{softmax}\left(\frac{\mathbf{q}_t^\top \mathbf{K}}{\sqrt{d}}\right)$, while the corresponding attention output $\mathbf{o}_t = \mathbf{a}_t^\top \mathbf{V} \in \mathbb{R}^d$. The output $\mathbf{o}_t$ is expected to approximate the output when using the full KV cache. Hence, the keys and values of tokens that contribute to higher attention scores for future tokens should be retained, while those with lower attention scores should be evicted. Since future tokens are inherently unavailable when performing KV cache at step t, determining which tokens are irrelevant for future generations remains a challenging problem.

4.2 Managing Anchor Tokens

In this section, we propose a novel method called **MAT** to unlock KV cache efficiency via Managing Anchor Tokens. The goal is to manage the KV cache by identifying and retaining tokens that significantly influence the model's output during the generation process while evicting redundant ones.

In our proposed MAT, the KV cache consists of two parts: an ***anchor cache*** contains β anchor tokens which receive massive attention logits from the subsequent tokens, and a window cache that contains the tokens within the sliding window. As discussed in Section 3, the performance of LLM drops significantly when removing the anchor tokens. Hence, retaining these tokens in the cache is crucial for preserving the original capabilities of LLMs. As the first token of any input always receives high attention logits from all the following tokens, it is an anchor token and is always kept in the anchor cache.

To seek other anchor tokens in the cache, we use the property discovered in Section 3.1: *attention logits between anchor tokens are particularly low*.



(a) Update for Deep Layers (b) Update for Shallow Layers

Figure 3: Illustration of the proposed MAT method. (a) For deep layers, we retain the tokens with Bottom-β attention logits (here, $\beta = 2$) to the first token as anchor tokens, and a sliding window keeps the last $c - \beta$ token in the cache. (b) For shallow layers, the KV cache includes the initial s (here, $s = 2$) tokens as sink tokens and a sliding window with size $c - s$ (here, $c = 5$).

Let $\hat{\mathbf{q}}_i$ be the query value of the i-th token in the cache and $\pi_i = \frac{\hat{\mathbf{q}}_i^\top \mathbf{k}_0}{\sqrt{d}}$ be the attention logit between the i-th token and the first token. We define an attention logit cache $\mathcal{A}$ as:

$$\mathcal{A} = \{\pi_i : i = 1, \ldots, n\}. \tag{3}$$

Since the first token is an anchor token and attention logits between anchor tokens are relatively lower, tokens in the cache with lower attention logits to the first token (i.e., lower value in $\mathcal{A}$) are more likely to be anchor tokens. In other words, tokens with higher attention logits to the first token are less important. Hence, we update the KV cache by selecting and evicting the token with the highest value in $\mathcal{A}$ when the cache exceeds memory limit:

$$r = \arg\max_{1 \le i \le \beta} \pi_i, \quad \mathcal{A} \leftarrow \mathcal{A}_{-r} \cup \left\{ \mathbf{q}_t^\top \mathbf{k}_0 / \sqrt{d} \right\},$$
$$\mathbf{K} \leftarrow [\mathbf{K}_{:,-r}, \mathbf{k}_t], \quad \mathbf{V} \leftarrow [\mathbf{V}_{:,-r}, \mathbf{v}_t], \tag{4}$$

where $\mathcal{A}_{-r}$ means removing the r-th element from $\mathcal{A}$, and $\mathbf{K}_{-r}$ (resp. $\mathbf{V}_{-r}$) means the matrix $\mathbf{K}$ (resp. $\mathbf{V}$) with the i-th column vector removed. The r-th token has the highest attention logit to $\mathbf{k}_0$, making it less likely to serve as an anchor token. Hence, removing this token has a minor impact on overall performance. As the first token is an anchor token that exhibits low attention logits to itself, it is always retained. Due to the strong local dependencies and interactions of language tasks, tokens within the sliding window are always retained in the cache. Selecting the r-th tokens is computationally efficient as storing the attention logits from cached tokens to the first token is very cheap.

According to Observation (ii) in Section 3.2, attention logits in the shallow layers of LLMs tend to be relatively sparse and diffused. No single token consistently exhibits large attention logits across all subsequent tokens, meaning no token has a dominant influence on the layer's output. Hence, intermediate anchor tokens are unavailable for the shallow layers. Following StreamingLLM (Xiao et al., 2024), we retain the first s tokens of inputs in the cache, as these initial tokens remain accessible to all subsequent tokens and thus are important. When exceeding the memory limit, we update the KV cache for the shallow layers by moving the sliding window:

$$\mathbf{K} \leftarrow [\mathbf{K}_{:,-s}, \mathbf{k}_t], \quad \mathbf{V} \leftarrow [\mathbf{V}_{:,-s}, \mathbf{v}_t], \tag{5}$$

When the KV cache has not exceeded the memory limits (i.e., $n \le c$), we add the current key and value vector to the cache as: $\mathbf{K} \leftarrow [\mathbf{K}, \mathbf{k}_t]$, $\mathbf{V} \leftarrow$

Algorithm 1 MAT.

Require: KV cache $\mathbf{K}$, $\mathbf{V}$, #shallow layers L_s, attention logits cache $\mathcal{A}$, attention weight matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$, anchor cache size β, sink cache size s, input prompt $\mathbf{x}_t$;

compute the key vector of the first token $\mathbf{k}_0$;
$\mathbf{q}_t = \mathbf{W}_Q^\top \mathbf{x}_t, \mathbf{k}_t = \mathbf{W}_K^\top \mathbf{x}_t, \mathbf{v}_t = \mathbf{W}_V^\top \mathbf{x}_t$;
$\pi_r = \frac{\mathbf{q}_t^\top \mathbf{k}_0}{\sqrt{d}}$, $\mathcal{A} \leftarrow \mathcal{A} \cup \{\pi_t\}$;
if $t < c$ **then**
  $\mathbf{K} \leftarrow [\mathbf{K}, \mathbf{k}_t], \mathbf{V} \leftarrow [\mathbf{V}, \mathbf{v}_t]$;
else if l is a deep layer (i.e., $l > L_s$) **then**
  select index $r = \arg\max_{1 \le i \le \beta} \pi_i$;
  $\mathbf{K} \leftarrow [\mathbf{K}_{:,-r}, \mathbf{k}_t], \mathbf{V} \leftarrow [\mathbf{V}_{:,-r}, \mathbf{v}_t], \mathcal{A} \leftarrow \mathcal{A}_{-r}$;
else if l is a shallow layer (i.e., $l \le L_s$) **then**
  $\mathbf{K} \leftarrow [\mathbf{K}_{:,-s}, \mathbf{k}_t], \mathbf{V} \leftarrow [\mathbf{V}_{:,-s}, \mathbf{v}_t], \mathcal{A} \leftarrow \mathcal{A}_{-s}$.
end if

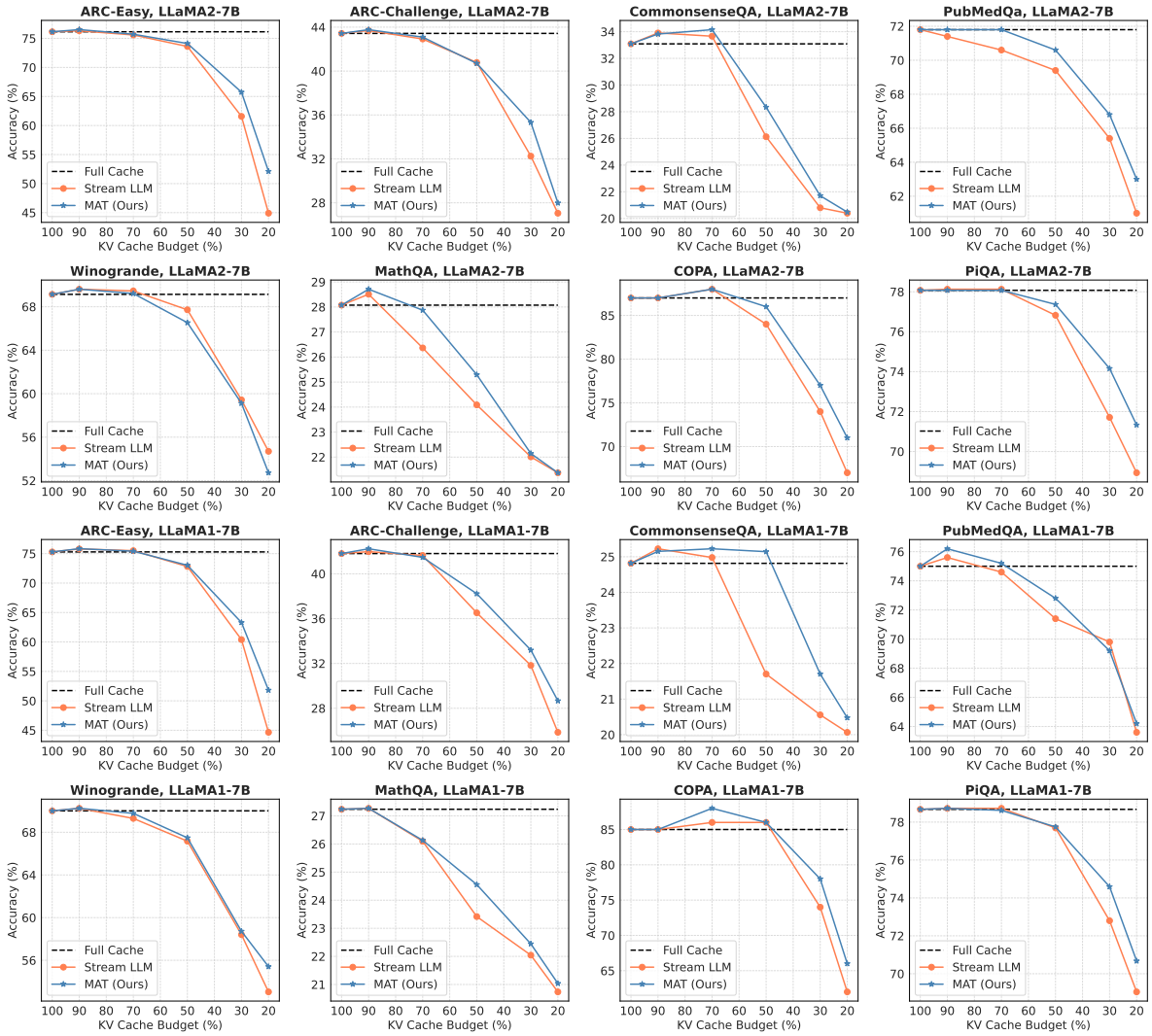


Figure 4: Performance comparison between full KV cache memory, Streaming LLM, and our proposed MAT for compressing KV cache of mainstream LLMs on multiple language tasks.

$[\mathbf{V}, \mathbf{v}_t]$, The overall update pipeline of MAT at step t for layer l is provided in Algorithm 1.

5 Experiments

5.1 End-to-End Tasks

Setup. Following (Zhang et al., 2023, 2024b), we evaluate MAT on the LLaMA family of models, i.e., LLaMA-1 (7B) (Touvron et al., 2023a) and LLaMA-2 (7B) (Touvron et al., 2023b). The evaluation is conducted on eight end-to-end tasks from the popular evaluation framework lm-eval-harness (Gao et al., 2024), including: ARC-Easy/Challenge (Clark et al., 2018), CommensenseQA (Talmor et al., 2019), PubMedQA (Jin et al., 2019), Winogrande (Sakaguchi et al., 2021), MathQA (Amini et al., 2019), COPA (Roemmele et al., 2011), PiQA (Bisk et al., 2020). We compare MAT with (i) Full Cache, which has no limit on cache size, (ii) StreamingLLM (Xiao et al., 2024), which caches the sink tokens together with the tokens within the sliding window. For a specific KV cache budget of c, we use a grid search to determine the optimal size β of the anchor cache, and the size of the sliding window is $c-\beta$. All experiments are conducted on an NVIDIA DGX A100 (80G) GPUs.

Results. Figure 4 shows the testing accuracy when compressing KV cache memory at ratios ranging from 20% to 100%. As can be seen, MAT consistently achieves higher accuracies than StreamingLLM for different tasks, models, and KV cache budgets. Moreover, with less than 70% KV cache budget, MAT performs comparably to the full KV cache. With 90% KV cache budget, MAT can outperform full KV cache for all the tasks, demonstrating that removing some redundant tokens can improve the model generalization ability.

Table 2 shows the testing accuracy for MAT and StreamingLLM under a constrained 30% KV cache

Table 2: Accuracy (%) on eight zero-shot tasks under 30% KV cache budget. The best is in **bold**.

Model	Method	ARC-E	ARC-C	C.QA	P.M.QA	Winogrande	MathQA	COPA	PiQA	AVG
LLaMA-1 (7B)	StreamingLLM	60.40	31.83	20.56	**69.80**	58.41	22.04	74.00	72.80	51.23
	MAT	**63.30**	**33.19**	**21.70**	69.00	**58.72**	**22.45**	**78.00**	**74.59**	**52.62**
LLaMA-2 (7B)	StreamingLLM	61.57	32.25	20.80	65.40	59.12	22.01	74.00	71.70	50.86
	MAT	**65.74**	**35.32**	**21.70**	**66.80**	**59.43**	**22.14**	**77.00**	**74.16**	**52.79**

budget. As can be seen, MAT consistently outperforms StreamingLLM across nearly all evaluated zero-shot tasks for both LLaMA-1 (7B) and LLaMA-2 (7B) models. For LLaMA-1 (7B), MAT achieves superior performance on 7 out of 8 tasks, with only PubMedQA showing a slight decrease (-0.8%) compared to StreamingLLM. Overall, MAT improves the average accuracy by 1.39% (from 51.23% to 52.62%). For LLaMA-2 (7B), the performance gains are even more substantial, with MAT outperforming StreamingLLM on all eight tasks and achieving an average improvement of 1.93% (from 50.86% to 52.79%). The most significant improvements are observed on ARC-Easy (+4.17%) and ARC-Challenge (+3.07%) tasks, suggesting that MAT's anchor token management strategy is especially effective for complex reasoning tasks. These consistent improvements across different model architectures and diverse task types demonstrate the effectiveness of MAT's approach to KV cache management, which intelligently prioritizes anchor tokens based on their attention patterns.

5.2 Language Modeling Task

Setup. Following (Zhang et al., 2023), we conduct language modeling experiments on WikiText-2 (Merity et al., 2016) to evaluate the perplexity performance of MAT. Since the pre-training window size for LLaMA-2 models is 4096, we report the average perplexity for both in-window sequences (input sequence length equals 4096) and out-of-window sequences (input sequence length equals 10,000). To handle infinite-length inputs, following StreamingLLM (Xiao et al., 2024), we adopt position rolling for the updated KV cache. MAT is compared with Full Cache and StreamingLLM under cache limit of 256 and 512.

Results. Table 3 shows the average perplexity for LLaMA-2 (7B) model. As can be seen, MAT consistently outperforms StreamingLLM in both in-window and out-of-window settings. For example, when $c = 256$, MAT achieves a perplexity improvement of 0.35 and 0.08 for input sequence lengths of 4096 and 10000, respectively. Note that the Full Cache performs poorly in the out-of-window setting due to the constraint of the pretraining window size. Compared with Full Cache, MAT significantly reduces the perplexity by over 97%, demonstrating its effectiveness in handling infinite-length inputs.

Table 3: Average perplexity on the WikiText-2 dataset for LLaMA-2(7B) under in-window (PPL-4096) and out-of-window (PPL-10000) settings.

Cache Size (c)	Method	PPL-4096	PPL-10000
-	Full Cache	5.05	230.76
256	StreamingLLM	6.60	6.33
	MAT	**6.25**	**6.25**
512	StreamingLLM	5.89	5.64
	MAT	**5.74**	**5.53**

5.3 Open-Ended Generation Tasks

Setup. We evaluate MAT on open-ended generation tasks using the popular dataset GSM8K (Cobbe et al., 2021), which is a mathematical benchmark with diverse grade school math word problems. Following chain-of-thought prompting (Wei et al., 2022), we use the 8-shot prompting to guide the LLM to generate intermediate natural language reasoning steps that lead to the final answer.

Results. Table 4 shows the testing accuracy and runtime KV cache usage for the GSM8K task under different configurations. As shown, MAT consistently outperforms StreamingLLM across all settings, achieving higher accuracy while maintaining comparable running time KV caches. For example, under $c = 256$ with LLaMA-2(7B), MAT improves accuracy by a margin of 0.91, while keeping roughly the same KV cache usage. These results further demonstrate the effectiveness of MAT in tackling complex generation tasks that demand multi-step logical reasoning.

5.4 Ablation Study

We conduct experiments on language modeling tasks to compare our MAT with two possible variants: (i) MAT (with sink), which applies the update

Table 4: Accuracy and average runtime KV cache usage on GSM8K. KV (%) here represents the ratio of KV cache usage at runtime compared with the full generated sequence length. The best is in **bold**.

	Method	LLaMA-2 (7B)		LLaMA-1(7B)	
		Acc(%)	KV(%)	Acc(%)	KV(%)
	Full Cache	14.56	100	9.48	100
c=256	StreamingLLM	12.74	27.07	2.72	27.23
	MAT	**13.65**	27.12	**4.16**	26.87
c=384	StreamingLLM	13.94	40.97	5.00	40.85
	MAT	**15.92**	40.90	**5.84**	40.56

Table 5: Comparison between MAT and two variants on the perplexity of WikiText-2.

Cache Size (c)	Method	PPL-4096	PPL-10000
-	Full Cache	5.05	230.7
256	MAT (*w.* sink)	6.60	6.33
	MAT (*w.* anchor)	6.27	6.32
	MAT	**6.25**	**6.25**
512	MAT (*w.* sink)	5.89	5.64
	MAT (*w.* anchor)	**5.74**	5.54
	MAT	**5.74**	**5.53**

rule Eq. (5) for both shallow and deep layers. In this case, MAT degenerates into StreamingLLM. (ii) MAT (with anchor), which applies the update rule Eq. (4) to both shallow and deep layers. Table 5 shows the perplexity. We can see that using the default strategy (i.e., MAT) achieves the lowest perplexity across different cache sizes. Moreover, MAT (with anchor) is consistently better than MAT (with sink), showing that retaining anchor tokens is more effective than retaining sink tokens used in StreamingLLM.

Effects of β. To study the effect of the size of the anchor cache (i.e., β), we evaluate MAT on the language modeling task using various values of β, while keeping the total cache limit fixed at $c = 256$. Figure 5 shows the average perplexity on the WikiText-2 dataset. As can be seen, MAT is insensitive to a wide range of $\beta \in [32, 96]$, achieving an improvement of approximately 0.3 compared to StreamingLLM. Moreover, increasing β reduces the average perplexity when β is small. However, excessively large values of β lead to a decline in performance.

5.5 Efficiency Analysis

We evaluate the efficiency of MAT by measuring generation throughput and peak GPU memory usage on an NVIDIA A100 (80GB) GPU. All input sequences used for evaluation were sampled from WikiText-2 and had a length of 50,000 tokens.

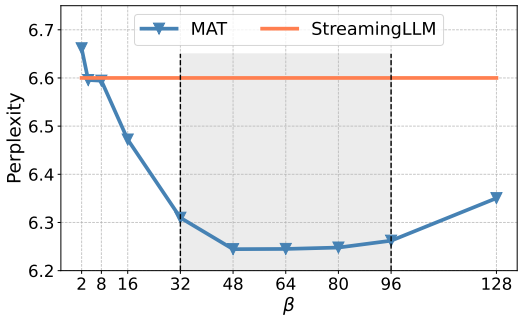


Figure 5: Average perplexity w.r.t. β on the language modeling task.

Table 6: Average generation throughput (tokens per second) and GPU memory consumption on the WikiText-2 dataset for LLaMA-2(7B).

Cache Size (c)	Method	Throughput	GPU memory
-	Full Cache	22.68 tokens/s	56.55 GB
256	StreamingLLM	29.05 tokens/s	25.96 GB
	MAT	29.21 tokens/s	25.96 GB
512	StreamingLLM	28.62 tokens/s	26.35 GB
	MAT	28.61 tokens/s	26.35 GB

As shown in Table 6, MAT demonstrates significant efficiency gains compared to the standard Full Cache baseline, particularly for processing long sequences. For instance, compared to Full Cache (22.68 tokens/s, 56.55 GB), MAT with a cache size (c) of 256 achieves a higher throughput of 29.21 tokens/s (a 1.29× speedup) while drastically reducing peak GPU memory usage by 54% to 25.96 GB. Furthermore, MAT exhibits throughput and memory consumption comparable to StreamingLLM but performs significantly better than StreamingLLM (as shown in Tables 2 and 4), due to its efficient reuse of attention scores computed during generation for its token selection metric.

6 Conclusion

In this paper, we propose a novel approach to enhance the performance of LLMs under memory constraints through KV cache eviction. Through a detailed analysis of attention patterns in LLMs, we observe the anchor tokens, which attract high attention logits from subsequent tokens, exhibit particularly low attention logits between themselves. Inspired by these observations, we propose MAT (unlocking KV cache Efficiency via **M**anaging **A**nchor **T**okens) to recognize and retain the important anchor tokens in the cache based on their attention logits to the first anchor token during the generation process. Experimental results across a variety of tasks show that MAT outperforms existing methods and preserves the original capability of LLM.

7 Limitations

Due to the limited computational resources, we only evaluate MAT using LLMs with a relatively small number of parameters, such as the 7B models from the LLaMA family. However, there are many publicly available LLMs with more parameters and greater capabilities (e.g., LLaMA-2(70B). Thus, it is reasonable to apply MAT to these more advanced, albeit costly, models. We leave the investigation of such scenarios to future work.

8 Ethical Consideration

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

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. Preprint arXiv:2303.08774.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and overcoming the challenges of efficient transformer quantization. Preprint arXiv:2109.12948.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and 1 others. 2024. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. Preprint arXiv:2406.02069.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. 2024. Palu: Compressing KV-cache with low-rank projection. Preprint arXiv:2407.21118.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. 2025. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*. Springer.

Tianxiang Chen, Zhentao Tan, Tao Gong, Yue Wu, Qi Chu, Bin Liu, Jieping Ye, and Nenghai Yu. 2024. Llama SLayer 8B: Shallow layers hold the key to knowledge injection. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5991–6002, Miami, Florida, USA. Association for Computational Linguistics.

Kevin Clark. 2019. What does BERT look at? an analysis of bert's attention. Preprint arXiv:1906.04341.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. Preprint arXiv:1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Hesse Christopher, and Schulman John. 2021. Training verifiers to solve math word problems. Preprint arXiv:2110.14168.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.Int8 (): 8-bit matrix multiplication for transformers at scale. In *Neural Information Processing Systems*.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. Preprint arXiv:1810.04805.

Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *Neural Information Processing Systems*.

Yao Fu. 2024. Challenges in deploying long-context transformers: A theoretical peak performance analysis. Preprint arXiv:2405.08944.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation. Technical report.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive KV cache compression for llms. (arXiv preprint arXiv:2310.01801).

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. Lm-infinite: Simple on-the-fly length generalization for large language models. Preprint arXiv:2308.16137.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Nneural Information Processing Systems*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 million context length llm inference with KV cache quantization. Preprint arXiv:2401.18079.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. Preprint arXiv:2310.06825.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Olga Kovaleva. 2019. Revealing the dark secrets of bert. Preprint arXiv:1908.08593.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*.

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

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: Plug-and-play 2bit KV cache quantization with streaming asymmetric quantization. Preprint arXiv:2402.02750.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. Preprint arXiv:1609.07843.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*. PMLR.

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. 2024a. Massive activations in large language models. Preprint arXiv:2402.17762.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024b. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023b. LLaMA 2: Open foundation and fine-tuned chat models. Preprint arXiv:2307.09288.

A Vaswani. 2017. Attention is all you need. In *Neural Information Processing Systems*.

Wenxiao Wang, Wei Chen, Yicong Luo, Yongliu Long, Zhengkai Lin, Liye Zhang, Binbin Lin, Deng Cai, and Xiaofei He. 2024. Model compression and efficient inference for large language models: A survey. Preprint arXiv:2402.09748.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Neural Information Processing Systems*.

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

Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, and 1 others. 2024. A survey of resource-efficient LLM and multimodal foundation models. Preprint arXiv:2401.08092.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. 2024a. Plug-and-play: An efficient post-training pruning method for large language models. In *International Conference on Learning Representations*.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024b. Cam: Cache merging for memory-efficient llms inference. In *International Conference on Machine Learning*.

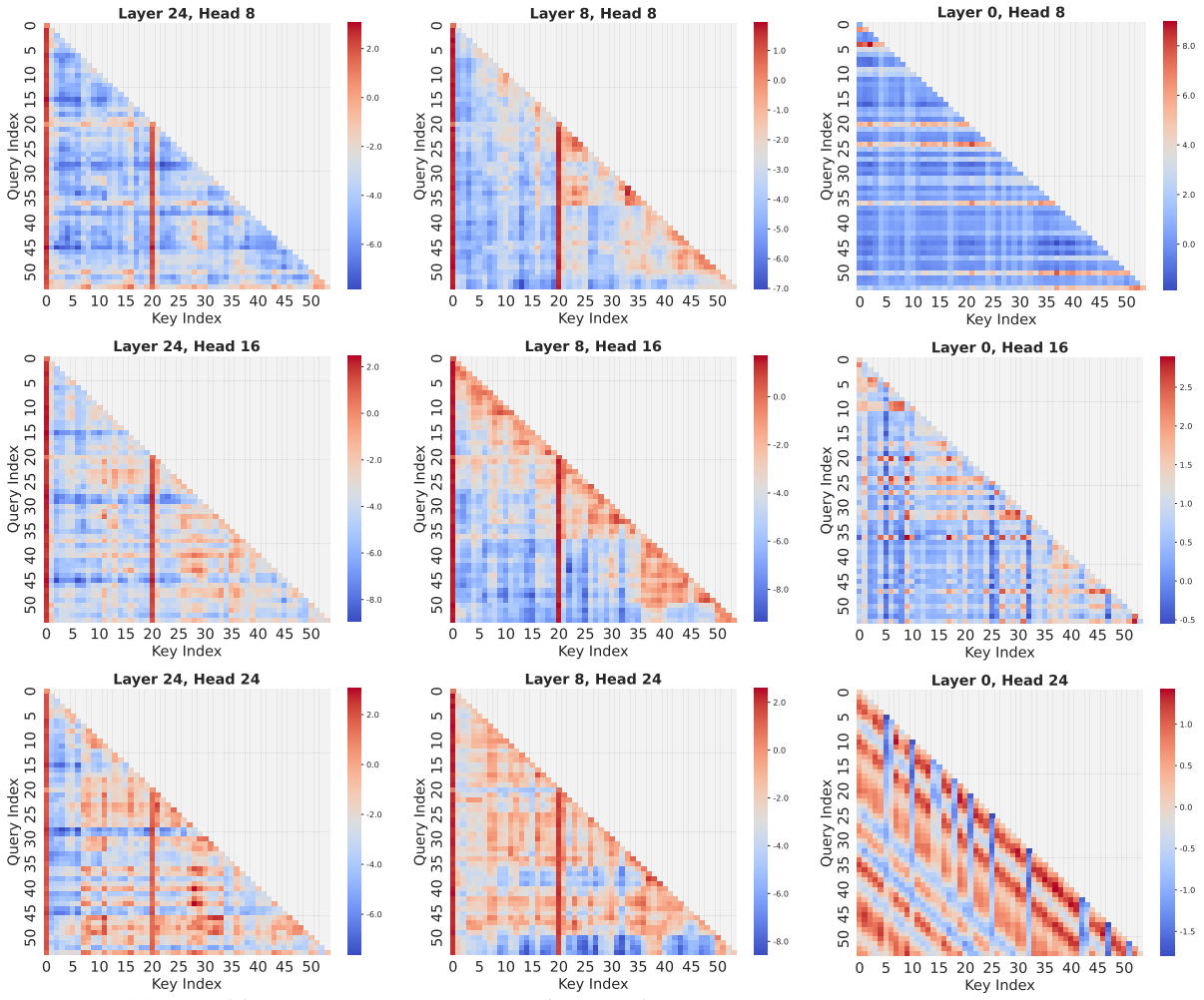
Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Neural Information Processing Systems*.

Jun Zhao, Zhihao Zhang, Yide Ma, Qi Zhang, Tao Gui, Luhui Gao, and Xuanjing Huang. 2023. Unveiling a core linguistic region in large language models. Preprint arXiv:2310.14928.

A Additional Attention Patterns

In Section 3, we have concluded three key observations by a specific prompt in LLaMA-2(7B). In this section, we study the attention patterns for diverse input prompts and models.

Figure 6 visualizes the attention patterns for a different input prompt. As shown, we can observe the notable attributes in Section 3 again. For example, a small set of tokens (the 0-th and 20-th tokens) consistently receive large attention logits from the subsequent tokens for deep layers (layer 8 and layer 24), validating the observation in Section **??**. Moreover, the attention logits from the 20-th token to the 0-th token are significantly lower than the attention tokens from the normal token to the 0-th token, validating the observation in Section 3.1.



(a) Layer 24. (b) Layer 8. (c) Layer 0.

Figure 6: Attention patterns for input prompt: “Question: One year, the oak trees in a park began producing more acorns than usual. The next year, the population of chipmunks in the park also increased. Which best explains why there were more chipmunks the next year?\nAnswers:" w.r.t. different layers and heads in LLaMA-2(7B).

Figure 7 shows the attention patterns for different input prompts on layer 24, head 24 of LLaMA-2(7B). Regardless of the prompt, the patterns consistently align with the observations detailed in Section **??** and Section 3.1. Additionally, the first token is consistently an anchor token for different prompts.

Figure 8 visualizes the attention patterns for different heads in layer 1 of LLaMA-1(7B). As can be seen, the attention patterns show distinct characteristics across the heads. Moreover, the attention in Layer 1 appears diverse, with no single token dominating the patterns across heads. These observations are consistent with the findings discussed in Section 3.2.

Figure 9 and Figure 10 shows the attention patterns on LLaMA-2(7B) and Mistral(7B), respectively.

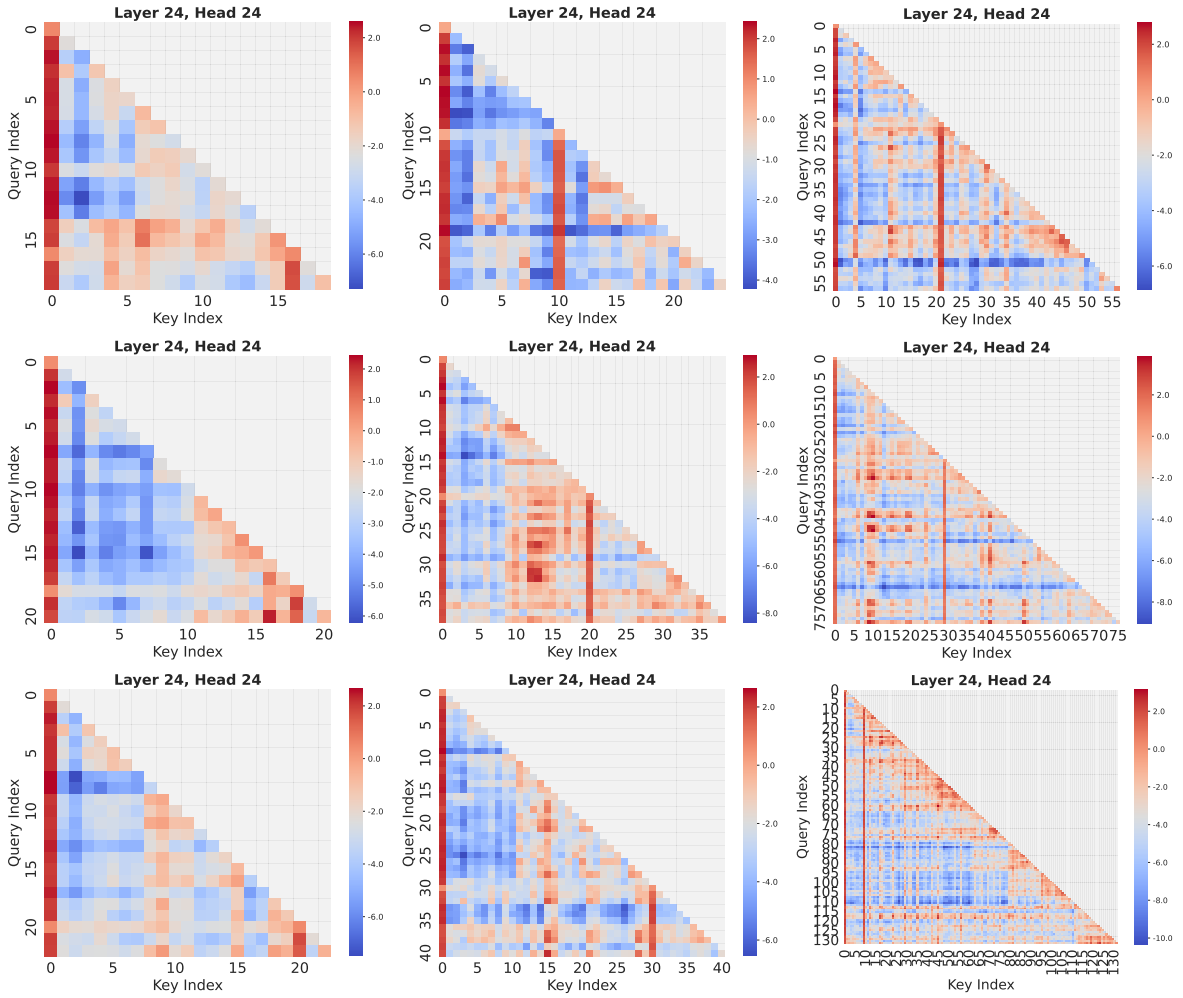


Figure 7: Attention patterns for layer 24, head 24 in LLaMA-2(7B) w.r.t. different input prompt.

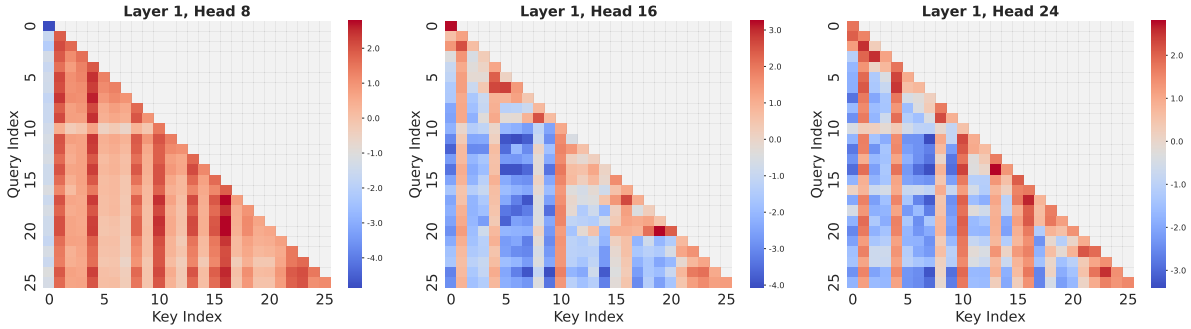
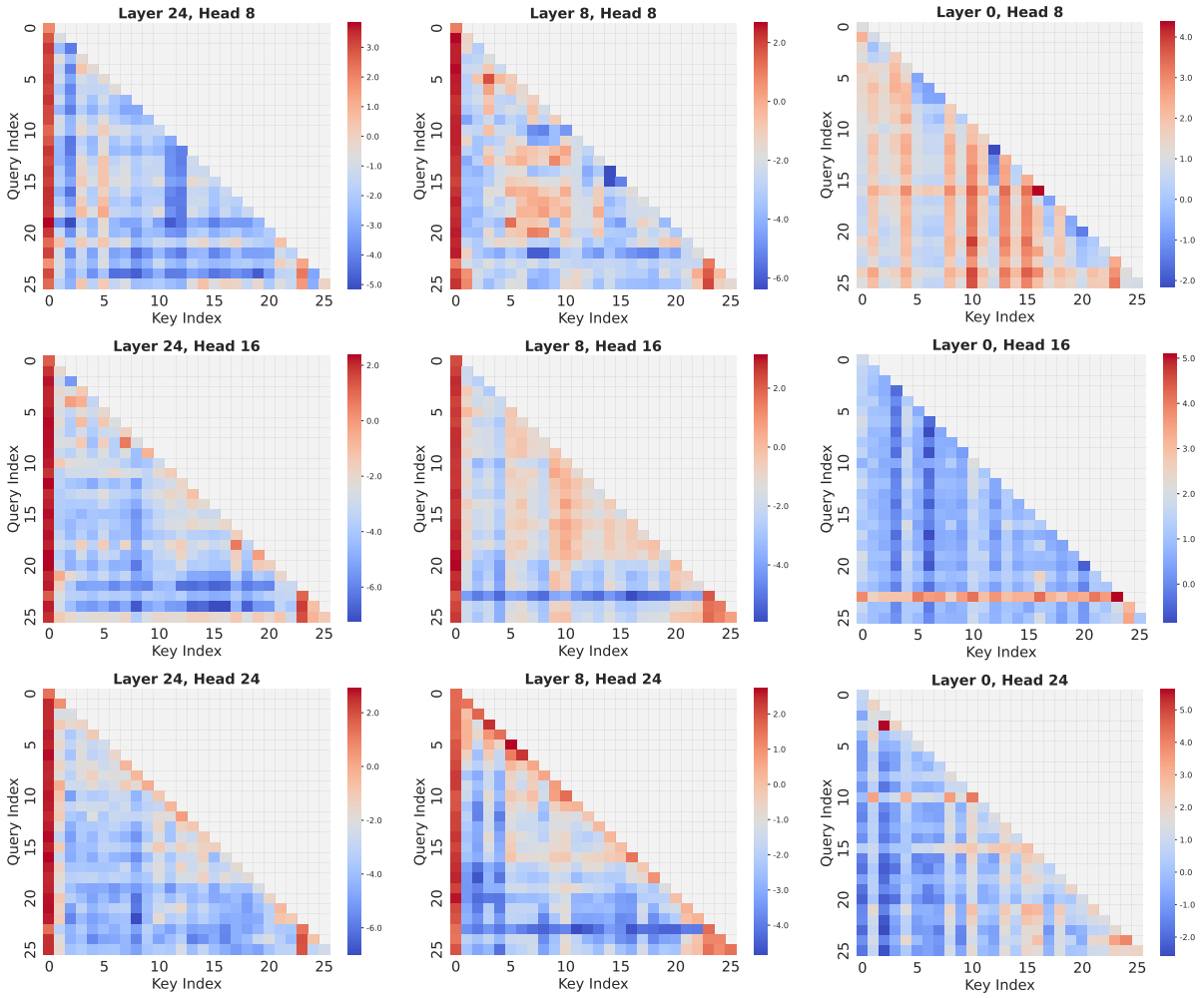


Figure 8: Attention patterns for input prompt: “Question: Plants and animals need food for growth. What happens to most of the food that plants produce?\nAnswer:" w.r.t. different heads in layer 1 of LLaMA-2(7B).

We can see that the observations in Section 3 remain consistent across different models.

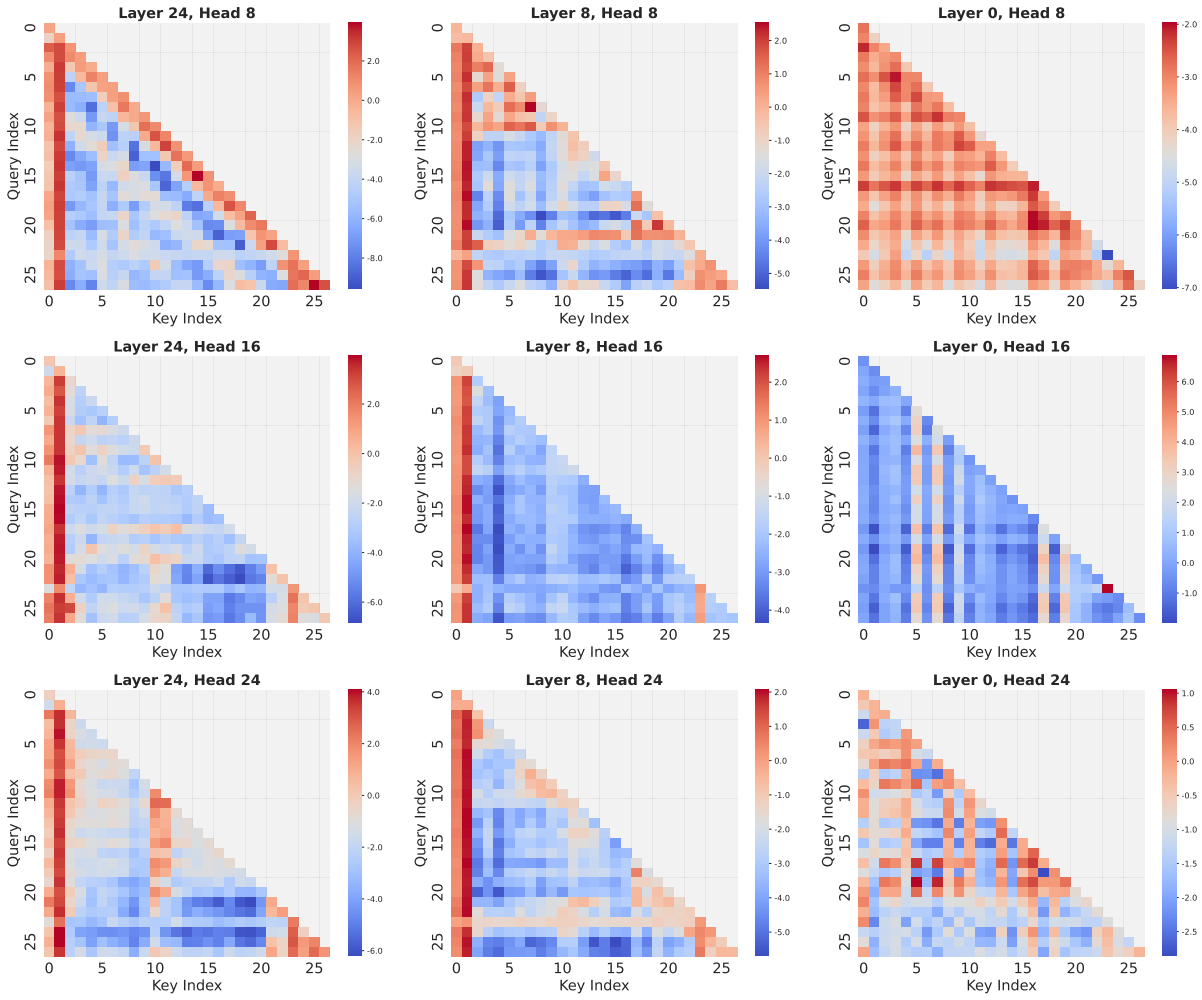


(a) Layer 24. (b) Layer 8. (c) Layer 0.

Figure 9: Attention patterns in LLaMA-1(7B) for input prompt: "Question: Plants and animals need food for growth. What happens to most of the food that plants produce?\nAnswer:" w.r.t. different layers and heads.

B Discussion on LLMs Inference Breakdown

Despite the exceptional performance of LLMs, deploying them at inference time poses considerable financial burdens due to the substantial scale that requires massive GPU memory. These memory usages can be divided into three primary components: the volume of model parameters, the size of the activation buffer, and the size of the KV cache. Model compression (Frantar and Alistarh, 2022; Xu et al., 2024; Wang et al., 2024) has shown significant advancements in tackling the first two components. Specifically, Pruning methods (Han et al., 2015; Zhang et al., 2024a; Sun et al., 2024b) discard the redundant or less important part of the model. Quantization methods (Lin et al., 2024; Dettmers et al., 2022; Shao et al., 2024) reduce the bits needed for model parameters. Different from these works, we aim to compress the size of the KV cache.



(a) Layer 24. (b) Layer 8. (c) Layer 0.

Figure 10: Attention patterns for input prompt: “Question: Plants and animals need food for growth. What happens to most of the food that plants produce?\nAnswer:" w.r.t different layers and heads in Mistral(7B).