TALE: Token-Adaptive Low-Rank KVCache Approximation with Reconstruction Elimination

Jaeseong Lee*, **Seung-won Hwang***, **Aurick Qiao**,
Daniel Campos, **Zhewei Yao**, **Yuxiong He**
Snowflake AI Research, Seoul National University, South Korea*

Abstract

KVCache, by storing key-value pairs for reuse, has been crucial for enhancing inference efficiency for large language models (LLMs). However, the increasing memory demands of KVCache, especially with recent trends of longer input sequences, present a major challenge. In this work, we propose an innovative **token-adaptive low-rank** approximation strategy for KVCache compression. By applying varying ranks based on token significance, our method compresses KVCache efficiently while retaining critical information. Moreover, we introduce a **lazy approximation** technique, which approximates lazily only when needed, alongside a **reconstruction-free** design to bypass costly recalculations. Combined with multi-level quantization, this method reduces KVCache size by 9.1× on the Llama-3.1-8B model, with minimal performance degradation on complex tasks such as GSM8K. Moreover, our custom attention implementation shows up to 2× latency reduction compared to the conventional method in long context scenarios. The code is publicly available.

1 Introduction

Large language models (LLMs) have become the de-facto standard to solve various tasks (OpenAI, 2023; Touvron et al., 2023; Dubey et al., 2024; Hui et al., 2024; Yang et al., 2024; Abdin et al., 2024; Team et al., 2024). Yet, as these models grow larger, so do the challenges associated with serving them efficiently. One prominent issue is the sheer size of the KVCache, a critical component in the token generation process. By storing key and value hidden representations to avoid repetitive computations, KVCache becomes a memory bottleneck, especially during long context generation (Yuan et al., 2024).

In response, researchers have explored various compression techniques to manage KVCache more effectively. Solutions range from selective token eviction to minimize unnecessary caching (Zhang et al., 2023; Xiao et al., 2023), to reducing precision through quantization (Hooper et al., 2024; Liu et al., 2024).

Among various techniques to compress KVCache, we focus on low-rank approximation of KVCache (Chang et al., 2024; Saxena et al., 2024). The core idea is to approximate key and value matrices with lower-rank representations, caching these smaller matrices instead of the full ones. Often, this approach is compounded with quantization to achieve higher compression ratios (Chang et al., 2024).

However, we observe three problems that reside in such design. First, different tokens in KVCache are known to have different importance (Xiao et al., 2023; Hooper et al., 2024), but existing low-rank methods do not adapt to this variation.

Second, existing approaches eagerly approximate key and value representations (Figure 1a green) due to their offline low-rank approximation (Figure 1a blue). This approximates the key and value representations before caching, yielding an unnecessary quality loss in computing next token probabilities.

Lastly, the conventional low-rank technique has eliminated the cost of reconstructing compressed caches into their full shape, by fusing this multiplication with the next computation. However, this only works for value cache, and not for key cache due to RoPE embedding (Figure 1a), which exacerbated the computational load in LLM inference (Figure 2).

In this paper, we propose a novel KVCache low-rank approximation method, TALE, to overcome all three problems. First, we introduce a **token-adaptive low-rank** approximation that dynamically adjusts the compression level based on token importance, drawing inspiration from token eviction techniques (Xiao et al., 2023). This enables variable low-rank approximations across tokens, tailored to their respective roles in generation.

*Work done while visiting Snowflake. Correspond to seungwonh@snu.ac.kr.

Transactions of the Association for Computational Linguistics, vol. 13, pp. 1298–1318, 2025. https://doi.org/10.1162/TACL.a.39
Action Editor: Naoaki Okazaki. Submission batch: 4/2025; Revision batch: 5/2025; Published 10/2025.

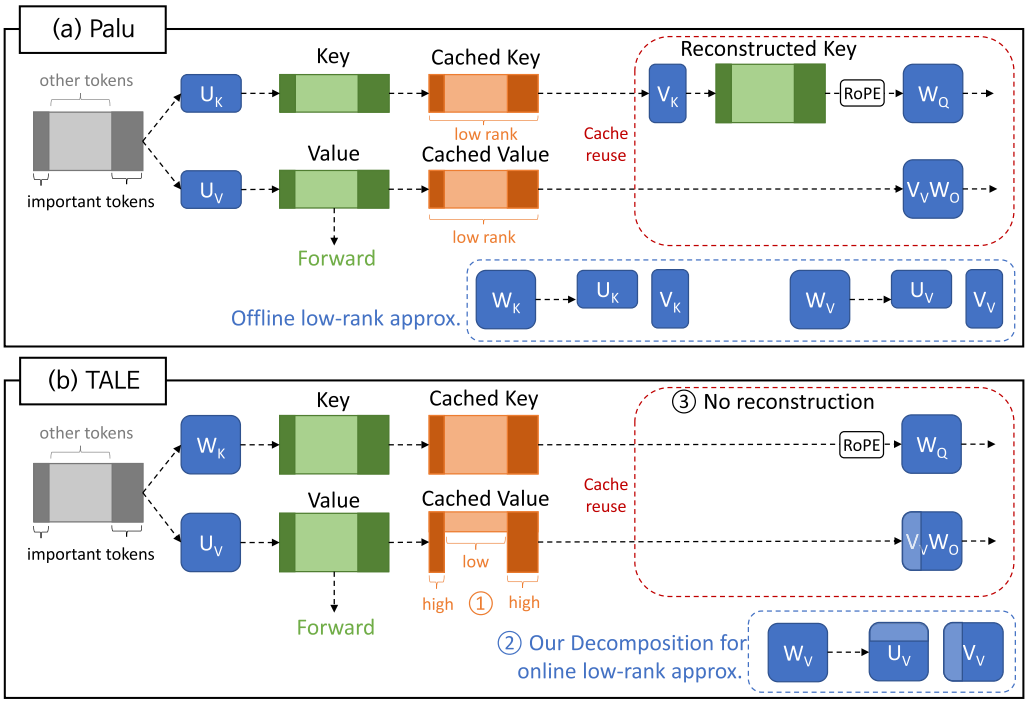

Figure 1: Comparison of Palu (existing work), and TALE (this work) attention head design. We point out three distinctions from Palu: ① TALE considers token-importance for low-rank compression, unlike Palu's uniform compression. ② TALE develops online low-rank approximation, which enables passing full representation for the model forward, while lazily compressing when caching the representations. ③ TALE removes the reconstruction overhead from Palu.

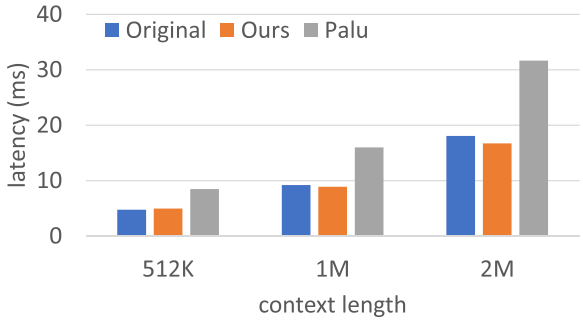

Figure 2: Latency comparison of a Llama-2-7B attention head with each low-rank KVCache compression methods.

Second, we introduce **lazy approximation**, which compresses the key and value representation only in the cached version. We enable this by efficiently implementing online low-rank approximation, leveraging the Eckart-Young-Mirsky theorem (Eckart and Young, 1936).

Finally, we **eliminate the reconstruction of the key cache**, by omitting low-rank approximation for the key cache. While this increases the size of the key cache, we show that our method can compress the value cache much more aggressively, to achieve an overall higher compression ratio than existing low-rank approximation methods. We demonstrate the overhead reduced from our custom attention implementation, showing almost 2× latency reduction compared to the conventional low-rank method.

To sum up, we devise TALE by integrating all of these tactics—**T**oken-**A**daptive, **L**azy approximation, and reconstruction **E**limination. We also show that TALE can be synergetically compounded with multi-level quantization, allowing further compression. Our contribution can be summarized as follows:

- We point out three limitations in existing KVCache low-rank methods: Token-unconditional uniform low-rank approximation, eager approximation of key and values, and increasing reconstruction cost.
- We propose TALE with three contributions, overcoming each of the key challenges: **T**oken-**A**daptive, **L**azy approximation, and **E**liminating reconstruction, which can be synergetically compounded with quantization as well.
- TALE can reduce the size of KVCache by 9.1×, with a negligible loss on GSM8K

performance for Llama-3.1-8B, contrary to existing low-rank approximation showing 44× higher accuracy loss than ours.

- Code is publicly available.[1]

2 Proposed Method

2.1 Preliminaries

KVCache is crucial for improving inference efficiency in transformers, particularly for long sequences, by caching computed keys and values for reuse in future attention steps. This setup greatly reduces redundant computations, but incurs a memory cost that grows rapidly with respect to the context size.

Formally, to decode a token corresponding to the ith input token, x_i in a vanilla transformer without KVCache, given W_Q, W_K, W_V, W_O as weight matrices, we need to compute q, k, v, the query, key, and values, as follows:

$$q_i = x_i \cdot W_Q \tag{1}$$
$$k_{1:i} = x_{1:i} \cdot W_K \tag{2}$$
$$v_{1:i} = x_{1:i} \cdot W_V \tag{3}$$
$$o_i = \text{softmax}\left(\frac{q_i \cdot k_{1:i}^T}{\sqrt{d}}\right) \cdot v_{1:i} \cdot W_O + x_i \tag{4}$$

By caching $k_{1:i}, v_{1:i}$, we can avoid the repeated computation in Eqs. 2, 3 as follows:

$$k_i^c = k_{1:i}, v_i^c = v_{1:i} \tag{5}$$
$$k_{1:i} = \text{concat}(k_{i-1}^c, x_i \cdot W_K) \tag{6}$$
$$v_{1:i} = \text{concat}(v_{i-1}^c, x_i \cdot W_V) \tag{7}$$

The key idea of existing KVCache low-rank approximation (Chang et al., 2024; Saxena et al., 2024) is applying low-rank approximation to W_K, W_V. Suppose $W_V \in \mathbb{R}^{m\times n}$, and $W_V \approx U_V^r V_V^r$ where $U_V^r \in \mathbb{R}^{m\times r}, V_V^r \in \mathbb{R}^{r\times m}, r < \min(m, n)$. Similarly, we can define U_K^r, V_K^r. By replacing the Eqs. 2, 3 with these low-rank approximations, we get:

$$k_{1:i}^h = x_{1:i} \cdot U_K^r, k_{1:i} = k_{1:i}^h \cdot V_K^r \tag{8}$$
$$v_{1:i}^h = x_{1:i} \cdot U_V^r, v_{1:i} = v_{1:i}^h \cdot V_V^r \tag{9}$$

With $k_{1:i}^h, v_{1:i}^h \in \mathbb{R}^{i\times r}$, which is much smaller than $k_{1:i}, v_{1:i} \in \mathbb{R}^{i\times n}$, we obtain the reduced KVCache:

$$k_i^{ch} = k_{1:i}^h, v_i^{ch} = v_{1:i}^h \tag{10}$$

However, we need to reconstruct the $k_{1:i}, v_{1:i}$ from the KVCache as follows:

$$k_{1:i} = \text{concat}(k_{i-1}^{ch} \cdot V_K^r, x_i \cdot U_K^r V_K^r) \tag{11}$$
$$v_{1:i} = \text{concat}(v_{i-1}^{ch} \cdot V_V^r, x_i \cdot U_V^r V_V^r) \tag{12}$$

This reconstruction step, requiring V_K^r multiplication, is costly and even increases as context length increases, but unavoidable in cases where RoPE embeddings are used, which are common in modern LLM architectures (Touvron et al., 2023; Jiang et al., 2023; Guo et al., 2024; Bai et al., 2023), as we detail in Section 2.2.3.

2.2 TALE

We propose TALE, with the following three main distinctions from existing KVCache low-rank compression methods, each of which is discussed in subsections:

1. **Token-Adaptive:** Existing approaches apply uniform compression across all tokens, disregarding the varying importance of individual tokens (Xiao et al., 2023). We introduce token-aware multi-level compression, which considers the importance of specific tokens during the compression process.

2. **Lazy Approximation:** Current techniques perform an eager approximation of the key and values, while we propose a lazy alternative, approximating the key and values only in their cached version.

3. **Reconstruction Elimination:** Existing methods incur increasing reconstruction costs with longer context lengths to deal with RoPE embeddings, undermining the efficiency gains from compression. We eliminate these reconstruction costs.

2.2.1 Token-Adaptive Compression

KVCache compression studies have shown that tokens vary in importance, which has been applied to adapt compression levels accordingly (Xiao et al., 2023; Liu et al., 2024; Hooper et al., 2024). However, uniform compression across all tokens (Figure 1a orange) has been a convention for KVCache. To address this, we propose a multi-level low-rank approximation that adapts to token importance. We treat the first α tokens, named

[1] https://github.com/thnkinbtfly/TALE.

sink tokens, and the last tokens as important, following Xiao et al. (2023) (Figure 1b orange). We show that this strategy generalizes to adversarial scenarios where middle tokens are important in Section 3.1.7.

Formally, consider a two-level approximation scenario, with rank $r_0 \leq r_1$. As a control knob of the compression ratio, we ensure the ratio of higher rank r_1 tokens does not exceed a threshold ratio p. Suppose the VCache has length κ, denoted as $v_{1:\kappa}$. Initially, we leave $v_{1:\alpha}$ intact, applying the higher rank r_1 to the last tokens in VCache, or, $v_{[\kappa-p(\kappa-\alpha)]:\kappa}$, and the lower rank r_0 to the middle tokens, or, $v_{\alpha:[\kappa-p(\kappa-\alpha)]}$. When a new token is added to the VCache, it is assigned the higher rank r_1. If the ratio of higher rank r_1 tokens exceeds p during addition, we move the first token in the r_1 region to the r_0 region, lowering its rank.

We can easily generalize this process for multiple ranks, $r_0 \leq \cdots \leq r_\beta$, as elaborated in Appendix A.

2.2.2 Lazy Approximation

To motivate lazy approximation, we first describe eager approximation, where the low-rank approximation is performed offline, replacing W_V with a smaller-rank matrix U_V^r, as in Figure 1a (blue). This, in turn, reduces the full-rank value tensor (Figure 1a green), yielding a low-quality next token probability distribution, and leading to unnecessary approximation errors.

In contrast, as in Figure 1b, we apply a lazy approximation strategy, keeping full information of the green box. During each forward pass, we thus calculate the next token representation more accurately with full information, and lazily compress only the cached version.

We efficiently implement lazy approximation with our novel online low-rank approximation technique, which requires only a truncation operation in the inference step, as detailed in Section 2.2.3.

2.2.3 Reconstruction Elimination

Illustration of Single-Head Attention Case To eliminate the reconstruction overhead from low-rank approximation, first, we remove the reconstruction cost entirely by leveraging matrix fusion in VCache, as demonstrated in Chang et al. (2024) and Saxena et al. (2024). This is easily done by redefining W_O^r as $V_V^r W_O$. Then $v_{1:i} \cdot W_O$ in Eq. 4 is converted by the associative law:

$$v_{1:i} \cdot W_O \tag{13}$$
$$= \text{concat}(v_{i-1}^{ch} \cdot V_V^r, x_i \cdot U_V^r V_V^r) \cdot W_O \tag{14}$$
$$= \text{concat}(v_{i-1}^{ch}, x_i \cdot U_V^r) \cdot W_O^r \tag{15}$$

However, this fusion cannot be applied to KCache when using RoPE embeddings, which are common in many LLMs. With RoPE embeddings, the $q_i \cdot k_{1:i}^T$ in Eq. 4 becomes as follows:

$$\begin{aligned} q_i \cdot k_{1:i}^T &= \text{RoPE}(x_i \cdot W_Q) \cdot \text{RoPE}(x_{1:i} \cdot W_K)^T \\ &= \text{RoPE}(x_i \cdot W_Q) \cdot \text{RoPE}(v_i^{ch} \cdot V_K^r)^T \end{aligned} \tag{16}$$

where V_K^r cannot be fused into W_Q as done in VCache.

As a result, we refrain from applying low-rank approximation to KCache in these cases. While this may seem to reduce compression efficiency, our experiments demonstrate that even with this restriction, our method still outperforms existing low-rank KVCache approximation techniques.

Online Low-rank Approximation With varying levels of low-rank approximation, it may seem necessary to introduce multiple W_O^r matrices, and matrix multiplication is required for the conversion to a lower rank in Section 2.2.1. However, multiplication can be removed if we design $U_V = U\sqrt{S}, V_V = \sqrt{S}V$, where $W_V = USV$ is the SVD result with singular values sorted in descending order, by the Eckart-Young-Mirsky theorem (Eckart and Young, 1936), as W_O^r is simply the truncation of $V_V W_O$ with the first r rows. Similarly, reducing the rank from r_j to r_{j-1} only involves truncating the last r_{j-1} columns, which enables a reconstruction-free low-rank approximation without extra computational cost.

Customizing Multi-Head Attention For Reconstruction Removal To cover most LLMs with multi-head, we also validate our approach of eliminating reconstruction costs by adapting multi-head attention. To illustrate, consider two attention heads with a batch size of 1. Let the attention weights, the output of softmax in Eq. 4 of each head, be $S_0, S_1 \in \mathbb{R}^{\sigma \times \kappa}$, where σ is the length of the query, and κ is the length of the key and value. Let $\mathcal{V}_0, \mathcal{V}_1 \in \mathbb{R}^{\kappa \times n}$ be the $v_{1:i}$ in Eq. 4 of each head, where n is the output dimension of the value

matrix of each head. Let $W_O = \begin{pmatrix} O_{00} & O_{01} \\ O_{10} & O_{11} \end{pmatrix}$, where each $O_{i,j} \in \mathbb{R}^{n,n}$. The original attention output is constructed as follows:

$$\begin{pmatrix} S_0\mathcal{V}_0 & S_1\mathcal{V}_1 \end{pmatrix} \cdot \begin{pmatrix} O_{00} & O_{01} \\ O_{10} & O_{11} \end{pmatrix} \tag{17}$$

Now, suppose we group the two attention heads and decompose W_V across both. If we decompose $\mathcal{V}_0, \mathcal{V}_1$ as

$$\begin{pmatrix} \mathcal{V}_0 & \mathcal{V}_1 \end{pmatrix} = \begin{pmatrix} U_0^r & U_1^r \end{pmatrix} \cdot \begin{pmatrix} V_{00}^r & V_{01}^r \\ V_{10}^r & V_{11}^r \end{pmatrix} \tag{18}$$

where $\begin{pmatrix} U_0^r & U_1^r \end{pmatrix} \in \mathbb{R}^{\kappa \times r}$, and $\begin{pmatrix} V_{00}^r & V_{01}^r \\ V_{10}^r & V_{11}^r \end{pmatrix} \in \mathbb{R}^{r \times 2n}$. The attention output can now be rewritten as:

$$\begin{pmatrix} S_0(U_0^r V_{00}^r + U_1^r V_{10}^r) & S_1(U_0^r V_{01}^r + U_1^r V_{11}^r) \end{pmatrix} \cdot \begin{pmatrix} O_{00} & O_{01} \\ O_{10} & O_{11} \end{pmatrix} \tag{19}$$

For an efficient implementation, we need to consider the following requirement:

- We need to merge $\begin{pmatrix} V_{00}^r & V_{01}^r \\ V_{10}^r & V_{11}^r \end{pmatrix}$ with $\begin{pmatrix} O_{00} & O_{01} \\ O_{10} & O_{11} \end{pmatrix}$ offline to build some matrix O^r, so that we can obtain the same result as Eq. 19.
- The associative law should be applied between S_is and U_j^rs first, since they share the largest dimension, κ, which is significantly larger than σ, r, n in long context generation.

In the end, we decompose Eq. 19 meeting these requirements. We operate S_is and U_j^rs:

$$\begin{pmatrix} S_0U_0^r & S_0U_1^r \\ S_1U_0^r & S_1U_1^r \end{pmatrix} = \begin{pmatrix} S_0 \\ S_1 \end{pmatrix} \cdot \begin{pmatrix} U_0^r & U_1^r \end{pmatrix} \tag{20}$$

then we reshape this matrix, and multiply with the pre-built matrix on the right-hand side:

$$\begin{pmatrix} S_0U_0^r & S_0U_1^r & S_1U_0^r & S_1U_1^r \end{pmatrix} \cdot \begin{pmatrix} V_{00}^r O_{00} & V_{00}^r O_{01} \\ V_{10}^r O_{00} & V_{10}^r O_{01} \\ V_{01}^r O_{10} & V_{01}^r O_{11} \\ V_{11}^r O_{10} & V_{11}^r O_{11} \end{pmatrix} \tag{21}$$

which yields the same result as Eq. 19.[2]

[2]A similar construction can be found in the Palu (Chang et al., 2024) implementation as well, but we additionally argued why this implementation is necessary for our purpose of reducing overhead.

This restructuring efficiently fuses the low-rank approximation with the attention mechanism, confirming that reconstruction costs are removed. We can generalize this into Multi-Level, and GQA, as detailed in the Appendix B and C. We summarize the algorithm for the reconstruction removal in the Appendix.

2.3 On Compounding with Quantization

Compounding quantization with our proposed multi-level low-rank approximation introduces the following opportunities in each aspect:

Token-Adaptive: In alignment with the multi-level low-rank approximation, quantization can be strategically applied based on token significance. By assigning b_j-bit quantization to regions corresponding to rank r_j, with $b_0 \leq \cdots \leq b_\beta$, higher bit-widths are allocated to more important tokens. This complements our multi-level compression, where leveling is guided by token-importance.

Lazy Approximation: KVCache quantization is more robust than low-rank approximation (Chang et al., 2024). The eager quantization can be directly used without severe performance degradation.

Reconstruction-Free: A key advantage of quantization is that it can be applied to the KVCache without introducing reconstruction overhead, unlike low-rank decomposition where compressing KCache introduces reconstruction overhead. This allows us to compress both KCache and VCache effectively without compromising the integrity of positional information.

3 Experiments

As a representative large language model, we use a recently released Llama-3.1-8B-Instruct (Dubey et al., 2024), and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023).

Research Questions In this section, we design experiments and answer the following research questions:

- RQ1: Is our token-adaptive compression effective?
- RQ2: Is our lazy approximation beneficial?
- RQ3: Is the reconstruction cost from conventional low-rank KVCache approximation eliminated?

- RQ4: Can TALE be faster and more memory-efficient than FlashAttention?
- RQ5: Does TALE generalize over various model sizes?
- RQ6: How does TALE perform in different languages?
- RQ7: Is TALE weak in capturing the importance of middle tokens?
- RQ8: How does TALE perform in different length of prompts?
- RQ9: Does TALE outperform training-required methods?

Comparisons We compare TALE with methods from each line of KVCache compression techniques:

- Palu (Chang et al., 2024): A representative low-rank approximation technique. We use 3-bit quantization and a 40% lower rank version to meet a similar compression ratio as ours. Following their default proposal, we group every 4 heads.
- KIVI (Liu et al., 2024): A representative quantization technique we used. We use 2-bit quantization, to meet a similar compression ratio as ours.
- HQQ (Badri and Shaji, 2023): Another representative quantization technique. We use 2-bit quantization, to meet a similar compression ratio as ours.
- Attention Sink (Xiao et al., 2023): A representative KVCache pruning technique. We keep the same number of tokens at the front, and set the number of tokens in the last part to be similar to ours.
- ShadowKV (Sun et al., 2024a): A state-of-the-art baseline using KCache decomposition and KVCache pruning with CPU offloading.

Tasks and Datasets We mainly evaluate on generation tasks, such as GSM8K (Cobbe et al., 2021). To evaluate generation performance in a long context scenario, we also evaluate on LongBench (Bai et al., 2024). Following previous work (Chang et al., 2024), we choose 8 tasks: LCC, Multi-news (M-News), Qasper, QMSum, RepoBench-P (R-Bench), SAMSum, TREC, and TriviaQA (TriQA). Finally, we also conduct perplexity-based evaluations, with ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and MMLU (Hendrycks et al., 2021).

Implementation Details We use $\alpha = 4$, to compare with Attention Sink (Xiao et al., 2023), which keeps the first 4 tokens as well. We use $p = 0.1, \beta = 1$, and let r_0 be the 50% of the full rank. We use $b_0 = 2, b_1 = 4$ for quantization. We use SVD-LLM (Wang et al., 2025) to perform SVD of each weight matrix, and use Palu (Chang et al., 2024) for SVD of grouped heads. We use KIVI quantization (Liu et al., 2024). For RQ3, we use a static cache similar to that of GPT-FAST[3] to reduce memory overhead.

We evaluate LongBench tasks with code provided by Palu (Chang et al., 2024), and evaluate the rest of the tasks on LM-EVALUATION-HARNESS (Gao et al., 2021). All experiments except for RQ5 are conducted on one H100 80GB GPU. Each evaluation is done within 5 hours. Evaluation is done only once, since we introduce no randomness in our experiment.

3.1 Experimental Results

3.1.1 RQ1: Token-Adaptive Compression is Effective

Table 1 demonstrates the effectiveness of TALE. Compared to the previously proposed low-rank KVCache approximation method, Palu, TALE shows significant improvement on the performance– For example, For GSM8K, TALE does not lose accuracy, while Palu loses by -53.07 percentage point. This highlights the merit of our token-adaptive low-rank approximation. Mistral-based experiments show a similar trend as well (Table 3).

We can also observe a consistent trend when comparing with the pruning technique, Attention Sink. Even with a much lower compression ratio, they lose information in the context by token eviction. Similarly, ShadowKV loses performance with a large margin in GSM8K, even though they keep KVCache much more than TALE by CPU offloading. On the other hand, TALE keeps all the tokens in the context, maintaining the performance.

Finally, TALE outperforms our quantization baselines as well. With token-adaptive compression design, we can aggressively compress

[3] https://github.com/pytorch-labs/gpt-fast.

	Ratio	GSM8K	Avg(→)	LCC	M-News	Qasper	QSum	RBen	SSum	TREC	TriQA
Orig	1.00×	81.65	48.78	63.23	27.14	15.60	23.80	52.38	43.45	72.50	92.14
Attention Sink	4.90×	56.41	42.74	59.28	17.76	8.07	22.77	50.16	42.38	50.50	90.99
KIVI (2bit)	8.00×	79.91	48.18	62.26	26.77	13.96	23.36	50.67	43.63	72.50	92.32
HQQ (2bit)	8.00×	76.73	47.78	59.20	26.81	14.09	23.74	49.23	44.04	72.50	92.66
Palu-40% (3bit)	8.89×	28.58	43.31	45.74	25.37	14.98	24.03	34.68	42.23	71.00	88.41
ShadowKV	3.2×(27.8×*)	58.68	**49.28**	65.38	27.34	14.55	23.56	55.82	41.99	73.00	92.62
TALE	9.14×	**81.88**	48.79	63.36	25.83	17.35	24.25	51.03	44.39	72.50	91.58

Table 1: Generation-based evaluation between different KVCache compression techniques. We report the compression ratio, GSM8K, and LongBench results. The best performance is emphasized with **bold**. *: Use CPU offloading to save GPU memory.

	Ratio	ARC	HS	MMLU	Avg
Original (16bit)	1.00×	56.74	59.34	68.09	61.39
Palu-40% (3bit)	8.89×	50.17	53.33	57.13	53.54
TALE	**9.14**×	**56.57**	**59.30**	**68.37**	**61.41**

Table 2: Perplexity-based evaluation between different KVCache compression techniques. We report the compression ratio, ARC-Challenge (ARC; 25-shot), HellaSwag (HS; 10-shot), and MMLU (5-shot). The best performance is emphasized with **bold**.

less-important tokens, and assign higher bits to the important tokens, while achieving a higher compression ratio and performance.

3.1.2 RQ2: Lazy Approximation is Beneficial

Table 2 emphasizes the importance of lazy approximation. For example, the existing low-rank approximation method, Palu, struggles to maintain the performance even in perplexity-based evaluations, where no generation is needed, i.e., no KVCache is needed. On the other hand, TALE successfully maintains the performance.

This performance gap is consistently observed in Tables 1 and 3, since the eager approximation propagates the error through the generation steps to reduce overall accuracy.

3.1.3 RQ3: Reconstruction Cost is Eliminated

To show that the reconstruction cost is successfully eliminated in our design, we conduct experiments with real implementation with long context length, 512K, 1M, and 2M, with a batch size of 1. Following previous work—Palu (Chang et al., 2024)—we experiment with an attention head of eager mode in TRANSFORMERS (Wolf et al., 2020), and we use Llama-2-7B (Touvron et al., 2023). For Palu, we use a compression ratio of 50% following their implementation, without quantization. We use $\alpha = 0, p = 0.1$, and r_0 as 50% of full rank, whose evaluation score yet outperforms 50% compression of Palu. We also extend our experiment with a more recent LLM, Llama-3.1-8B (Dubey et al., 2024), by varying context length from 128K to 4M.

Figures 2 and 3 show that TALE successfully removes the reconstruction cost which is significant in Palu (Chang et al., 2024), especially in longer context scenarios. For example, in the Llama-2-7B 2M scenario (Figure 2), TALE actually reduces the latency, while Palu nearly doubles it. We see a similar trend in Llama-3.1-8B (Figure 3)–TALE reduces the latency almost by half on context length of 2M, and we even enable longer context, 4M, where original attention faces out-of-memory (OOM) error.

3.1.4 RQ4: TALE is Faster and More Memory-Efficient than FlashAttention

While TALE could reduce the number of computations and memory operations leading to latency reduction, one may question the efficiency of the eager mode attention head in TRANSFORMERS (Wolf et al., 2020). Therefore in this section, we compare the throughput of TALE with the state-of-the-art attention implementation, FlashAttention (Dao et al., 2022; Dao, 2024).

Focusing on TALE without quantization, we begin by measuring latency at the attention head level, varying context length and batch size. Next, we evaluate end-to-end throughput using the entire model, leveraging GPT-FAST[4] for benchmarking.

As shown in Figure 4, TALE outperforms FlashAttention in larger batch sizes for the

[4] https://github.com/pytorch-labs/gpt-fast.

	Ratio	ARC	HS	MMLU	GSM8K	LBAvg
Palu-40%	8.89×	56.23	62.22	55.72	28.13	49.47
TALE	9.14×	60.58	65.38	62.14	44.88	52.59

Table 3: Mistral-7B-Instruct-v0.3 evaluation between different KVCache compression techniques with similar compression ratios. The settings are similar to Tables 1 and 2. LBAvg is the average score over LongBench tasks.

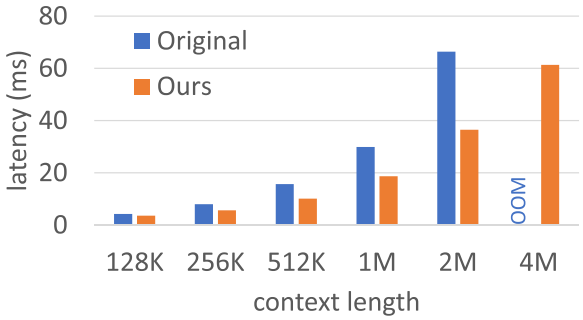

Figure 3: Latency comparison of a Llama-3.1-8B attention head with each low-rank KVCache compression methods. Palu kernel is not supported for Llama-3.1, and the original attention head faces OOM on 4M.

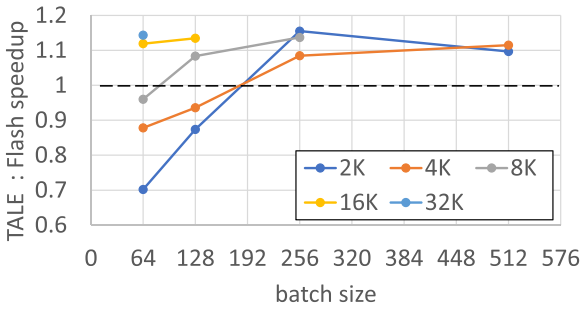

Figure 4: Relative speedup of TALE compared to FlashAttention with a Llama-2-7B attention head. Each colored line represents different context lengths.

Llama-2-7B attention head, with the performance gap widening as context length increases.

Additionally, Figure 5 demonstrates that TALE's memory efficiency enables larger batch sizes, leading to significantly higher end-to-end throughput compared to FlashAttention.

Finally, we compare the maximum context length with a finer granularity between TALE with quantization and FlashAttention on Llama-3.1-8B-Instruct: Figure 6 shows the detailed GPU memory consumption across various context lengths. We can clearly observe TALE drastically reducing the GPU memory consumption, enabling up to 1.5× more context length compared to the competitive baseline, FlashAttention.

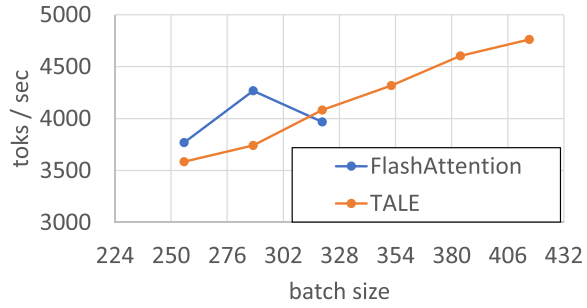

Figure 5: End-to-End throughput comparison between TALE-based Llama-2-7B and FlashAttention-based Llama-2-7B.

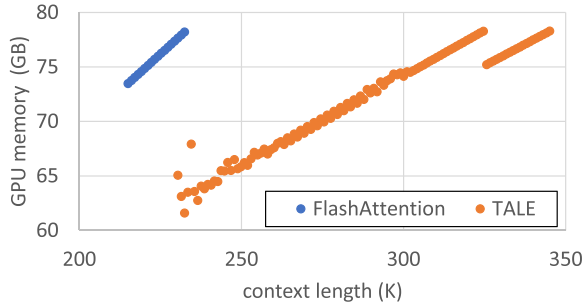

Figure 6: GPU memory consumption on H100-80GB across various context lengths, using Llama-3.1-8B-Instruct.

3.1.5 RQ5: TALE Generalizes over Model Sizes

To evaluate TALE across diverse model architectures and sizes, in addition to the 7-8B scale model results in the main table, we provide results on both a smaller 3B model and a larger 70B model, Llama-3.2-3B-Instruct, and Llama-3.3-70B-Instruct. Tables 5 and 6 show consistent results as well, which captures our generalization over a diverse range of practical model sizes.

3.1.6 RQ6: TALE's Generalization Across Languages—Findings and Limitations

Table 4 presents the evaluation results on the Multilingual GSM benchmark. Our findings indicate that TALE generalizes well across most languages, with the exception of low-resource

Model	Ratio	bn	de	en	es	fr	ja	ru	sw	te	th	zh
Original (16bit)	1.00×	0.556	0.740	0.848	0.764	0.668	0.548	0.744	0.556	0.536	0.700	0.716
TALE	9.14×	0.544	0.744	0.844	0.764	0.680	0.536	0.764	0.516	0.372	0.636	0.716

Table 4: Multilingual GSM evaluation on Llama-3.1-8B-Instruct over 11 different languages.

	Ratio	ARC	HS	MMLU	GSM8K	LBAvg
Original	1.00×	47.44	52.99	59.81	71.80	49.23
KIVI	8.00×	47.61	53.20	60.74	68.84	48.97
TALE	9.14×	47.44	53.01	59.86	72.10	49.26

Table 5: Llama-3.2-3B-Instruct evaluation between different KVCache compression techniques with similar compression ratios. The settings are similar to Tables 1 and 2. LBAvg is the average score over LongBench tasks.

	Ratio	ARC	HS	MMLU	GSM8K	LBAvg
Original	1.00×	68.26	67.33	82.22	94.77	51.01
TALE	9.14×	68.26	67.34	82.22	95.00	51.01

Table 6: Llama-3.3-70B-Instruct evaluation between different KVCache compression techniques with similar compression ratios. The settings are similar to Tables 1 and 2. LBAvg is the average score over LongBench tasks.

languages such as Telugu (te). We hypothesize that addressing imbalances in the training corpus could enhance KVCache resilience in these cases. However, we leave this as an avenue for future exploration.

3.1.7 RQ7: TALE Captures the Importance of Middle Tokens

Following Xiao et al. (2023), TALE prioritizes sink tokens and last tokens, ensuring they are retained with higher fidelity. A potential concern is whether this approach neglects critical information in the middle of sequences, which is a well-documented weakness of LLMs, known as the ‘‘lost-in-the-middle’’ problem. To test the robustness of this problem, Needle-in-a-Haystack is a task devised to place key information adversarially, among which we evaluate RULER benchmark (Hsieh et al., 2024), as it requires reasoning over multiple needles, rather than retrieving a single needle.

Despite our concern, as shown in Table 9, TALE successfully handles middle-token retrieval, maintaining strong performance over original models, across various RULER benchmark tasks, differently from our baseline, Palu. This demonstrates that TALE successfully preserves critical middle-token information even in its lowest-fidelity representations—highlighting the effectiveness of low-rank decomposition validated, even in an extreme adversarial scenario to the token selection we used.

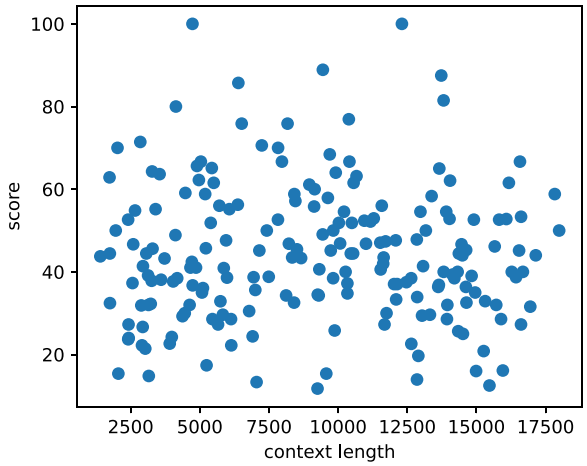

Figure 7: Score distribution of a LongBench task (SamSum) over various length of prompts using TALE on Llama-3.1-8B-Instruct.

3.1.8 RQ8: TALE Generalizes over Length of Prompt

Since TALE compresses large portion of tokens, one may question whether samples with longer prompts would suffer performance degradation.

Nevertheless, we observe that TALE generalizes over various length of prompts. Figure 7 shows an example task of SamSum with TALE on

	LCC	M-News	Qasper	QmSum	R-Bench	SAMSum	TREC	TriQA
pearson correlation	0.025	−0.065	0.023	−0.067	0.025	−0.018	0.049	0.046

Table 7: Pearson correlation coefficient between the context length and score in the LongBench, using TALE on Llama-3.1-8B-Instruct.

	GSM8K	LongBench Avg
TALE	**81.88**	**48.79**
TALE-sink	81.12	47.49
TALE-last	80.97	48.55

Table 8: Ablation study of token-adaptive multi-level design. The best performance is emphasized with bold.

	Ratio	FWE	VT	QA1	QA2	avg
original	1.00×	85.42	97.29	82.29	59.38	81.09
Palu-40%	8.89×	72.22	69.79	59.38	38.54	59.98
TALE	9.14×	87.15	90.42	83.33	55.21	79.03

Table 9: RULER (Needle-in-a-Haystack) test with Llama-3.1-8B-Instruct. We evaluate frequent word extraction (FWE), variable tracking (VT), and two different question answering which is SQuAD (QA1) and HotPotQA (QA2).

Llama-3.1-8B-Instruct—there is no tendency of performance degradation as the length of prompt gets longer. Table 7 shows the Pearson correlation coefficient between the context length and the score in the LongBench evaluation with TALE on Llama-3.1-8B-Instruct. It implies that there is no correlation—TALE generalizes over various length of prompts.

3.1.9 RQ9: TALE Outperforms Training-Required Method

To compare with KVCache compression methods requiring training, such as CLA (Brandon et al., 2024), which enforces KVCache sharing across layers from the pretraining stage. Since the pretraining stage of LLM is prohibitively costly, we employ LoRA (Hu et al., 2022) with rank of 256 to update Llama-3.1-8B-Instruct. We perform knowledge distillation using ultrachat_200k, MetaMathQA, Magicoder-OSS-Instruct-75K, lmsys-chat-1m, and SlimOrca datasets, for 1 epoch. While such training enables CLA, the performance is far below our training-free method, TALE (Table 10). This indicates while various pretraining-stage KVCache compression techniques are available, it is challenging to apply them for existing LLM without retraining them. In contrast, TALE is applicable to LLMs without such challenging training stage.

	Ratio	train	GSM8K
CLA	2.00×	✗	0.00
CLA	2.00×	✓	48.83
TALE	9.14×	✗	**81.88**

Table 10: Comparing with KVCache compression method requiring training.

3.2 Ablation Studies

3.2.1 Ablation of Token-Adaptive Multi-Level Design

TALE leverages token-adaptive multi-level design, inspired by Attention Sink, strongly attending to initial **sink** tokens and **last** tokens, validated in Table 8.

More specifically, TALE keeps the first α sink tokens intact and keeps the $p = 0.1$ ratio as a high rank for the last tokens. We first ablate the sink heuristics, by keeping $\alpha = 0$ (second row), where performance degrades. Similarly, we ablate the last token heuristics by setting $p \approx 0$ (third row), which also leads to decreased performance. Overall, our design of assigning high rank for sink and recent tokens is empirically validated.

3.2.2 Ablation of Grouping Heads

To show the effectiveness of grouping heads for LLMs without GQA design (Eq. 30), we leverage Llama-2-13B (Touvron et al., 2023) to estimate the impact of prebuilt matrix size. We use the same settings as in the main experiments, except for setting $g = 4$ or $g = 1$, and compare with $g = 32$, meaning no group-head strategy is applied.

Table 11 shows that grouping heads is significantly important for keeping the model size. Although removing the group-head structure introduces the best performance in some benchmarks, it comes with a higher model size—for example, with 32 heads, the model size increases up to 3.2×. In contrast, grouping heads yields no

	Model Size	KV Compress. Ratio	GSM8K	LongBench Avg
Orig (Llama-2-13b)	**1.0**×	1.00×	22.82	44.77
TALE ($g = 4$)	<u>1.2×</u>	9.28×	**22.90**	<u>43.69</u>
TALE ($g = 1$)	**1.0**×	9.28×	21.76	43.41
TALE w/o grouped head	3.2×	9.28×	<u>22.59</u>	**44.53**
Palu-40% ($g = 4$, 3bit)	<u>1.2×</u>	8.89×	13.65	41.04

Table 11: Comparing model size, KVCache compression ratio, and accuracies by varying the head group size on Llama-2-13B, a representative non-GQA LLM. The best performance is emphasized with **bold**, and the second-best is underlined.

	Ratio	GSM8K	Avg(→)	LCC	M-News	Qasper	QmSum	R-Bench	SAMSum	TREC	TriQA
Orig	1.00×	81.65	48.78	63.23	27.14	15.60	23.80	52.38	43.45	72.50	92.14
Palu-30%	1.43×	41.55	47.61	57.38	26.44	14.40	23.26	53.62	43.09	73.00	89.72
TALE	1.46×	**80.74**	**48.10**	60.64	25.14	14.94	25.33	50.47	43.47	73.00	91.82

Table 12: Generation-based evaluation between different KVCache low-rank compression techniques, without quantization. We report the compression ratio, GSM8K, LongBench results. The best performance is emphasized with **bold**.

or marginal performance degradation, while maintaining the model size almost intact. We highlight that our design with $g = 1$ already outperforms Palu-40%, which requires a larger model size than ours.

3.2.3 Ablation of Quantization

While we showed that TALE can achieve a significant compression ratio by compounding with quantization, we also show that TALE is a successful low-rank KVCache compression method even without quantization. To test this, we use a 30% compressed version of Palu and apply TALE with similar settings, except for reducing r_0 to 30% of the full rank to match Palu's compression ratio.

Table 12 depicts that TALE outperforms the existing method, Palu, by a high margin. For example, for GSM8K, the loss of TALE is only -0.91 percentage point, which is $44.1\times$ smaller than that of Palu.

3.2.4 Quality-Efficiency Tradeoff in Rank Selection

Table 13 presents the LongBench average score across different ranks, all maintaining a similar KVCache compression ratio. The results indicate that using a smaller rank than our chosen hyperparameter significantly degrades performance due to the aggressive application of low-rank decomposition. Conversely, increasing the rank requires reducing p, the proportion of ''important tokens'' to maintain the compression ratio, which also leads to performance degradation. Thus, our choice represents the optimal balance.

rank (r_0)	p	compression ratio	LBAvg
37.5%	0.15	9.05×	48.45
50%	0.1	9.14×	**48.79**
62.5%	0.05	9.18×	48.36

Table 13: LongBench average score over various ranks with TALE on Llama-3.1-8B-Instruct with similar KVCache compression ratios.

4 Related Work

To enable longer-context generation, various KVCache compression techniques have been proposed. At the training stage, methods such as cross-layer key-value sharing (Brandon et al., 2024), linear attention (Sun et al., 2024b), and MLA (DeepSeek-AI et al., 2024) significantly reduce KVCache size. However, these approaches require modifications during training, making them incompatible with pre-trained models and thus outside the scope of our work.

In contrast, our focus of **training-free KV-Cache compression** is particularly valuable because most widely used LLMs, such as Llama (Dubey et al., 2024), Mistral (Jiang et al., 2023), and Qwen (Bai et al., 2023), are not trained with the techniques cited above, rather rely on post-training optimizations. Post-training KV-Cache compression methods fall into three main

categories: pruning, low-rank decomposition, and quantization, each presenting complementary trade-offs in efficiency and accuracy.

KVCache Pruning aims at identifying and evicting less important key-value pairs from the cache. Toward the goal, classification of unimportant parts to discard in the cache is a key technique, and token eviction methods such as Attention Sink (Xiao et al., 2023) is an illustrative example, using token importance to decide which parts of the cache should be retained or discarded.

KVCache Low-rank Decomposition seeks to approximate the key-value matrices as low-rank structures, storing these compressed representations rather than the full KVCache. However, a significant challenge is reconstruction, especially in models using RoPE embeddings, where the reconstruction cost becomes a bottleneck as sequence lengths increase (Chang et al., 2024).

LoRC (Zhang et al., 2024) uses lower-rank decomposition in deeper layers to compress key-value matrices. ShadowKV (Sun et al., 2024a) integrates KCache decomposition with KVCache pruning, significantly reducing KVCache size. We could empirically compare with ShadowKV using their released code, to observe its struggle to generalize across certain tasks, highlighting their limitations in broader applications.

KVCache Quantization reduces the memory size of the cache by converting the stored key-value pairs into a lower-bit representation (Hooper et al., 2024; Liu et al., 2024). Usually, it introduces certain challenges, such as outliers induced by SVD, when compounding with other techniques, such as low-rank decomposition methods (Chang et al., 2024).

Ours combines the strengths of all three approaches: Rather than discarding less important key-value pairs (as in pruning), we retain them in a token-guided multi-level low-rank approximation with significantly reduced reconstruction costs.

By integrating quantization, our method achieves substantial compression with minimal performance trade-offs, delivering memory and computational efficiency, even in contexts with millions of tokens.

5 Conclusion

We introduced a novel token-guided multi-level low-rank approximation method for KVCache compression, addressing a new challenge of managing inference efficiency for growing trends of longer input context. By adaptively adjusting the compression levels based on token importance, our approach minimizes accuracy loss while reducing the KVCache size. Specifically, we presented an error-reducing compression technique by avoiding the convention of eager approximation of full-rank representations and introduced a reconstruction-free design to minimize computational overhead. We also extended the approach by integrating multi-level quantization, further optimizing compression.

Acknowledgments

We would like to thank the action editor and anonymous reviewers for their valuable feedback and suggestions.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel

Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 technical report: A highly capable language model locally on your phone. https://doi.org/10.48550/arXiv.2404.14219

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.298

Hicham Badri and Appu Shaji. 2023. Half-quadratic quantization of large machine learning models.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. https://doi.org/10.48550/arXiv.2309.16609

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.acl-long.172

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. 2024. Reducing transformer key-value cache size with cross-layer attention. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, and Kai-Chiang Wu. 2024. Palu: Compressing KV-cache with low-rank projection. https://doi.org/10.48550/arXiv.2407.21118v1

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. https://doi.org/10.48550/arXiv.1803.05457

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. https://doi.org/10.48550/arXiv.2110.14168

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya

Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. 2024. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan

Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U., Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A., Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu,

Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The Llama 3 herd of models.

Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218. https://doi.org/10.1007/BF02288367

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation. Zenodo. https://doi.org/10.5281/zenodo.5371628

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. https://doi.org/10.48550/arXiv.2401.14196

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder technical report. https://doi.org/10.48550/arXiv.2409.12186

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. https://doi.org/10.48550/arXiv.2310.06825

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32332–32344. PMLR.

OpenAI. 2023. GPT-4 technical report. https://doi.org/10.48550/arXiv.2303.08774

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen Attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15332–15344, Miami, Florida, USA. Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.findings-emnlp.899

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2024a. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. https://doi.org/10.48550/arXiv.2410.21465

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024b. You only cache once: Decoder-decoder architectures for language models. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju-yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. 2024. Gemma 2: Improving open language models at a practical size. https://doi.org/10.48550/arXiv.2408.00118

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. https://doi.org/10.48550/arXiv.2307.09288

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. 2025. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations*. https://doi.org/10.18653/v1/2025.naacl-long.217

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.6

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. https://doi.org/10.48550/arXiv.2409.12122

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM inference unveiled: Survey and roofline model insights. https://doi.org/10.48550/arXiv.2402.16363

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1472

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. 2024. LoRC: Low-rank compression for LLMs KV cache with a progressive compression strategy. https://doi.org/10.48550/arXiv.2410.03111

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

Appendix

A Multi-Level Token Importance

In Section 2.2.1, we addressed two-level token importance. Here we generalize it into larger-level token importance case.

Consider a multi-level approximation scenario, with rank $r_0 \leq \cdots \leq r_\beta$, and let $0 = p_{-1} \leq p_0 \leq \cdots \leq p_\beta = 1$ be the control knobs for the compression ratio. Suppose the VCache has length κ, denoted as $v_{1:\kappa}$. Initially, we leave $v_{1:\alpha}$ intact. Then we assign rank r_i to $v_{[\alpha+p_{i-1}(\kappa-\alpha)]:[\alpha+p_i(\kappa-\alpha)]}$. Whenever a new token is added to the VCache, it is assigned the highest rank r_β first, then consecutively examines the control knobs. If the length of higher rank r_{i+1} tokens exceeds $(p_{i+1}-p_i)(\kappa-\alpha)$ during addition, we move the first u tokens in the r_{i+1} region to the r_i region, lowering its rank. In practice, we efficiently move the region by batching ($u > 1$) to reduce memory operations. Especially, when combined with the multi-level quantization, for efficient dequantization and quantization operations, we enforce u to be the multiple of $\frac{v}{b_i}$, where v is the number of bits we use to pack multiple b_i bit numbers in implementation. We may perform a grid search to choose the hyperparameters.

B Reconstruction Removal Generalization to Multi-Level KVCache

In Section 2.2.3, we discussed our reconstruction removal assuming single-level KVCache. Now we generalize this to the multi-level case, with ranks $r_0 \leq \cdots \leq r_\beta$. One might conjecture the overhead will increase as we may need multiple versions of Eq. 21 per rank. However, similarly to the single-headed case, all we need is the full-rank version of Eq. 21, and the lower-rank case can be dealt with truncation.

For simplicity, suppose we have only two levels, each with rank r_0, r_β, and $\alpha = 0$. To obtain $\begin{pmatrix} S_0U_0 & S_0U_1 \\ S_1U_0 & S_1U_1 \end{pmatrix}$ as in Eq. 20, the first r_0 columns can be filled by

$$\begin{pmatrix} S_0 \\ S_1 \end{pmatrix} \cdot \text{concat}(\begin{pmatrix} U_0^{r_0} & U_1^{r_0} \end{pmatrix}, \begin{pmatrix} U_0^{r_\beta} & U_1^{r_\beta} \end{pmatrix}[:, : r_0]) \tag{22}$$

and the remaining r_β columns can be filled by

$$\begin{pmatrix} S_0 \\ S_1 \end{pmatrix}[\kappa - p\kappa :] \cdot \begin{pmatrix} U_0^{r_\beta} & U_1^{r_\beta} \end{pmatrix}[:, r_0 :] \tag{23}$$

where we use concatenation along rows, $[i :]$ means the submatrix starting from ith row, $[:, i :]$ means the submatrix starting from ith column, and $[:, : i]$ means the submatrix ending with ith column. Then, Eq. 21 can be calculated as:

$$\begin{pmatrix} S_0U_0 & S_0U_1 & S_1U_0 & S_1U_1 \end{pmatrix} \cdot \begin{pmatrix} V_{00}O_{00} & V_{00}O_{01} \\ V_{10}O_{00} & V_{10}O_{01} \\ V_{01}O_{10} & V_{01}O_{11} \\ V_{11}O_{10} & V_{11}O_{11} \end{pmatrix} \tag{24}$$

where the matrix in the right-hand side is prebuilt with full-rank SVD results. In conclusion, we do not need multiple versions of Eq. 21, and only need to truncate and concatenate the VCaches, the U matrices. This design removes the reconstruction cost (Figure 2), which is the main reason why Palu is slower even though it is smaller than the original attention head.

C Grouping Attention Heads to Reduce Prebuilt Matrix (Eq. 24)

While the above implementation reduces the latency almost by half (Figure 2), as a byproduct, we notice that the prebuilt matrix on the right-hand side of Eq. 24 can be larger as the number of heads for key and values increases, compared with $\begin{pmatrix} O_{ij} \end{pmatrix}$. For such a case, we show that we can make the prebuilt matrix small by grouping attention heads.

First, the prebuilt matrix is already small if the target LLM employs grouped query attention (GQA; Ainslie et al., 2023), which is popular in recent LLMs (Dubey et al., 2024; Hui et al., 2024; Yang et al., 2024; Team et al., 2024; Abdin et al., 2024).

To illustrate the size of the prebuilt matrix, for example, suppose we have h heads. Our target is obtaining the same output between:

$$\left(S_i\mathcal{V}_i\right) \cdot \left(O_{i,j}\right) = \left(S_{[k/h]}U_{k\%h}\right) \cdot M_{k,j} \tag{25}$$

where $0 \leq i, j < h$, $0 \leq k < h^2$, and $k\%h \equiv k \pmod h$. If we adopt the former strategy, then the prebuilt matrix M will become as follows:

$$M_{k,j} = \left(V_{k\%h,[k/h]}O_{[k/h],j}\right) \tag{26}$$

which is h times larger than the original $\left(O_{i,j}\right)$. This roots from merging $\left(V_{i,j}\right)$ from Eq. 18 into $\left(O_{i,j}\right)$. Thus, reducing the size of $\left(V_{i,j}\right)$ should be beneficial.

Now, for simplicity, assume a simple GQA, where the number of heads for query is $h = 2$ as before, and the number of heads for key and value

is 1. The size of the decomposed matrix is reduced as the number of heads for values:

$$\mathcal{V}_0 = U_0 \cdot V_{00} \tag{27}$$

With this decomposition, the original attention output is updated as follows:

$$\begin{aligned}
&\begin{pmatrix} S_0\mathcal{V}_0 & S_1\mathcal{V}_0 \end{pmatrix} \cdot \begin{pmatrix} O_{00} & O_{01} \\ O_{10} & O_{11} \end{pmatrix} = \\
&\begin{pmatrix} S_0U_0V_{00} & S_1U_0V_{00} \end{pmatrix} \cdot \begin{pmatrix} O_{00} & O_{01} \\ O_{10} & O_{11} \end{pmatrix} = \\
&\begin{pmatrix} S_0U_0 & S_1U_0 \end{pmatrix} \cdot \begin{pmatrix} V_{00}O_{00} & V_{00}O_{01} \\ V_{00}O_{10} & V_{00}O_{11} \end{pmatrix}
\end{aligned} \tag{28}$$

where the first term in the final expression can be obtained by reshaping

$$\begin{pmatrix} S_0U_0 \\ S_1U_0 \end{pmatrix} = \begin{pmatrix} S_0 \\ S_1 \end{pmatrix} \cdot U_0 \tag{29}$$

In Eq. 28, we can observe that the prebuilt matrix size is smaller than that of Eq. 24, supporting our design does not occur significant increase of model weights.

Second, if GQA is not employed, we manually group the heads to reduce the prebuilt matrix size. In the GQA case, we notice that the gist for a smaller prebuilt matrix is reducing the size of the decomposed matrix as in Eq. 27. Therefore, we propose to group the heads before decomposition.[5] Suppose we are manipulating LLM without GQA, which we group heads by g heads, and we properly reshape the given matrices. Then we decompose as:

$$\left(\mathcal{V}_{[i/g],0,i\%g}\right) = \left(U_{[i/g],0,i\%g}\right) \odot \left(V_{[i/g],i\%g,j\%g}\right) \tag{30}$$

where $0 \leq i < h$, $0 \leq j < g$, and $\odot$ is the batched matrix multiplication, which we define $A = B \odot C$ as follows:

$$A_{i,j,l} = \sum_k B_{i,j,k} \cdot C_{i,k,l} \tag{31}$$

Notice the size of $\left(V_{[i/g],i\%g,j\%g}\right)$ is $\frac{h}{g}$ times smaller than $\left(V_{i,j}\right)$ from Eq. 18. Now, the output of attention is updated as follows:

$$\begin{aligned}
&\left(S_{[i/g],0,i\%g} \textstyle\sum_j U_{[i/g],0,j} V_{[i/g],j,i\%g}\right) \\
&\qquad \odot \left(O_{[i/g],i\%g,j}\right) \\
&= \left(S_{[k/g^2],0,[k/g]\%g} U_{[k/g^2],0,k\%g}\right) \odot M_{k,j}
\end{aligned} \tag{32}$$

where $0 \leq k < gh$ and the first term in the final expression is reshaped version of

$$\left(S_{[i/g],i\%g,0}\right) \odot \left(U_{[i/g],0,i\%g}\right) \tag{33}$$

and the second term is

$$M_{k,j} = \left(V_{[k/g^2],k\%g,[k/g]\%g} O_{[k/g^2],[k/g]\%g,j}\right) \tag{34}$$

This makes $M_{k,j}$ only g times larger than $\left(O_{i,j}\right)$. The same idea can be applied to GQA design as well for further reduction. We elaborate the required modification to the forward pass in TRANSFORMERS (Wolf et al., 2020), a popular framework, in Figure 8.

[5] This is inspired by the group-head strategy of Palu (Chang et al., 2024) for a different purpose. Our distinction is grouping heads to reduce the size of the prebuilt matrix.

```
def forward(self, hidden_states, attention_mask, position_ids, past_key_value, alpha, p, r_0):
    ... # calculate key_states, query_states

    value_v_states = self.UV(hidden_states)
    keep_rows = r_0 * self.head_dim * self.head_group_size
    value_v_states = value_v_states.view(bsz, kv_len, self.num_head_groups, self.head_dim * self.head_group_size).transpose(1, 2)

    if prefill:
        value_states = self.VV(value_v_states)
        kv_seq_len = key_states.shape[-2]

        ... # cache sink tokens, last tokens

        mid_len = (kv_seq_len - alpha) * (1-p)
        mid_key_v_states = key_v_states[..., alpha:alpha+mid_len, :]
        mid_value_v_states = value_v_states[..., alpha:alpha+mid_len, : keep_rows]
        past_key_value.update(mid_key_v_states, mid_value_v_states, self.layer_idx)
        value_states = value_states.view(bsz, -1, self.num_key_value_heads, self.head_dim).transpose(1, 2)

        attn_output = flash_attention(query_states, key_states, value_states, ...)
        attn_output = attn_output.transpose(1, 2).contiguous()
        attn_output = attn_output.reshape(bsz, q_len, self.hidden_size)
        attn_output = F.linear(attn_output, self.orig_o_weight, None)

    else:
        if current_mid_len < total_len * (1 - p):
            last_key, last_value = cached_last_tokens.pop(self.layer_idx)
            past_key_value.update(last_key, last_value[..., :keep_rows], self.layer_idx)

        ... # calculate attn_weights using sink, mid, last tokens

        attn_weights = attn_weights.reshape(bsz, self.num_head_groups, self.num_key_value_groups * q_len * self.head_group_size, -1)

        attn_output = torch.matmul((attn_weights[..., -last_len:], last_value_states)
        attn_output += torch.matmul(attn_weights[..., :alpha], sink_value_states)
        attn_output[..., :keep_rows] += torch.matmul(attn_weights[..., alpha:-last_len], mid_value_states)

        attn_output = attn_output.view(bsz, self.num_head_groups * self.num_key_value_groups * self.head_group_size, q_len, self.head_group_size * self.head_dim).transpose(1, 2)
        attn_output = attn_output.reshape(bsz, q_len, self.new_o_weight_out_dim)
        attn_output = self.o_proj(attn_output) # o_proj: prebuilt matrix

    return attn_output, attn_weights, past_key_value
```

Figure 8: Modified forward pass considering TALE without quantization.