MAPLE: Memory-Aware Predict and Load for Efficient LLM Inference

Zhenyu Liu[1], **Zhemin Zhang**[1], **Zirui Zhang**[2], **Yanyuan Qin**[3], **Jiayi Luo**[4], **Zhenyu Gu**[3], **Liu Liu**[1]
[1]ECSE, Rensselaer Polytechnic Institute
[2]01.AI, [3]IN.AI, [4]University of Illinois at Urbana-Champaign
{liuz32, zhangz29, liu.liu}@rpi.edu, jiayil5@illinois.edu
{zirui.dream, guzhenyu}@gmail.com, yanyuan.qin@uconn.edu,

Abstract

Large Language Models (LLMs) perform well across various natural language processing (NLP) tasks. However, the inference for extensive text generation faces challenges due to the significant memory demands of the key-value (KV) cache, which scales with sequence length. In this paper, we introduce a novel, bandwidth-efficient method for managing the KV cache. Utilizing learning-based techniques, our method predicts and retrieves only the essential KV entries, thereby eliminating the need for comprehensive KV pair transfers. Distinct from previous approaches, our method decouples the prediction phase from the computation phases by storing low-rank Keys in HBM, drastically reducing bandwidth consumption while minimally impacting memory usage and maintaining accuracy.

1 Introduction

Large Language Models (LLMs) have demonstrated state-of-the-art performance across various natural language processing (NLP) tasks, such as question-answering (QA) [16, 31], summarization [12, 29], and code completion [22, 18], often achieving human-like levels of comprehension and generation. One of the key components in the model architecture of LLMs is Key-Value (KV) Cache [15], which stores previous keys and values in memory. This mechanism bypasses the need to recompute these KV states in future decoding steps, significantly reducing computational cost and enhancing the overall speed of the inference process.

While the KV cache is crucial for improving efficiency in LLMs, it also introduces significant challenges when processing requests with long contexts. Due to the auto-regressive nature of LLMs, generating a single token needs reading the entire KV cache. However, unlike model weights, which remain constant, the KV cache size scales with the sequence length, often consuming more memory capacity than the model weights themselves [11]. This scalability issue becomes particularly evident during inference, as depicted in Figure 1 left, as the sequence length extends, the linear increase in the size of the KV cache brings pressure to memory capacity and degrades inference performance.

Meanwhile, modern inference systems often allow offloading the KV cache to the CPU's host memory, enabling the handling of longer contexts beyond the GPU's memory capacity [20, 1]. However, CPU memory scalability is inadequate to support the expanding demands of the KV cache. For example, when conducting inference with the Llama3.1 405B model [4], at a batch size of 128 and a sequence length of 8192, the KV cache size approaches approximately 4TB, highlighting significant scalability challenges. In this scenario, Compute Express Link (CXL) memory technology [7] offers a solution by providing connections between CPUs and external memory devices like memory expanders. Nevertheless, the bandwidth limitations of the CXL memory pool can struggle to meet the demands of frequently accessing large volumes of KV cache, which may significantly increase inference latency.

Workshop on Machine Learning and Compression (NeurIPS 2024).

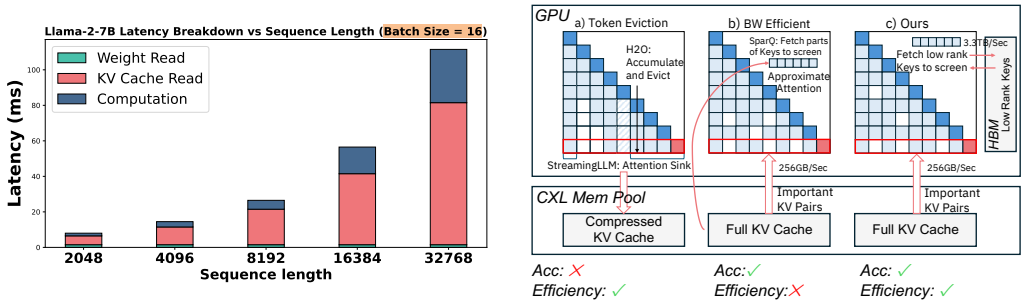

Figure 1: (Left) Breakdown of Inference Latency on Llama2-7B [25]. (Right) Illustration of three different methods to alleviate bandwidth pressure: a) Token eviction, as used in H2O [30] and StreamingLLM [27], which may lead to information loss due to permanent token removal; b) Bandwidth (BW)-efficient methods, such as SparQ [17], which perform selective retrieval of KV pairs from the KV cache and need to copy a K cache; and c) Our method, which decouples the screening and scoring processes by storing low-rank Keys in HBM, minimizing bandwidth pressure.

One strategy to mitigate this issue involves reducing the number of stored KV pairs, with token eviction as a notable technique [30, 13, 27, 6]. Although this method can decrease the KV cache capacity, it assumes a uniform attention pattern across all tokens, potentially leading to significant information loss, as depicted on the right in Figure 1 right. With the help of CXL memory, kv token eviction seems unnecessary. Alternatively, another approach keeps KV cache intact and focuses on selecting only the most critical KV pairs to reduce data transfer [17, 23], aiming to maintain accuracy. While theoretically promising, these methods frequently require reloading parts of KV pairs from the KV cache to compute approximate attention, and for SparQ [17], it also needs to store a copy of Key cache, which may not deliver the expected speed improvements in real-world applications [17].

To develop a method that is both effective and efficient, we propose **MAPLE**:**M**emory-**A**ware **P**redict and **L**oad for **E**fficient LLM Inference. We build on our basic idea that predicts and retrieves only the essential key-value (KV) entries, thereby minimizing the need for comprehensive transfers. We enhance the accuracy of these predictions through a learning-based method [26]. Furthermore, we boost our system's efficiency and effectiveness by decoupling the computation of approximate attention from memory access, significantly reducing additional memory overhead.

We conduct initial experiments using the Llama2-7B model [25] across various downstream datasets. Our algorithm consistently exhibits higher accuracy compared to baseline models. Additionally, our method demonstrates greater potential for practical implementation.

2 Related Work

This section briefly introduces two mainstream methods designed to alleviate the KV cache's capacity issues (Section 2.1) and bandwidth pressure (Section 2.2). However, these methods fail to achieve a double-win of enhancing both accuracy and bandwidth.

2.1 KV Cache Sparsity

Previous efforts have aimed to compress the KV cache size to minimize memory consumption based on attention sparsity. Based on the idea that only parts of KV pairs are important for current token generation, these methods generally discard less critical tokens based on predefined criteria. For instance, StreamingLLM [27] manages infinitely long texts by utilizing attention sinks and maintaining a finite KV cache. H2O [30] uses accumulated attention scores to determine token importance, thereby efficiently reducing cache size by eliminating less significant tokens. Similarly, Scissorhands [13] identifies key tokens that significantly influence the current and future generations. FastGen [6] implements an adaptive eviction policy across transformer layers. However, these methods permanently discard tokens, which, though currently deemed non-essential, could hold value for subsequent generations, potentially resulting in the loss of crucial information.

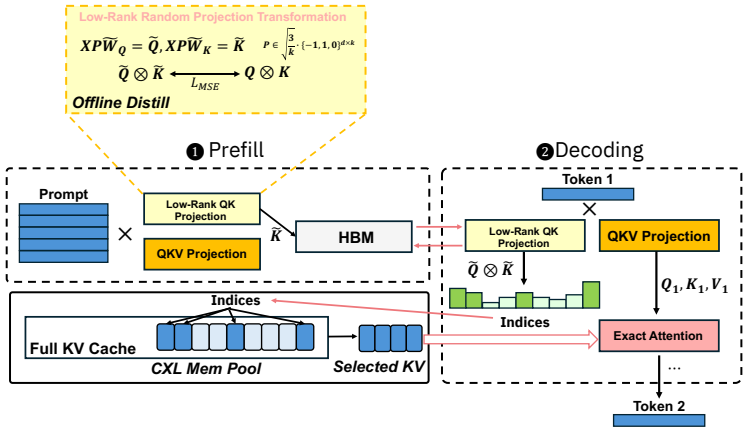

Figure 2: Overview of MAPLE. Before deploying our method, we first conduct an offline distillation for our low-rank Query and Key weights. Then, (1) During Prefill, we process the prompt for both high-rank and low-rank QK Projection, offload full KV to the CXL Mem Pool and low-rank Keys are kept in the HBM. (2) When generating token by token, our method firstly fetch low-rank Keys from HBM and following a approximate attention computation to get indices for important KV pairs. After this, send indices to CXL Mem Pool and get selected important KV pairs to do exact attention.

2.2 Bandwidth Efficient Inference

Another line of research related to ours aims to reduce data transfer at the decoding phase. These approaches typically retain the full KV cache but selectively compute with only a part of the key-value pairs at a time. SparQ [17] speeds up attention by focusing on the most important query dimensions and matching them with corresponding key dimensions to get approximate scores. It then selects the top-k keys based on these scores for full attention computation. Quest [23] retains all of the KV cache and selects part of the KV cache based on the current query. Both works need to fetch parts of KV cache to do screening, which is not friendly for KV cache offloading. A recent study Loki [21] explores the low-rank of keys of each token, the selection process is then performed in low-rank space to reduce the data transfer of the decoding process. However, this uses an online PCA method that is computationally unfriendly and does not consider system-level joint optimization.

3 Memory-Aware Predict and Load Methods

In this section, we present MAPLE, an efficient KV cache selection framework. We mainly show the two stages of our proposed solution (Section 3.1 and Section 3.2). The challenge lies in developing a prediction method that is both efficient and accurate. Our solution introduces a novel approach, illustrated in Figure 2. The core of our method lies in decoupling the prediction module from the full KV cache in CXL memory. This separation allows for more dynamic and precise access to the necessary KV pairs during the inference process.

3.1 On-Device Predict from Distilled Low-Rank Transformation

We first obtain the query matrix and key matrix in the low-dimensional space by random projection:

$$\tilde{Q} = XP\tilde{W}_Q, \quad \tilde{K} = XP\tilde{W}_K \tag{1}$$

where $\tilde{W}_Q \in \mathbb{R}^{k \times k}$ and $\tilde{W}_K \in \mathbb{R}^{k \times k}$ are trainable parameters, $P \in \sqrt{\frac{3}{k}} \cdot \{-1, 0, 1\}^{d \times k}$ is a sparse random projection matrix, in which k is much smaller than d. We then incorporate quantization to decrease the bit-width of approximation parameters to further reduce computation costs. Specifically, we apply a one-time quantization step on $\tilde{W}_Q, \tilde{W}_K$ to the INT8 fixed-point arithmetic.

When it comes to learning the approximation parameters, the trainable parameters $\tilde{W}_Q$ and $\tilde{W}_K$ are learned by minimizing the mean squared error (MSE) as the optimization objective through an offline

Algorithm 1 KV Cache Access and Exact Attention Calculation

Require: Low-rank attention scores A_{approx}, Cached key and value matrices K', V', Full query matrix Q
Ensure: Exact attention output

function TOP-K-SELECTION(A_{approx}, K', V')
  $K'_S \leftarrow \text{concat}(K', K_S)$ ▷ Concatenate current and cached keys
  $V'_S \leftarrow \text{concat}(V', V_S)$ ▷ Concatenate current and cached values
  indices $\leftarrow \text{topK}(A_{\text{approx}}, k)$ ▷ Select top-K values based on approximate attention
  $K''_S \leftarrow K'_S[\text{indices}]$ ▷ Retrieve selected keys from top-K indices
  $V''_S \leftarrow V'_S[\text{indices}]$ ▷ Retrieve selected values from top-K indices
  return K''_S, V''_S
end function
function EXACT-ATTENTION(Q, K''_S)
  $A_{\text{exact}} \leftarrow \text{softmax}\left(\frac{Q \cdot K''^T_S}{\sqrt{d}}\right)$ ▷ Compute exact attention with selected top-K keys
  return A_{exact}
end function
function ATTENTION-OUTPUT(A_{exact}, V''_S)
  output $\leftarrow A_{\text{exact}} \cdot V''_S$ ▷ Multiply attention scores with selected values to get the output
  return output
end function

distillation:

$$L_{MSE} = \frac{1}{B}||QK^T - \tilde{Q}\tilde{K}^T||_2^2 \tag{2}$$

where B is the mini-batch size. The random projection matrix P is not trainable and stays constant after initialization. The final fine-tuning process uses the following loss:

$$Loss = L_{Model} + \alpha L_{MSE} \tag{3}$$

where α is the hyper-parameters to control the training process.

3.2 Adaptive Load from CXL Memory

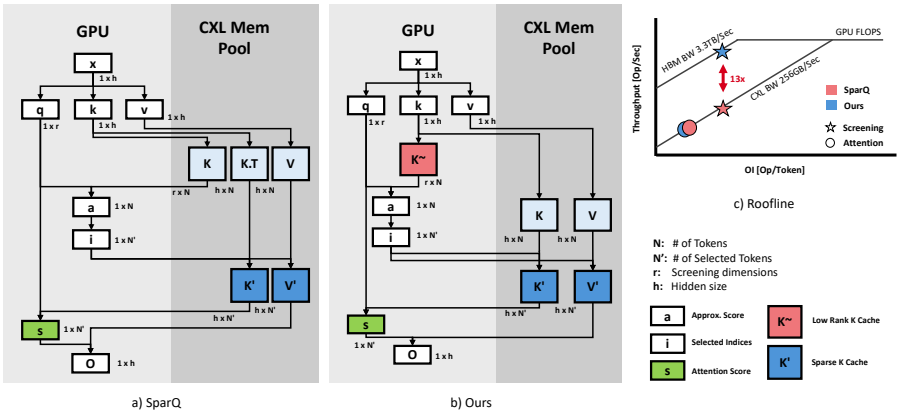

Figure 3: Dataflow illustration and roofline model. Arrows crossing the GPU and CXL memory pool boundary indicate cross-device communication. a) SparQ dataflow b) Our proposed design, with a low-rank K cache kept on GPU memory for efficient screening. c) Roofline model showing throughput improvement in the screening phase.

Here we illustrate our method in Algorithm 1 and dataflow in Figure 3 for KV cache offloading on CXL memory. SparQ retrieves r dimensions from K cache based on the highest query vector magnitudes, performs dot product scoring, and fetches selected indices. It stores two K cache copies to mitigate non-contiguous access. In contrast, our method screens faster with a smaller footprint by

keeping a low-rank K cache on GPU memory. This eliminates CXL transfer during screening and avoids additional copies due to efficient contiguous memory access.

4 Evaluation

In this section, we quantitatively evaluate the effectiveness and efficiency of our proposed method for enhancing LLM inference performance. Section 4.1 provides a brief overview of the experimental settings. Then, in Section 4.2, we evaluate our method on selected datasets from lm-harness [5] and two extra datasets of language modeling, comparing it with StreamingLLM [27] and H2O [30]. Section 4.3 is a theoretical analysis of efficiency improvements.

4.1 Methodology

Models and Tasks. In our experiments, we utilize one state-of-the-art model, Llama2-7B [25], which is known for its robust performance across various tasks. In terms of task selection, we evaluate zero-shot downstream tasks and language modeling datasets. We use four zero-shot tasks from the lm-evaluation-harness benchmark [5]: BoolQ [3], SIQA [19], Hellaswag [28] and MathQA [2]. For language modeling, we choose MMLU [8] for general natural language understanding and Lambada [14] for token prediction.

Baselines. We compare our method against three baseline approaches, each maintaining a consistent KV cache size across different layers to ensure fairness. H2O [30] and StreamingLLM [27] represent methods for efficient KV cache management through token eviction strategies. H2O [30] determines the importance of a token by accumulating historical attention score information, while StreamingLLM [27] performs token eviction by retaining only a portion of the initial token and local token. Additionally, we include a comparison with SparQ [17] in terms of efficiency, a recent approach that optimizes KV access for enhanced bandwidth efficiency.

Training Strategies. We first distill the pretrained Llama2-7B model [25] using the Alpaca dataset [24], which comprises 52,000 instruction-demonstration pairs generated by OpenAI's text-davinci-003 engine. This dataset is specifically designed for instruction tuning of language models to enhance their ability to follow instructions more accurately. The distillation process continues in three epochs, with the learning rate decreasing by a factor of ten at the end of each epoch. After distillation, we conduct the fine-tuning on this dataset for only 1 epoch to get models with different KV cache transfer budget.

4.2 Accuracy Evaluation

Figure 4 illustrates the accuracy of baselines and our method on Llama2-7B with various zero-shot tasks. The term relative KV cache size refers to the proportion of the KV cache used in the attention computation relative to a full-cache baseline. We evaluate five relative KV cache sizes: 50%, 25%, 20%, 15%, and 12.5%.

Lm-evaluation-harness. As shown in the first 4 subfigures in Figure 4. Our method can achieve better compression while maintaining a higher accuracy than permanent KV cache token eviction methods, verifying our method's effectiveness. Our method can achieve almost no accuracy loss under 25% KV cache. For the SIQA and MathQA datasets, we observe that the dataset used for distillation and fine-tuning lacks certain key features present in these datasets. As a result, our method does not perform optimally on SIQA and MathQA. In the future, we plan to identify a more suitable dataset to enhance generalization and improve performance

Language Modeling. Last two subfigures in Figure 4 are the results of MMLU [8] as well as Lambada [14]. As illustrated in the figures, our method consistently shows better accuracy across these two tasks when the relative KV cache size is less than 50%. At the same time, other baselines show a noticeable accuracy drop because some of the important tokens are evicted permanently.

4.3 Footprint and I/O Complexity Analysis

The Dense method shows a consistent but high KV Cache Footprint and Fetch I/O across both LLaMA2 models as shown in Table1. SparQ significantly reduces Fetch I/O at the cost of increased

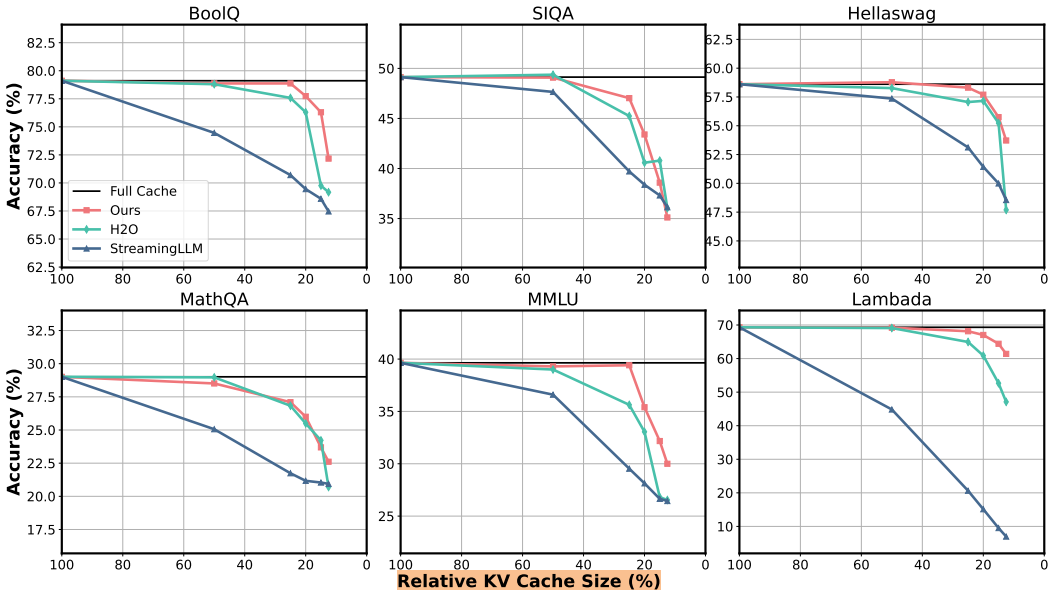

Figure 4: Accuracy of Llama2-7B on zero-shot tasks in lm-evaluation-harness, MMLU as well as Lambada.

Footprint, with the largest memory usage among all methods. Our method achieves the lowest Fetch I/O while maintaining a Footprint only slightly larger than Dense, striking an optimal balance between memory usage and data transfer efficiency for both 7B and 13B models.

Table 1: Memory Footprint and IO Complexity Comparison

Model	Method	Footprint (GB)	Overhead	Data Load (GB)	Reduction
LLaMA2-7B	Dense	512	-	512	-
	SparQ	768	+50%	132	-74.22%
	MAPLE	543	+6.05%	128	-75%
LLaMA2-13B	Dense	1000	-	1000	-
	SparQ	1500	+50%	256.25	-74.38%
	MAPLE	1062.5	+6.25%	250	-75%

5 Discussion and Conclusion

In this paper, we present a novel approach named MAPLE to manage the KV cache for large language models, effectively addressing efficiency and accuracy concerns. By utilizing a learning-based technique, our method focuses on fetching only the essential KV entries, eliminating the need for extensive KV pair transfers. Our unique strategy of separating the screening and computing phases by storing low-rank Keys in GPU device memory not only minimizes the impact on memory footprint but also preserves the accuracy of the model. We demonstrate the potential of our method to enhance both the effectiveness and efficiency of language model inference.

MAPLE employs two low-rank learnable matrices and relies on distillation to enhance its generalization. In this study, we initially apply a simple and general instruction fine-tuning dataset to test MAPLE, acknowledging that performance may vary with different datasets. In the future, we intend to test MAPLE on larger text datasets to enhance its stability and generalization. Moreover, we plan to extend MAPLE to a broader range of models in terms of size and architecture, including Llama2-13B [25], Mistral-7B [9] (featuring Group-Query Attention), and Mixtral-8x7B [10] (utilizing MoE Architecture).

References

[1] Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

[2] Amini, A., Gabriel, S., Lin, P., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

[3] Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[4] Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[5] Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 07 2024. URL `https://zenodo.org/records/12608602`.

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

[7] Gouk, D., Kwon, M., Bae, H., Lee, S., and Jung, M. Memory pooling with cxl. *IEEE Micro*, 43 (2):48–57, 2023.

[8] Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[9] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[10] Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[11] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

[12] Liu, Y. and Lapata, M. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[13] Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

[14] Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[15] Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

[16] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

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

[18] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[19] Sap, M., Rashkin, H., Chen, D., LeBras, R., and Choi, Y. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[20] Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

[21] Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024.

[22] Svyatkovskiy, A., Zhao, Y., Fu, S., and Sundaresan, N. Pythia: Ai-assisted code completion system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2727–2735, 2019.

[23] Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[24] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

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

[26] Vempala, S. S. *The random projection method*, volume 65. American Mathematical Soc., 2005.

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

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

[29] Zhang, X., Wei, F., and Zhou, M. Hibert: Document level pre-training of hierarchical bidirectional transformers for document summarization. *arXiv preprint arXiv:1905.06566*, 2019.

[30] Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[31] Zhuang, Y., Yu, Y., Wang, K., Sun, H., and Zhang, C. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36, 2024.