Position-Adaptive Temporal Sparsity for KV Caches in Long-Context LLMs

Abstract

Large Language Models (LLMs) are being deployed with context lengths of millions of tokens. Such long contexts need to allocate substantial memory capacity, and require a high memory bandwidth demand. Inference systems use a key-value (KV) cache to store keys and values across all layers, which grows linearly with context length, model size, and batch size.

In this paper, we propose Position-Adaptive Temporal Sparsity (PATS), a method for reducing the memory cost of key-value (KV) caches in large language models. PATS applies bit truncation to KV entries based on token position, preserving all tokens while reducing the effective cache size and avoiding the context loss of token eviction-based methods. PATS requires no re-training and can be integrated into existing inference systems to extend usable context length. We evaluate three *bit-truncation schedules* and demonstrate that PATS achieves up to 36% memory requirement reduction with a 0.9% perplexity increase, and up to 42% reduction with less than 5% perplexity increase. We show that even when PATS reduces the KV cache by 42%, it can help the model generalize and increase accuracy on popular reasoning benchmarks by up to 5%.

Keywords

Large Language Models, KV Cache Sparsity

1 Introduction

Large Language Models (LLMs) are increasingly powering applications from different domains. LLMs are being deployed with context lengths of millions of tokens. This progression enables applications that require reasoning over full documents, multi-session dialogue, and retrieval-augmented generation (RAG). However, such long contexts require substantial memory capacity and bandwidth. In autoregressive decoding, each new token requires attending to all previous tokens. To avoid recomputing attention over the full prefix for each new token, inference systems use a key-value (KV) cache to store keys and values across all layers. The size of this cache grows linearly with context length, model size, and batch size. Figure 1 shows the percentage of KV cache of the total memory required for different sequence lengths at a batch size of one. For instance, the KV cache for Llama-3.2-3B with a 32K context length accounts for 64% of the total used memory capacity. This grows to more than 98% with a 1M context length. Hence, even at relatively small batch sizes, inference is memory-bound [2, 5]. As a result, the KV cache size has become the primary bottleneck for scaling LLMs to longer contexts.

Various approaches have been proposed to reduce the cost of KV caches. Mainly, quantization and token eviction or sparsification. Quantization methods compress keys and values into fewer bits, which can substantially shrink the cache size. However, aggressive quantization can incur accuracy degradation [7, 11], while some approaches may require re-training, calibration, or per-layer tuning to recover lost performance [4, 8]. Sparsification or token eviction methods retain only a subset of tokens, such as maintaining the most recent window of tokens or a fixed set of "high-attention" tokens identified during inference [16, 19]. These approaches risk discard tokens that can potentially be important for generating future tokens [13].

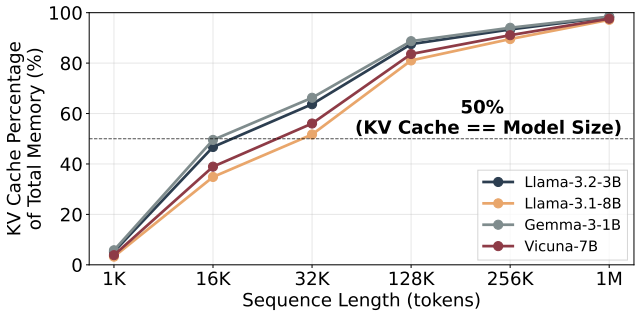

Figure 1: KV cache memory as a percentage of total memory (KV cache + model weights) for four LLMs at different sequence lengths. For all models, the KV cache exceeds model size beyond 32K tokens.

This paper proposes Position-Adaptive Temporal Sparsity (PATS), a method for reducing the effective KV cache size without requiring model-specific calibration or evicting entire tokens' data. PATS applies position-dependent truncation of low-order bits in the floating-point representation of KV entries. The truncation level is determined by a token's position in the sequence: tokens that are more critical to future token generation are preserved at higher precision, while less influential tokens are truncated more aggressively. PATS is motivated by two observations: (1) Not all tokens contribute equally to generating subsequent tokens; and (2) Floating-point values can be truncated with minimal numerical impact when least-significant mantissa bits are removed. We study three *truncation schedules* that control the distribution of truncation levels across token positions.

We evaluate PATS on four LLMs under varying levels of truncation strengths and measure the change in perplexity, memory savings, and accuracy on reasoning benchmarks. Among the three schedules, *Middle-Heavy*, which keeps windows of recent and initial tokens at high precision while truncating intermediate tokens most aggressively, achieves the best memory-accuracy tradeoff. It reduces the KV cache size by up to 42% with less than 5% perplexity increase on a subset of Wikitext2 [9], and improves accuracy on the MMLU [3] and HellaSwag [17] reasoning benchmarks by up to 5%.

In summary, this paper makes the following contributions:

MICRO 2025, Seoul, Korea
2025. ACM ISBN 978-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

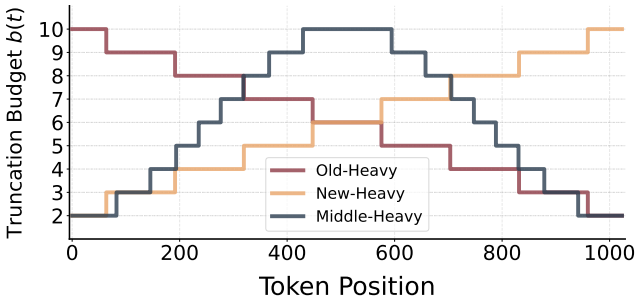

Figure 2: PATS truncation schedules for KV cache with 1K sequence length. Old-heavy truncates more bits for earlier tokens, New-heavy for recent tokens, and Middle-heavy peaks in the middle while preserving precision at both ends. For the three schedules, $b_{\min} = 2$ and $b_{\max} = 10$.

- We propose PATS, a position-adaptive approach for KV cache compression, which reduces the memory costs while preserving all tokens.
- We demonstrate PATS on four LLMs and show it achieves up to 42% KV cache reduction with less than 5% perplexity degradation, and improves the accuracy on reasoning benchmarks by up to 5%.

2 Background and Related Work

In this section, we review the functionality of KV cache in LLMs, and prior approaches for reducing its footprint with a focus on quantization and token sparsification.

2.1 KV Caches in Self-Attention

In Transformer-based [15] autoregressive LLMs, each newly generated token attends to all previous tokens. To avoid recomputing attention over the full prefix (prompt + generated tokens) at every step, inference commonly uses key-value (KV) caches. At each new time step, the model computes query Q, key K, and value V projections for the next token for each layer. K and V are appended to the KV cache, and Q attends over the cached projections. The cache size grows linearly with the sequence length T, the number of layers L, the number of heads H, the head dimension d, and the batch size b. The KV cache size is computed as $2 \cdot d \cdot H \cdot L \cdot T \cdot b$, where the factor of two accounts for both keys and values. For long contexts, storage and bandwidth requirements dominate inference. Hence, effectively compressing the KV cache is essential for supporting long contexts, even if decompression is complex.

2.2 KV Cache Compression

Recent work compresses KV caches by quantizing K and V. KIVI [22] shows a tuning-free 2-bit scheme with per-channel keys and per-token values. KVQuant [4] quantizes the KV cache into sub-four-bit precision using pre-RoPe [12] key quantization and non-uniform per-layer precision using calibration. Zhang et al. [18] demonstrate that quantizing each channel independently can lead to suboptimal performance as K/V channels have interdependent information. They propose Coupled Quantization, which jointly encodes multiple channels to leverage channel correlation. Quantizing activations (including KV Caches) may lead to a significant degradation [20] or require model re-training to maintain accuracy [8]. Furthermore, coarse-grained fixed-precision quantization does not exploit the inherent non-uniformity in token significance for next-token decoding.

Table 1: Mean square error for Llama-3.1-3B KV Caches for different schedules and max bits.

Schedule	4 Bits	6 Bits	8 Bits	10 Bits
New-Heavy	0.000	0.000	0.003	0.027
Middle-Heavy	0.000	0.000	0.003	0.032
Old-Heavy	0.000	0.000	0.003	0.030

Another direction is to sparsify KV caches by dropping insignificant tokens. Sparsification is based on the observation that not all tokens are equally important for future token generation [16]. SteamingLLM [16] computes attention over the latest N tokens plus a small set of the earliest *sink tokens* (e.g., the first few prompt tokens). The Heavy-Hitter Oracle (H_2O) [19] extends this idea by identifying and always retaining the tokens with the highest attention scores, while evicting other tokens with less influence on future token generation. While eviction-based approaches can substantially reduce the required memory by retaining only a subset of tokens, prior studies have shown that this can harm long-range reasoning by irreversibly dropping information that might become relevant later [13].

3 PATS Mechanism

Our proposal, PATS, reduces the memory demands for KV caches during inference by selectively truncating mantissa bits in proportion to the token's position in the context. The intuition behind PATS is driven by two observations. First, in self-attention, not all tokens contribute equally to the generation of the next token. In particular, the most recent token window and the earliest *sink* tokens that anchor the context have the largest density in the attention distribution [16]. Second, floating-point values can be truncated by zeroing out the least significant bits with a minimal numerical impact. Formally, for each token position $t \in \{1, ..., T\}$, PATS assigns a truncation budget $b(t) = \max(b_{\min}, \min(b_{\max}, f(t, T)))$ denoting the number of least-significant mantissa bits to truncate in both K and V entries for token at position t, where f is a PATS *truncation schedule* function and $b_{\min}$ and $b_{\max}$ are the minimum and maximum number of bits to truncate at any position. K and V are stored as per-head tensors of shape $[T, d]$ such that each row (token position t) is truncated to a fixed bit width $w = \text{dtype\_size} - b(t)$. Each row is packed into 64-byte blocks, with $B = \lfloor 512/w \rfloor$ elements per block, and consecutive rows are grouped into regions, as illustrated in Figure 2. Fetching an element $[t, c]$ then corresponds to a few integer operations to compute the block $b_{id} = \lfloor c/B \rfloor$, and the bit offset $(c \bmod B) \cdot w$, followed by a memory load and a shift-and-mask operation to recover the value. We study three PATS schedules and show their truncation characteristics in Figure 2. *Old-Heavy* truncates older tokens more aggressively in a linear manner. *New-Heavy* linearly truncates more bits from recent tokens, while keeping initial sink tokens at higher precision. Finally,

Table 2: Perplexity (PPL) and memory (MEM) impacts using the three PATS schedules. Across models and truncated bits, the Middle-Heavy schedule achieves the best tradeoff between perplexity and memory requirement reduction. For 8-bit maximum truncation, Middle-Heavy achieves the lowest increase in perplexity for all models.

Model	Schedule	Baseline PPL	4 bits		6 bits		8 bits		10 bits	
			PPL %	MEM %	PPL %	MEM %	PPL %	MEM %	PPL %	MEM %
Gemma-3-1B	New-Heavy	34.84	**-0.1**	21.9	**-0.3**	29.2	+0.4	35.9	**+2.9**	42.4
	Old-Heavy		0.0	21.9	0.0	29.2	+1.7	35.9	+14.7	42.4
	Middle-Heavy		+0.1	21.8	-0.2	29.0	**+0.2**	35.9	+5.9	42.3
Llama-3.1-8B	New-Heavy	7.39	0.0	21.9	+0.4	29.2	+2.1	35.9	+24.0	42.4
	Old-Heavy		+0.1	21.9	+0.4	29.2	+3.1	35.9	+20.6	42.4
	Middle-Heavy		**0.0**	21.8	**+0.2**	29.0	**+0.9**	35.9	**+4.9**	42.3
Llama-3.2-3B	New-Heavy	11.48	**-0.1**	21.9	0.0	29.2	+1.2	35.9	+17.9	42.4
	Old-Heavy		0.0	21.9	+0.6	29.2	+3.6	35.9	+15.7	42.4
	Middle-Heavy		0.0	21.8	**0.0**	29.0	**+0.5**	35.9	**+1.8**	42.3
Vicuna-7B-v1.5	New-Heavy	7.75	-0.2	21.9	-0.2	29.2	+0.9	35.9	+14.2	42.4
	Old-Heavy		-0.1	21.9	-0.3	29.2	+0.1	35.9	+11.4	42.4
	Middle-Heavy		**-0.1**	21.8	**-0.4**	29.0	**-1.2**	35.9	**+0.2**	42.3

Middle-Heavy gradually increases the truncation toward the middle of the context, and decreases toward both ends. All three schedules are static and only depend on a token's position in the sequence.

PATS preserves more tokens in the cache, avoiding the hard eviction strategies used by prior work. Compared to window attention or sink-token schemes, PATS employs a soft form of sparsity, allowing more KV cache entries to fit in the same memory. This operation does not require modifying the LLM architecture, re-training the model, or model-specific calibration. Furthermore, PATS is orthogonal to other compression approaches and can integrate for complementary results.

4 Evaluation

We evaluate PATS on four LLMs: Llama-3.1-8B, Llama-3.2-3B [1], Vicuna-7B-v1.5 [21], and Gemma-3-1B [14]. We evaluate PATS and measure its effect on memory requirements, perplexity on 512 documents from the Wikitext2 dataset [9], and accuracy on 5000 questions and cases from the MMLU [3] and HellaSwag [17] benchmarks. In all perplexity-measuring experiments, we lock the value of b_{min} to two bits.

To quantify the numerical effect introduced by PATS truncation, we measure the mean square error (MSE) between the original and truncated K and V tensors. On Llama-3.2-3B, MSE remains very small (≈ 0.03) for the most aggressive truncation level, which truncates a maximum of 10 bits, as shown in Table 1.

Table 2 demonstrates the memory savings and the change in perplexity. Both New-Heavy and Old-Heavy schedules experience a substantial increase in perplexity at higher truncation levels. For instance, the perplexity of Llama-3.1-8B increases by 24% for New-Heavy with a maximum truncated bits of 10. Across all models, truncating 8 bits with the Middle-Heavy schedule achieves 35.9% memory requirement reduction while incurring a maximum of 0.9% perplexity increase in Llama-3.1-8B. Increasing the maximum bits to 10 increases the memory savings to more than 42%, allowing for nearly doubling the supported sequence length on the same available memory. Figure 3 shows the best achieved perplexity across all models, schedules, and bit-truncation levels.

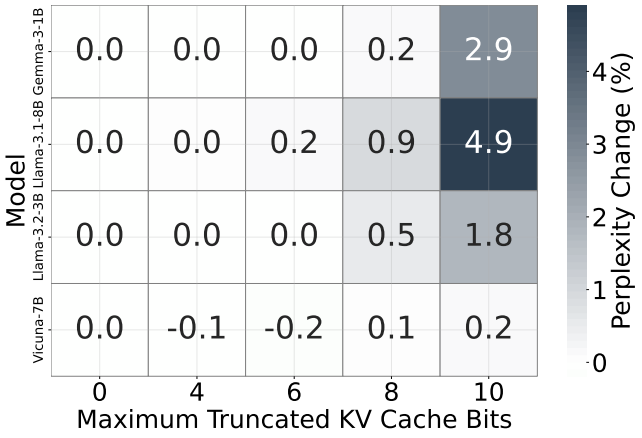

Figure 3: Perplexity change across different maximum truncation levels. Reported values correspond to the best-performing schedule at each truncation level.

Next, we evaluate the impact of PATS Middle-Heavy schedule on reasoning benchmarks. We observe a general trend of increasing accuracy on both benchmarks for most models. For MMLU, the accuracy improves across all four models, with a maximum of 5.1% increase for Gemma-3-1B. HellaSwag shows a similar trend, with three out of the four models exhibiting an increase in accuracy, while Vicuna-7B-v1.5 loses a minimal accuracy of 0.4%. This shows that PATS work may regularize and help the model generalize by truncating but not dropping less critical tokens in the context. Tables 3, 4 include the accuracy of using different truncation levels of the Middle-Heavy PATS schedule on MMLU and HellaSwag, respectively.

Table 3: Accuracy (0-shot) on 5000 questions from the MMLU benchmark for the *Middle-Heavy* PATS schedule. PATS achieves better accuracy in all four models.

Model	Baseline	8 Bits	10 Bits
Llama-3.2-3B	48.2%	50.2%	**48.4%**
Llama-3.1-8B	59.8%	**60.8%**	59.6%
Vicuna-7B-v1.5	45.3%	45.0%	**47.3%**
Gemma-3-1B	39.4%	42.1%	**41.4%**

Table 4: Accuracy (0-shot) on 5000 cases from the HellaSwag benchmark for the *Middle-Heavy* PATS schedule. PATS with a maximum of 10 truncated bits achieves better accuracy in two out of the four models.

Model	Baseline	8 Bits	10 Bits
Llama-3.2-3B	68.8%	**69.4%**	67.1%
Llama-3.1-8B	75.1%	75.1%	**77.0%**
Vicuna-7B-v1.5	**71.3%**	70.2%	71.0%
Gemma-3-1B	50.5%	**55.5%**	51.5%

5 Future Work

While PATS demonstrates that position-dependent token compression can substantially reduce the KV cache size without adversely affecting the model's accuracy, various future directions remain to be explored. First, we only discuss *static* schedules that use predetermined truncation distributions to simplify computation. However, conditioning the schedule on dynamic attention information or downstream loss may further improve PATS accuracy. Second, PATS is orthogonal to other quantization and eviction works, and combining them can result in complementary savings while improving the accuracy. Finally, integrating PATS into encoder-decoder architectures [6, 10] would cover the case where both encoder and decoder maintain KV caches, and where cross-attention amplifies the memory and bandwidth demands.

6 Conclusion

We introduced PATS, a method for alleviating the memory bottleneck caused by the KV cache LLM inference by position-aware bit truncation. PATS preserves more tokens given the same available memory, avoids re-training, and integrates into existing inference systems without modifications. We studied three bit-truncation schedules and showed that PATS can reduce cache size by up to 42% with a maximum increase in perplexity of less than 5%. Beyond memory savings, our findings suggest that position-aware truncation can help models generalize and improve the accuracy on popular reasoning benchmarks by up to 5%.

References

[1] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, and Amy Yang *et al.* 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[2] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. 2024. AI and Memory Wall. arXiv:2403.14123 [cs.LG] https://arxiv.org/abs/2403.14123

[3] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. arXiv:2009.03300 [cs.CY] https://arxiv.org/abs/2009.03300

[4] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2025. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. arXiv:2401.18079 [cs.LG] https://arxiv.org/abs/2401.18079

[5] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. 2024. SqueezeLLM: Dense-and-Sparse Quantization. arXiv:2306.07629 [cs.CL] https://arxiv.org/abs/2306.07629

[6] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. arXiv:1910.13461 [cs.CL] https://arxiv.org/abs/1910.13461

[7] Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. 2023. LLM-FP4: 4-Bit Floating-Point Quantized Transformers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 592–605. https://doi.org/10.18653/v1/2023.emnlp-main.39

[8] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models. arXiv:2305.17888 [cs.CL] https://arxiv.org/abs/2305.17888

[9] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683 [cs.LG] https://arxiv.org/abs/1910.10683

[11] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. arXiv:2308.13137 [cs.LG] https://arxiv.org/abs/2308.13137

[12] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. RoFormer: Enhanced Transformer with Rotary Position Embedding. arXiv:2104.09864 [cs.CL] https://arxiv.org/abs/2104.09864

[13] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2024. RazorAttention: Efficient KV Cache Compression Through Retrieval Heads. arXiv:2407.15891 [cs.LG] https://arxiv.org/abs/2407.15891

[14] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, and Alexandre Ramé *et al.* 2025. Gemma 3 Technical Report. arXiv:2503.19786 [cs.CL] https://arxiv.org/abs/2503.19786

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

[16] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453 [cs.CL] https://arxiv.org/abs/2309.17453

[17] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? arXiv:1905.07830 [cs.CL] https://arxiv.org/abs/1905.07830

[18] Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024. KV Cache is 1 Bit Per Channel: Efficient Large Language Model Inference with Coupled Quantization. arXiv:2405.03917 [cs.LG] https://arxiv.org/abs/2405.03917

[19] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. arXiv:2306.14048 [cs.LG] https://arxiv.org/abs/2306.14048

[20] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit Quantization for Efficient and Accurate LLM Serving. arXiv:2310.19102 [cs.LG] https://arxiv.org/abs/2310.19102

[21] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL] https://arxiv.org/abs/2306.05685

[22] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2023. KIVI : Plug-and-play 2bit KV Cache Quantization with Streaming Asymmetric Quantization. (2023). https://doi.org/10.13140/RG.2.2.28167.37282