QJL: 1-BIT QUANTIZED JL TRANSFORM FOR KV CACHE QUANTIZATION WITH ZERO OVERHEAD

Amir Zandieh[†], **Majid Daliri**[*], **Insu Han**[‡]
[†] Google Research [*] New York University [‡] KAIST

ABSTRACT

Serving LLMs requires substantial memory due to the storage requirements of Key-Value (KV) embeddings in the KV cache, which grows with sequence length. An effective approach to compress KV cache is quantization. However, traditional quantization methods face significant memory overhead due to the need to store quantization constants (at least a zero point and a scale) in full precision per data block. Depending on the block size, this overhead can add 1 or 2 bits per quantized number. We introduce QJL, a new quantization approach that consists of a Johnson-Lindenstrauss (JL) transform followed by sign-bit quantization. In contrast to existing methods, QJL eliminates memory overheads by removing the need for storing quantization constants. We propose an asymmetric estimator for the inner product of two vectors and demonstrate that applying QJL to one vector and a standard JL transform without quantization to the other provides an unbiased estimator with minimal distortion. We have developed an efficient implementation of the QJL sketch and its corresponding inner product estimator, incorporating a lightweight CUDA kernel for optimized computation. When applied across various LLMs and NLP tasks to quantize the KV cache to only 3 bits, QJL demonstrates a more than fivefold reduction in KV cache memory usage without compromising accuracy, all while achieving faster runtime.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable success across various domains, from chatbots Achiam et al. (2023); Antropic (2024) to text-to-image Ramesh et al. (2022); FireFly (2023); Midjourney (2022), text-to-video synthesis OpenAI (2024b), coding assistants Copilot (2023), and multimodal tasks OpenAI (2024a). Their core architecture, Transformers with self-attention Vaswani et al. (2017), scales with model size Kaplan et al. (2020), leading to significant memory challenges.

Auto-regressive decoding is costly due to the need to store all key-value (KV) embeddings, creating memory and latency bottlenecks. Current solutions include reducing KV heads Shazeer (2019); Ainslie et al. (2023), pruning Zhang et al. (2024b); Liu et al. (2024a), offloading Sheng et al. (2023), and memory paging Kwon et al. (2023). Quantization of KV caches has also been explored Yue et al. (2024); Yang et al. (2024); Dong et al. (2024); Kang et al. (2024); Zhang et al. (2024a), with recent per-channel methods Liu et al. (2024b); Hooper et al. (2024) improving efficiency but introducing memory overhead.

We propose **QJL**, a novel, data-oblivious quantization method leveraging **Johnson-Lindenstrauss (JL) transforms** Dasgupta & Gupta (2003). By applying a random Gaussian projection to key embeddings and quantizing them to a single bit (sign bit), while maintaining the same transformation on queries, we achieve an **unbiased** and **low-distortion** inner product estimator (Lemma 3.2, Lemma 3.3). Our method eliminates storage overhead from quantization constants, requires no tuning, and scales logarithmically with context length (Theorem 3.4).

For value embeddings, we adopt standard token-wise quantization Liu et al. (2024b); Hooper et al. (2024). We also address outliers in deeper LLM layers by applying separate quantization for extreme values (Figure 1 in Appendix D).

Our CUDA-optimized QJL method compresses the KV cache in Llama-2 Touvron et al. (2023) and long-context models Li et al. (2023), reducing memory usage **5×** with **no accuracy loss** at just **3**

bits per FPN. It surpasses recent KV quantization methods in F1 scores on long-context tasks from LongBench Bai et al. (2023) while minimizing overhead.

2 PRELIMINARIES: TOKEN GENERATION IN ATTENTION

Deploying auto-regressive language models for inference requires caching key and value embeddings at each transformer layer to eliminate redundant computations. The model sequentially updates the KV cache to generate tokens one at a time.

At each generation phase i, let $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i \in \mathbb{R}^d$ be the query, key, and value embeddings, respectively. Given n total tokens, the attention output at phase n is:

$$\boldsymbol{o}_n = \sum_{i \in [n]} \texttt{softmax}\left([\langle \boldsymbol{q}_n, \boldsymbol{k}_1 \rangle, \ldots, \langle \boldsymbol{q}_n, \boldsymbol{k}_n \rangle]\right)_i \cdot \boldsymbol{v}_i. \tag{1}$$

The output $\boldsymbol{o}_n$ is used to compute $\boldsymbol{q}_{n+1}, \boldsymbol{k}_{n+1}, \boldsymbol{v}_{n+1}$ unless generation terminates. Storing all previous $\{\boldsymbol{k}_i, \boldsymbol{v}_i\}_{i \in [n]}$ in full precision demands high memory for long contexts. Computing equation 1 requires $O(nd)$ operations due to n inner products. Additionally, large KV cache sizes slow inference as they must be repeatedly loaded from GPU memory, leading to low arithmetic intensity and GPU underutilization Pope et al. (2023). This work compresses the KV cache via token quantization, reducing memory usage while maintaining efficiency.

3 QUANTIZED JOHNSON-LINDENSTRAUSS (QJL) TRANSFORM

Our goal is to reduce KV cache memory while preserving the inner product between query and key embeddings. To achieve this, we first apply a Johnson-Lindenstrauss (JL) transform Johnson et al. (1986), a random projection that maintains inner products, followed by quantization. Specifically, we use a 1-bit Johnson-Lindenstrauss transform, which projects embeddings onto a random subspace using a Gaussian matrix and then quantizes the result to a single sign bit. This yields an unbiased, low-distortion inner product estimator Dasgupta & Gupta (2003).

The Quantized Johnson-Lindenstrauss (QJL) transformation, along with its inner product estimator, is formally defined as follows:

Definition 3.1 (QJL and inner product estimator)**.** For integers d, m, let $\boldsymbol{S} \in \mathbb{R}^{m \times d}$ be a JL transform matrix with i.i.d. standard normal entries. The QJL mapping $\mathcal{H}_S : \mathbb{R}^d \to \{-1, +1\}^m$ is:

$$\mathcal{H}_S(\boldsymbol{k}) := \texttt{sign}(\boldsymbol{S}\boldsymbol{k}), \quad \forall \boldsymbol{k} \in \mathbb{R}^d. \tag{2}$$

The inner product estimator for $\boldsymbol{q}, \boldsymbol{k} \in \mathbb{R}^d$ is:

$$\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) := \frac{\sqrt{\pi/2}}{m} \cdot \|\boldsymbol{k}\|_2 \cdot \langle \boldsymbol{S}\boldsymbol{q}, \mathcal{H}_S(\boldsymbol{k}) \rangle. \tag{3}$$

Unlike applying QJL to both vectors, which estimates only their angle Charikar (2002), asymmetric quantization preserves an unbiased inner product estimator. We formally establish this in the following results:

Lemma 3.2 (Unbiasedness of $\texttt{Prod}_{\texttt{QJL}}$)**.** *For any $\boldsymbol{q}, \boldsymbol{k} \in \mathbb{R}^d$, the expectation of $\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})$ in Equation* (3) *satisfies:*

$$\mathop{\mathbb{E}}_{\boldsymbol{S}}[\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})] = \langle \boldsymbol{q}, \boldsymbol{k} \rangle,$$

where the expectation is over the randomness of $\boldsymbol{S}$ in Definition 3.1.

Furthermore, $\texttt{Prod}_{\texttt{QJL}}$ retains bounded distortion with high probability:

Lemma 3.3 (Distortion of $\texttt{Prod}_{\texttt{QJL}}$)**.** *For any $\boldsymbol{q}, \boldsymbol{k} \in \mathbb{R}^d$, if $\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})$ is defined as in Equation* (3) *for $m \geq \frac{4}{3} \cdot \frac{1+\varepsilon}{\varepsilon^2} \log \frac{2}{\delta}$, then:*

$$\Pr_{\boldsymbol{S}}\left[|\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) - \langle \boldsymbol{q}, \boldsymbol{k} \rangle| > \varepsilon \|\boldsymbol{q}\|_2 \|\boldsymbol{k}\|_2\right] \leq \delta.$$

The proof is in Appendix A. Notably, Lemma 3.3 shows that QJL achieves lower distortion constants than the standard JL transform, indicating that quantizing one vector to a single sign bit preserves accuracy. These properties enable a robust KV cache quantizer.

Algorithm 1 QJL Key Cache Quantizer

Input: Stream of key tokens $\boldsymbol{k}_1, \boldsymbol{k}_2, \ldots \in \mathbb{R}^d$, integer m

1: Draw a random sketch $\boldsymbol{S} \in \mathbb{R}^{m \times d}$ with i.i.d. entries $\boldsymbol{S}_{i,j} \sim \mathcal{N}(0,1)$ as per Definition 3.1
2: **repeat**
3: Compute $\tilde{\boldsymbol{k}}_i \leftarrow \texttt{sign}\left(\boldsymbol{S}\boldsymbol{k}_i\right)$ and $\nu_i \leftarrow \|\boldsymbol{k}_i\|_2$
4: **store** the quantized vector $\tilde{k}_i$ and the key norm ν_i in the cache
5: **until** token stream ends

Procedure ESTIMATESCORES($\boldsymbol{q}_n$)

6: Compute inner product estimators $\widetilde{\mathbf{qK}}(j) \leftarrow \frac{\sqrt{\pi/2}}{m} \cdot \nu_i \cdot \langle \boldsymbol{S}\boldsymbol{q}_n, \tilde{\boldsymbol{k}}_j \rangle$ for every $j \in [n]$
7: $\widetilde{\texttt{Score}} \leftarrow \texttt{softmax}\left(\widetilde{\mathbf{qK}}\right)$

return $\widetilde{\texttt{Score}}$

3.1 KEY CACHE QUANTIZATION VIA QJL

The key cache is used in the computation of attention scores as shown in Equation (1). To calculate these scores, we need to compute the inner products of the current query embedding with all key embeddings in the cache. We design a quantization scheme that allows for a low-distortion estimate of the inner products between an arbitrary query and all keys in the cache. In this section, we develop a practical algorithm with provable guarantees based on QJL and the inner product estimator defined in Definition 3.1.

The quantization scheme presented in Algorithm 1 applies QJL, defined in Definition 3.1, to each key embedding, mapping them to binary vectors and storing the results in the key cache. We show in the following theorem that the attention scores calculated by Algorithm 1 have very small $(1 \pm \varepsilon)$ relative distortion with high probability:

Theorem 3.4 (Distortion bound on QJL key cache quantizer)**.** *For any sequence of key tokens $\boldsymbol{k}_1, \ldots \boldsymbol{k}_n \in \mathbf{R}^d$ and any integer m, Algorithm 1 stores binary vectors $\tilde{\boldsymbol{k}}_1, \ldots \tilde{\boldsymbol{k}}_n \in \{-1, +1\}^m$ along with scalar values $\nu_1, \ldots \nu_n$ in the cache. If the key embeddings have bounded norm $\max_{i \in [n]} \|\boldsymbol{k}_i\|_2 \leq r$ and $m \geq 2r^2\varepsilon^{-2} \log n$, then for any query embedding $\boldsymbol{q}_n \in \mathbf{R}^d$ with bounded norm $\|\boldsymbol{q}_n\|_2 \leq r$ the output of the procedure* ESTIMATESCORES*($\boldsymbol{q}_n$) satisfies the following with probability $1 - \frac{1}{\texttt{poly}(n)}$ sinultaneously for all $i \in [n]$:*

$$\left| \widetilde{\texttt{Score}}(i) - \texttt{Score}(i) \right| \leq 3\varepsilon \cdot \texttt{Score}(i),$$

where Score *is the vector of attention scores defined in Equation* (1)*.*

The proof is provided in Appendix A. This theorem shows that if the query and key embeddings have constant norms, as is common in practical scenarios, we can quantize each key embedding such that only $m \approx \varepsilon^{-2} \log n$ bits are needed to store each key token. This is independent of the embedding dimension of the tokens and scales only logarithmically with the sequence length.

3.2 VALUE CACHE QUANTIZATION

We quantize the value cache using a standard quantization method, i.e., normalizing each token's entries and then rounding each entry to a few-bit integer representation. This approach aligns with prior work, which has shown that standard token-wise quantization is highly effective for the value cache and results in a minimal accuracy drop Liu et al. (2024b); Hooper et al. (2024).

4 EXPERIMENTS

We validate the empirical performance of our algorithm through experiments on a single A100 GPU (80GB). Our implementation includes two primary CUDA kernels: one for quantizing embedding vectors using bfloat16, FP16, and FP32, and another for computing inner products between an arbitrary vector and all quantized vectors in the cache. The algorithm's wrapper is implemented in PyTorch for ease of integration, with a future CUDA implementation planned for further acceleration.

Methods	Bits	Datasets from LongBench Bai et al. (2023)					
		NarrativeQA	Qasper	MultiQA-en	MultifQA-zh	HotpotQA	2WikiMultiQA
FP16 (baseline)	16	20.79	29.42	42.83	34.33	33.05	24.14
KIVI Liu et al. (2024b)	3	**20.96**	**29.01**	40.93	**34.75**	32.79	23.01
QJL (ours)	3	20.67	28.48	**40.94**	29.71	**35.62**	**23.60**
KVQuant Hooper et al. (2024)	4.3	20.14	28.77	**44.22**	**34.44**	34.06	**23.05**
QJL (ours)	4.3	**20.72**	**30.02**	41.18	31.73	**34.22**	22.63
KIVI Liu et al. (2024b)	5	20.49	28.90	**43.24**	**34.66**	33.07	**24.86**
QJL (ours)	5	**21.09**	**29.11**	41.58	31.86	**35.65**	24.61

Table 1: Evaluation of various quantization methods and different bits per floating-point number (FPN) on long-context question-answering datasets from LongBench (F1 scores).

As shown in Theorem 3.4, attention score distortion is proportional to embedding norms. To mitigate this, we identify and isolate outlier channels, which significantly contribute to key embedding norms. These outliers are quantized separately using an independent QJL instance with a lower compression rate, allocating more bits for accurate representation.

4.1 End-to-End Text Generation

We benchmark our method on LongBench Bai et al. (2023), a suite for evaluating long-context tasks. The base model is `longchat-7b-v1.5-32k` Li et al. (2023), a fine-tuned Llama-2 with a 16,384-token context length. We compare the following quantization methods: KIVI Liu et al. (2024b), KVQuant Yue et al. (2024), and our QJL-based approach. Each floating-point number (FPN) in the base model uses 16 bits. We configure KIVI and QJL to match a 3-bit per FPN setup, while KVQuant defaults to 4.3 bits per FPN.

To assess the quality of these quantized models, we evaluate them on six QA datasets from LongBench, setting a maximum sequence length of 31,500. The evaluation follows the methodology of the original repository. Table 1 summarizes the results, showing that QJL achieves the highest F1 scores among quantization methods for `NarrativeQA`, `Qasper`, and `2WikiMultiQA`.

Models	Methods	Bits	Datasets from LM-eval Gao et al. (2023)				
			Lambada-OpenAI	HellaSwag	PIQA	MathQA	MMLU
Llama-2-7B	FP16 (baseline)	16	73.90	57.18	78.07	28.11	41.85
	KIVI Liu et al. (2024b)	3	73.88	57.13	78.07	28.11	41.81
	QJL (ours)	3	73.88	57.14	78.07	28.17	41.78
Llama-3-8B	BF16 (baseline)	16	75.59	60.17	79.65	40.64	62.09
	QJL (ours)	3	75.61	60.13	79.87	40.60	62.12

Table 2: Evaluation (accuracy) of various quantization methods on regular length datasets from LM-eval Gao et al. (2023). These comparisons are not typically based on long-context length; however, even in these cases, our QJL with 3 bits per FPN performs comparably to the baseline with 16 bits per FPN.

Experiments with Llama3 and Llama2 models. We further evaluate our method on `Lambada-OpenAI`, `HellaSwag`, `PIQA`, `MathQA`, and `MMLU`, which have shorter sequence lengths. Benchmarking is conducted using the LM-eval framework Gao et al. (2023) to ensure thorough evaluation across various metrics. We assess quantization methods based on accuracy on Llama-2-7B Touvron et al. (2023) and Llama-3-8B Llama3 (2024). Notably, KIVI supports only half-precision floating point, preventing its use on Llama-3, whereas our method is compatible with any precision format.

QJL achieves an 81% memory reduction by using 3 bits per FPN compared to the 16-bit baseline, without significant performance loss. As shown in Table 2, QJL on Llama-3-8B achieves comparable or slightly superior accuracy to the baseline across all datasets.

REFERENCES

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

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

Antropic. claude, 2024. https://www.anthropic.com/news/claude-3-family.

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. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Stéphane Boucheron, Gábor Lugosi, and Olivier Bousquet. Concentration inequalities. In *Summer school on machine learning*, pp. 208–240. Springer, 2003.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388, 2002.

Microsoft Copilot, 2023. https://github.com/features/copilot.

Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*, 2024.

Adobe FireFly, 2023. https://firefly.adobe.com/.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023. URL https://zenodo.org/records/10256836. https://github.com/EleutherAI/lm-evaluation-harness.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. *arXiv preprint arXiv:2401.18079*, 2024.

Jianqiu Ji, Jianmin Li, Shuicheng Yan, Bo Zhang, and Qi Tian. Super-bit locality-sensitive hashing. *Advances in neural information processing systems*, 25, 2012.

William B Johnson, Joram Lindenstrauss, and Gideon Schechtman. Extensions of lipschitz maps into banach spaces. *Israel Journal of Mathematics*, 54(2):129–138, 1986.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can open-source llms truly promise on context length?, 2023. URL https://lmsys.org/blog/2023-06-29-longchat. https://huggingface.co/lmsys/longchat-7b-v1.5-32k.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024a.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024b.

Llama3, 2024. https://github.com/meta-llama/llama3.

Midjourney, 2022. https://www.midjourney.com/home.

OpenAI. Introducing gpt-4o, 2024a. https://openai.com/index/hello-gpt-4o/.

OpenAI. Sora: Creating video from text, 2024b. https://openai.com/index/sora/.

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

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024.

Felix Xinnan X Yu, Ananda Theertha Suresh, Krzysztof M Choromanski, Daniel N Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. *Advances in neural information processing systems*, 29, 2016.

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*, 2024a.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024b.

A SUPPORTING PROOFS

Lemma 3.2 (Unbiasedness of $\texttt{Prod}_{\texttt{QJL}}$)**.** *For any $\boldsymbol{q}, \boldsymbol{k} \in \mathbb{R}^d$, the expectation of $\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})$ in Equation* (3) *satisfies:*

$$\mathop{\mathbb{E}}_{\boldsymbol{S}}[\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})] = \langle \boldsymbol{q}, \boldsymbol{k} \rangle,$$

where the expectation is over the randomness of $\boldsymbol{S}$ in Definition 3.1.

Proof. Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots \boldsymbol{s}_m$ denote the rows of the JL matrix $\boldsymbol{S}$. Additionally, let us decompose $\boldsymbol{q}$ to its projection onto the vector $\boldsymbol{k}$ and its orthogonal component, i.e., $\boldsymbol{q}^{\perp k} := \boldsymbol{q} - \frac{\langle \boldsymbol{q}, \boldsymbol{k} \rangle}{\|\boldsymbol{k}\|_2^2} \cdot \boldsymbol{k}$. We can write,

$$\begin{aligned}
\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) &= \frac{\sqrt{\pi/2}}{m} \sum_{i \in [m]} \|\boldsymbol{k}\|_2 \cdot \boldsymbol{s}_i^\top \boldsymbol{q} \cdot \texttt{sign}(\boldsymbol{s}_i^\top \boldsymbol{k}) \\
&= \frac{\sqrt{\pi/2}}{m} \sum_{i \in [m]} \frac{\langle \boldsymbol{q}, \boldsymbol{k} \rangle}{\|\boldsymbol{k}\|_2} \cdot \boldsymbol{s}_i^\top \boldsymbol{k} \cdot \texttt{sign}(\boldsymbol{s}_i^\top \boldsymbol{k}) + \|\boldsymbol{k}\|_2 \cdot \boldsymbol{s}_i^\top \boldsymbol{q}^{\perp k} \cdot \texttt{sign}(\boldsymbol{s}_i^\top \boldsymbol{k}) \\
&= \frac{\sqrt{\pi/2}}{m} \sum_{i \in [m]} \frac{\langle \boldsymbol{q}, \boldsymbol{k} \rangle}{\|\boldsymbol{k}\|_2} \cdot |\boldsymbol{s}_i^\top \boldsymbol{k}| + \|\boldsymbol{k}\|_2 \cdot \boldsymbol{s}_i^\top \boldsymbol{q}^{\perp k} \cdot \texttt{sign}(\boldsymbol{s}_i^\top \boldsymbol{k}).
\end{aligned}$$

Since $\boldsymbol{s}_i$'s have identical distributions, we have:

$$\mathop{\mathbb{E}}_{\boldsymbol{S}}[\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})] = \sqrt{\pi/2} \left(\frac{\langle \boldsymbol{q}, \boldsymbol{k} \rangle}{\|\boldsymbol{k}\|_2} \cdot \mathbb{E}\left[|\boldsymbol{s}_1^\top \boldsymbol{k}| \right] + \|\boldsymbol{k}\|_2 \cdot \mathbb{E}\left[\boldsymbol{s}_1^\top \boldsymbol{q}^{\perp k} \cdot \texttt{sign}(\boldsymbol{s}_1^\top \boldsymbol{k}) \right] \right).$$

To calculate the above expectation let us define variables $x := \boldsymbol{s}_1^\top \boldsymbol{k}$ and $y := \boldsymbol{s}_1^\top \boldsymbol{q}^{\perp k}$. Note that x and y are both zero-mean Gaussian random variables and because $\langle \boldsymbol{q}^{\perp k}, \boldsymbol{k} \rangle = 0$. By the following Fact A.1, x and y are independent.

Fact A.1. If $\boldsymbol{x} \in \mathbb{R}^d$ is a vector of i.i.d. zero-mean normal entries with variance σ^2 and $A \in \mathbb{R}^{m \times d}$ is a matrix, then $\boldsymbol{A} \cdot \boldsymbol{x}$ is a normal random variable with mean zero and covariance matrix $\sigma^2 \cdot \boldsymbol{A}\boldsymbol{A}^\top$.

This implies that the second expectation term above is zero because $\mathbb{E}\left[\boldsymbol{s}_1^\top \boldsymbol{q}^{\perp k} \cdot \texttt{sign}(\boldsymbol{s}_1^\top \boldsymbol{k}) \right] = \mathbb{E}[y \cdot \texttt{sign}(x)] = \mathbb{E}[y] \cdot \mathbb{E}[\texttt{sign}(x)] = 0$. Furthermore, x is a Gaussian random variable with mean zero and variance $\|\boldsymbol{k}\|_2^2$. Therefore, we have

$$\mathop{\mathbb{E}}_{\boldsymbol{S}}[\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})] = \sqrt{\pi/2} \cdot \frac{\langle \boldsymbol{q}, \boldsymbol{k} \rangle}{\|\boldsymbol{k}\|_2} \cdot \mathop{\mathbb{E}}_{x}[|x|] = \langle \boldsymbol{q}, \boldsymbol{k} \rangle.$$

where the equality comes from the following Fact A.2:

Fact A.2 (Moments of Normal Random Variable). If x is a normal random variable with zero mean and variance σ^2, then for any integer ℓ, the ℓ-th moment of x is $\mathbb{E}\left[|x|^\ell \right] = \sigma^\ell \cdot 2^{\ell/2} \Gamma((\ell+1)/2)/\sqrt{\pi}$.

This completes the proof of Lemma 3.2. □

Lemma 3.3 (Distortion of $\texttt{Prod}_{\texttt{QJL}}$)**.** *For any $\boldsymbol{q}, \boldsymbol{k} \in \mathbb{R}^d$, if $\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k})$ is defined as in Equation* (3) *for $m \geq \frac{4}{3} \cdot \frac{1+\varepsilon}{\varepsilon^2} \log \frac{2}{\delta}$, then:*

$$\Pr_{\boldsymbol{S}}\left[|\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) - \langle \boldsymbol{q}, \boldsymbol{k} \rangle| > \varepsilon \|\boldsymbol{q}\|_2 \|\boldsymbol{k}\|_2 \right] \leq \delta.$$

Proof. First note that, letting $\boldsymbol{s}_1, \boldsymbol{s}_2, \dots \boldsymbol{s}_m$ denote the rows of the JL transform matrix S, we have:

$$\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) = \frac{1}{m} \sum_{i \in [m]} \sqrt{\pi/2} \cdot \|\boldsymbol{k}\|_2 \cdot \boldsymbol{s}_i^\top \boldsymbol{q} \cdot \texttt{sign}(\boldsymbol{s}_i^\top \boldsymbol{k}).$$

Since $\boldsymbol{s}_i$'s are i.i.d. the above is indeed the average of m i.i.d. estimators defined as $z_i := \sqrt{\pi/2} \cdot \|\boldsymbol{k}\|_2 \cdot \boldsymbol{s}_i^\top \boldsymbol{q} \cdot \texttt{sign}(\boldsymbol{s}_i^\top \boldsymbol{k})$ for $i \in [m]$. Let us now calculate the ℓ-th moment of z_i using Fact A.2:

$$\mathbb{E}\left[|z_i|^\ell\right] = \left(\sqrt{\pi/2} \cdot \|\boldsymbol{k}\|_2\right)^\ell \cdot \mathbb{E}\left[|\boldsymbol{s}_i^\top \boldsymbol{q}|^\ell\right] = \left(\sqrt{\pi} \cdot \|\boldsymbol{k}\|_2 \|\boldsymbol{q}\|_2\right)^\ell \cdot \frac{\Gamma((\ell+1)/2)}{\sqrt{\pi}}, \tag{4}$$

where the second equality above follows because $\boldsymbol{s}_i^\top \boldsymbol{q}$ is a Gaussian random variable with mean zero and variance $\|\boldsymbol{q}\|_2^2$ along with Fact A.2. Now we can prove the result by invoking the unbiasedness of the estimator, Lemma 3.2, along with an appropriate version of Bernstein inequality and using the moment bounds in Equation (4). More specifically, our moment calculation in Equation (4) implies:

$$\mathbb{E}\left[|z_i|^\ell\right] = \mathbb{E}\left[|z_i|^2\right] \cdot \left(\sqrt{\pi}\|\boldsymbol{k}\|_2\|\boldsymbol{q}\|_2\right)^{\ell-2} \cdot \frac{\Gamma((\ell+1)/2)}{\Gamma(3/2)} \leq \mathbb{E}\left[|z_i|^2\right] \cdot \left(\frac{2}{3} \cdot \|\boldsymbol{k}\|_2\|\boldsymbol{q}\|_2\right)^{\ell-2} \cdot \frac{\ell!}{2}$$

Therefore, by invoking a proper version of the Bernstein inequality, for instance Corollary 2.11 from Boucheron et al. (2003), we have the following:

$$\Pr_{\boldsymbol{S}}\left[|\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) - \langle \boldsymbol{q}, \boldsymbol{k}\rangle| > t\right] \leq 2\exp\left(\frac{3}{4} \cdot \frac{mt^2}{\|\boldsymbol{k}\|_2^2\|\boldsymbol{q}\|_2^2 + \|\boldsymbol{k}\|_2\|\boldsymbol{q}\|_2 \cdot t}\right).$$

If we set $t = \varepsilon\|\boldsymbol{q}\|_2\|\boldsymbol{k}\|_2$ the above simplifies to:

$$\Pr_{\boldsymbol{S}}\left[|\texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}, \boldsymbol{k}) - \langle \boldsymbol{q}, \boldsymbol{k}\rangle| > \varepsilon\|\boldsymbol{q}\|_2\|\boldsymbol{k}\|_2\right] \leq 2\exp\left(\frac{3}{4} \cdot \frac{m\varepsilon^2}{1+\varepsilon}\right).$$

Therefore if $m \geq \frac{4}{3} \cdot \frac{1+\varepsilon}{\varepsilon^2} \log \frac{2}{\delta}$ the error bound follows. This completes the proof of Lemma 3.3. □

Theorem 3.4 (Distortion bound on QJL key cache quantizer)**.** *For any sequence of key tokens $\boldsymbol{k}_1, \dots \boldsymbol{k}_n \in \mathbf{R}^d$ and any integer m, Algorithm 1 stores binary vectors $\tilde{\boldsymbol{k}}_1, \dots \tilde{\boldsymbol{k}}_n \in \{-1, +1\}^m$ along with scalar values $\nu_1, \dots \nu_n$ in the cache. If the key embeddings have bounded norm $\max_{i \in [n]} \|\boldsymbol{k}_i\|_2 \leq r$ and $m \geq 2r^2\varepsilon^{-2} \log n$, then for any query embedding $\boldsymbol{q}_n \in \mathbf{R}^d$ with bounded norm $\|\boldsymbol{q}_n\|_2 \leq r$ the output of the procedure* ESTIMATESCORES*($\boldsymbol{q}_n$) satisfies the following with probability $1 - \frac{1}{\texttt{poly}(n)}$ sinultaneously for all $i \in [n]$:*

$$\left|\widetilde{\texttt{Score}}(i) - \texttt{Score}(i)\right| \leq 3\varepsilon \cdot \texttt{Score}(i),$$

where Score *is the vector of attention scores defined in Equation* (1)*.*

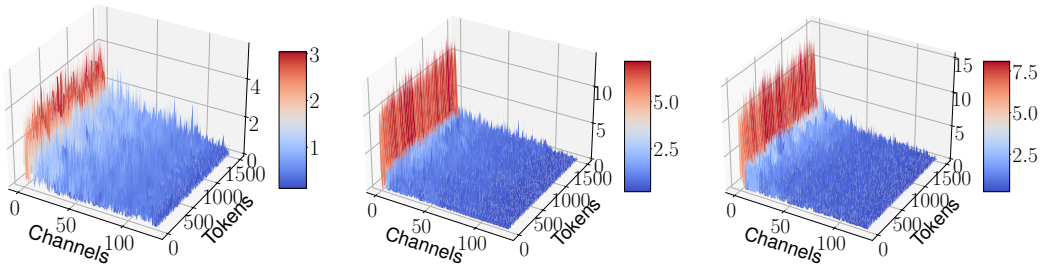
Proof. The proof is by invoking Lemma 3.3 and a union bound. For every $j \in [n]$ the estimator $\widetilde{\mathbf{qK}}(j)$ computed in line 6 of Algorithm 1 is in fact equal to the inner product estimator $\widetilde{\mathbf{qK}}(j) = \texttt{Prod}_{\texttt{QJL}}(\boldsymbol{q}_n, \boldsymbol{k}_j)$ as defined in Equation (3). Thus by Lemma 3.3 we have the following with probability at least $1 - \frac{1}{n^{3/(2+2\varepsilon)}}$:

$$\left|\widetilde{\mathbf{qK}}(j) - \langle \boldsymbol{q}_n, \boldsymbol{k}_j\rangle\right| \leq \frac{\varepsilon}{r^2} \cdot \|\boldsymbol{q}_n\|_2\|\boldsymbol{k}_j\|_2 \leq \varepsilon,$$

where the second inequality follows from the preconditions of the theorem regarding the boundedness of the norms of the query and key embeddings. By union bound, the above inequality holds simultaneously for all $j \in [n]$ with high probability in n. Thus after applying the softmax function in line 7 of Algorithm 1 we get that with high probability in n:

$$\widetilde{\texttt{Score}}(i) \in e^{\pm 2\varepsilon} \cdot \texttt{Score}(i) \in (1 \pm 3\varepsilon) \cdot \texttt{Score}(i).$$

This completes the proof of Theorem 3.4. □



(a) Layer 0, Head 0 (b) Layer 15, Head 0 (c) Layer 31, Head 0

Figure 1: The magnitude of key cache entries for different layers of the Llama-2 model, based on an example prompt, reveals notable patterns. The coordinates of embeddings (channels) are sorted by their average magnitude over tokens. In the initial layers, no significant outlier patterns are observed. However, in the deeper layers, a few channels (approximately four) exhibit visibly larger magnitudes, indicating the presence of significant outliers. This observation highlights the importance of addressing these outliers to improve quantization accuracy and reduce distortion in the key cache.

B Practical Consideration

Outliers. As reported in recent works e.g., KIVI Liu et al. (2024b), KVQuant Hooper et al. (2024), key embeddings typically contain outliers exhibiting a distinct pattern. Specifically, certain coordinates of key embeddings display relatively large magnitudes. To further investigate these observations, we analyze the distribution of the magnitudes of key embedding coordinates across different layers.

Firstly, we observe that there are no significant outliers in the initial attention layers. However, in the deeper layers, certain fixed coordinates of key embeddings consistently exhibit large magnitudes, and this pattern persists within these channels across all tokens. The distribution of outliers across different layers for the Llama-2 model is plotted in Figure 1. It is evident that in the initial layers, outliers are rare, but as we approach the final layers, their frequency and impact increase significantly. Secondly, the outliers show a persistent pattern in specific fixed coordinates of the key embeddings. This observation aligns with previous findings that certain fixed embedding coordinates exhibit larger outliers Dettmers et al. (2022); Lin et al. (2023); Liu et al. (2024b); Hooper et al. (2024).

Orthogonalized JL transform. We observed that orthogonalizing the rows of the JL matrix S in Definition 3.1 almost always improves the performance of our QJL quantizer. This finding aligns with previous work on various applications of the JL transform, such as random Fourier features Yu et al. (2016) and locality sensitive hashing Ji et al. (2012). Consequently, in our implementation and all experiments, we first generate a random JL matrix S with i.i.d. Gaussian entries and then orthogonalize its rows using QR decomposition. We then use this orthogonalized matrix in our QJL quantizer, as described in Algorithm 1.

C Ablation Study

Here, we perform an ablation study on the relative distortion of the attention scores in one attention layer after applying QJL on key embeddings. The distortion for various layers of the Llama2-7B model is plotted against the number of bits per token and embedding channel m/d, where $d = 128$ is the embedding dimension, as shown in Figure 2. Our theoretical result from Theorem 3.6 suggests that $m \sim 1/\varepsilon^2$ which aligns with our observations in Figure 2. An interesting observation is that the first layer has a much higher distortion compared to all other layers, suggesting that the first layer is more challenging to quantize and requires a higher number of bits per FPN. This finding is noteworthy and indicates the need for tailored quantization strategies for different layers. This is consistent with the outlier distribution depicted in Figure 1, where the first layer appears distinct from the others.

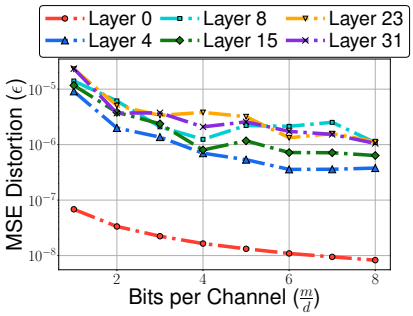
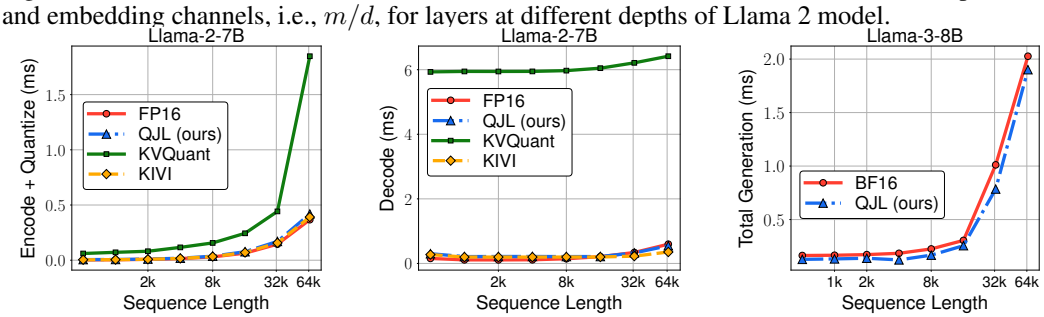


Figure 2: The relative distortion on the attention scores ε versus the number of bits of QJL per token and embedding channels, i.e., m/d, for layers at different depths of Llama 2 model.



(a) Prompt encoding (Llama2) (b) Token generation (Llama2) (c) Encode and generate (Llama3)

Figure 3: Wall-clock time (ms) to encode a prompt and quantize the KV cache (left), generate 128 tokens for llama2 model (middle), and generate 64 tokens for llama3 model (right) using different quantization methods in a single attention layer model. The input sequence length varies from 1k to 64k. Both KIVI and QJL (ours) with 3 bits per FPN show faster decoding time than the baseline. However, KVQuant is significantly slower during both quantizing and decoding phases. QJL is the only method that can quantize Llama3, as our kernels support grouped query attention and BF16 data type. We observe the same speed for Llama3 as the exact method for generation. Note that our memory usage is at least 5-fold less than the exact method and can support all data types.

D Extended Experiments

Runtime and Peak-Memory Evaluations. To evaluate the runtime and memory consumption of QJL we additionally report runtimes of: **(1)** prompt encoding, **(2)** KV cache quantization, and **(3)** decoding (token generation) in a single attention layer as well as the **(4)** peak memory consumption during prompt encoding and decoding. Figure 3 shows the wall-clock time to encode a prompt and quantize the KV cache, generate 128 tokens for Llama2 model, and generate 64 tokens for Llama3 model using different quantization methods in a single attention layer of these models. Note that QJL is the only method that can quantize Llama3, as our kernels support grouped query attention and BF16 data type. we observe the same speed for Llama3 as the exact method for generation. The input sequence lengths vary between 1k to 128k. As shown in Figure 3, KVQuant runs slower than other methods during both prompt encoding and decoding phases, as it requires a huge amount of preprocessing which leads to slow runtime. On the other hand, both KIVI and our QJL with 3 bits per FPN show marginal runtime overhead compared to the exact baseline during prompting but reduce KV cache memory usage by at least a factor of 5.

Next, we compare the peak memory consumption of various KV cache quantization methods applied to the Llama2 model for encoding prompts of different lengths and generating 128 new tokens, as shown in Figure 4. Both QJL and KIVI quantize the KV cache to 3 or 5 bits per FPN. However, peak memory consumption also includes the memory required to store model parameters. Even considering total memory consumption, we observe an over two-fold reduction in peak memory usage. We did not include KVQuant in the peak memory study as this method was extremely slow and running it repeatedly for different sequence lengths takes a very long time.

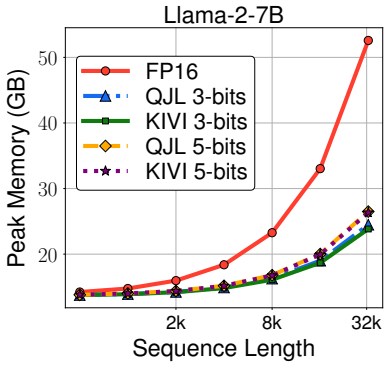


Figure 4: Peak memory usage for encoding the prompt and generating 128 tokens with Llama2, comparing various KV cache quantization methods to the exact model without quantization.