KURTAIL : KURTOSIS-BASED LLM QUANTIZATION

Sadegh Akhondzadeh[*][1], **Aleksandar Bojchevski**[1] **Evangelos Eleftheriou**[2], **Martino Dazzi**[2]

ABSTRACT

One of the challenges of quantizing a large language model (LLM) is the presence of outliers. Outliers often make uniform quantization schemes less effective, particularly in extreme cases such as 4-bit quantization. We introduce KurTail, a new post-training quantization (PTQ) scheme that leverages Kurtosis-based rotation to mitigate outliers in the activations of LLMs. Our method optimizes Kurtosis as a measure of tailedness. This approach enables the quantization of weights, activations, and the KV cache in 4 bits. KurTail utilizes layer-wise optimization, ensuring memory efficiency. KurTail outperforms existing quantization methods, offering a 13.3% boost in MMLU accuracy and a 15.5% drop in Wiki perplexity compared to QuaRot (Ashkboos et al., 2024b). It also outperforms SpinQuant (Liu et al., 2024) with a 2.6% MMLU gain and reduces perplexity by 2.9%, all while reducing the cost of training the rotation. For comparison, learning the rotation using SpinQuant for Llama3-70B requires at least four NVIDIA H100 80GB GPUs, whereas our method requires only a single GPU, making it a more accessible solution for consumer GPU.

1 INTRODUCTION

Large language models (LLMs) have advanced significantly in recent years, showcasing remarkable performance and capabilities. As these models grow in size and complexity, the computational cost required for their deployment and inference has increased dramatically. This has shifted the focus toward accelerating model performance while reducing memory and computational requirements. An effective method to achieve this is post-training quantization (PTQ), which involves representing model weights and/or activations in lower numerical precisions. PTQ can significantly reduce the memory footprint and computational overhead and subsequently decrease latency and energy consumption, which are especially beneficial for inference on resource-constrained edge devices.

Serving a model involves two stages of *prefilling* and *generation*. During *prefilling*, the model processes the input prompt and stores the internal state, which is known as key-value (KV) caching. During *generation*, tokens are produced auto-regressively. The *prefilling* stage is considered compute-bound, while the generation stage is memory-bound due to repeated access to and updates of the KV cache. Quantizing each stage offers distinct advantages for improving inference efficiency. KV-cache quantization reduces memory requirements and accelerates data movement, which enhances the *generation* stage, particularly in scenarios involving long-context inference. Weight quantization, on the other hand, reduces the memory footprint independently, and when it is combined with activation quantization, it also reduces the computational demands, which mainly speeds up the *prefilling* stage. However, activation quantization presents challenges due to large outliers (Dettmers et al., 2022; Xiao et al., 2023) in certain channels, which limits the effectiveness of uniform integer quantization as it destroys the dynamic range of the activations. While channel-wise quantization can effectively address this issue, the lack of hardware support makes it computationally expensive in practice. Several methods have been proposed to address this challenge. Dettmers et al. (2022) and Ashkboos et al. (2023) advocate for mixed-precision computation in which they store some of the channels in higher precision and less sensitive channels in lower precision to balance accuracy and efficiency. Xiao et al. (2023) introduces channel-wise scaling into the layer normalization and the weights of linear layers. Ashkboos et al. (2024b) proposed random rotation which takes the advantage of the computational invariance framework (Ashkboos et al., 2024a) to mitigate the outliers problem.

[*]Work done during an internship at Axelera AI. [1]University of Cologne [2]Axelera AI

We introduce *KurTail* – a novel approach to mitigating activation outliers by applying a learnable rotation to the activations. We advocate for learnable rotation instead of random rotation, which is suboptimal (Liu et al., 2024). Unlike SpinQuant (Liu et al., 2024), which requires end-to-end training KurTail focuses on reducing the tail density of activations independently per layer. We perform layer-wise inference to store activations and optimize the transformation based on the Kurtosis of activations. As a result, KurTail can be implemented in a significantly more memory-efficient manner. For instance, while SpinQuant requires at least four NVIDIA H100 80GB GPUs to compute rotations for Llama3-70B, KurTail achieves the same with just a single GPU. Despite its lower computational requirements, KurTail outperforms existing methods in terms of perplexity and zero-shot reasoning tasks. KurTail outperforms existing quantization methods with a 13.3% improvement in MMLU accuracy and a 15.5% decrease in Wiki perplexity compared to QuaRot(Ashkboos et al., 2024b). It also performs better than SpinQuant(Liu et al., 2024), achieving a 2.6% boost in MMLU accuracy and a 2.9% drop in perplexity, all while reducing the cost of training the rotation.

2 Background

Kurtosis. Kurtosis is a statistical measure that describes the degree of tailedness in the distribution of a dataset. It helps determine whether the data have heavy or light tails compared to a normal distribution. Mathematically, Kurtosis is defined as the standardized fourth moment of a population around its mean, and it is calculated using $\kappa = \frac{\mathbb{E}[(x-\mu)^4]}{(\mathbb{E}[(x-\mu)^2])^2} = \frac{\mu_4}{\sigma^4}$ where, μ is the mean, μ_4 is the fourth moment about the mean, and σ is the standard deviation. The Kurtosis of a normal distribution is 3. To center the Kurtosis value at zero for the normal distribution, the adjusted measure Kurtosis $-\,3$ is often used, which is referred to as excess Kurtosis. Positive Kurtosis is characterized by heavy tails and a sharp peak (indicating greater tail density than a normal distribution, e.g., the Laplace or t-distribution). Positive Kurtosis also means the shift of mass from the shoulders to both the tails and the center. On the contrary, negative Kurtosis is a sign of light tails and a flatter distribution (like uniform or beta distribution) caused by mass moving from the tails and center to the shoulders. (Banner et al., 2019) demonstrates that deep neural network weights and activations typically follow Gaussian or Laplace distributions. Furthermore, Dettmers et al. (2022) identifies the presence of extreme outliers in LLM parameters, which are critical for maintaining performance. Our key insight is that distributions with outliers exhibit high kurtosis, which measures the presence of extreme values. Therefore, by minimizing kurtosis, we can reduce the impact of outliers and bring the distribution closer to uniform by optimizing the rotation. Uniform distribution is the desired distribution of the activation and weight for uniform quantization (§ A.1) since we use the Kurtosis as a proxy for moving the distribution to get close to the uniform distribution. Therefore, we defined the loss as $\mathcal{L}_\kappa = \frac{1}{L}\sum_{i=1}^{L} |\kappa(\bigoplus_{j=1}^{N} \boldsymbol{a}_i) - \kappa_u|$ where $\bigoplus$ denotes the concatenation of the activation of all tokens at that layer and κ_u is the Kurtosis of the uniform distribution.

Quantization Sensitivity. Quantization sensitivity measures the difference in the quantization error when we slightly perturb the optimal scaling (Chmiel et al., 2020), see Theorem A.1 for formal definition. Theoretically the sensitivity decreases as the distribution become closer to uniform (see Theorem A.2). We evaluate our method by measuring activation sensitivity both before and after applying rotations optimized with Kurtosis. We expect that after applying these rotations, the activation distribution will be closer to uniform, resulting in better quantization robustness. We empirically measure the sensitivity of the activation distribution before and after applying the rotation. We utilize the Llama3.1 8-B model and apply two rotation techniques: one using a random Hadamard transformation and another using a Kurtosis-optimized rotation. First, we compute the optimal scaling (Chmiel et al., 2020) for activation quantization and then calculate the quantization sensitivity based on Theorem A.1.

In Fig. 1, the symbol α indicates the fraction of the optimal step size used to analyze quantization sensitivity. The results show that the random Hadamard transformation reduces quantization sensitivity. Additionally, our Kurtosis-based method exhibits an even more significant reduction in sensitivity, suggesting that it more effectively aligns the distribution with uniformity.

Evaluation of KurTail on Channel Outliers. To demonstrate that the learned rotation by KurTail reduces the degree of tailedness in the distribution, we visualize the inputs of multi-head self-attention (MHSA) and feed-forward network (FFN) blocks of layer 15 in Llama3-8B. In Fig. 2, we

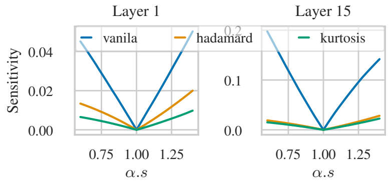

Figure 1: Empirical sensitivity of the MHSA input distribution across different rotations.

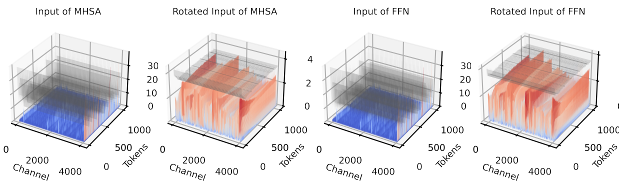

Figure 2: The input distribution of the MHSA and FFN blocks in the LLaMA3-8B model is shown before and after applying KurTail .

compare the input distribution once without rotation and once with KurTail learned rotation. Additionally, we highlight the maximum value for each token with a gray surface above each token. As shown, KurTail effectively mitigates outliers in activation quantization. We also analyze per-token maximum absolute values using rotations, with results in § A.5.

3 KurTail

Placement of the Rotations. Following the computational invariance theorem — as introduced by Elhage et al. (2023); Ashkboos et al. (2024a) and later utilized by QuaRot (Ashkboos et al., 2024b) and SpinQuant (Liu et al., 2024) — we adopted a similar framework to transform the activation functions at each layer. The placement of rotations is illustrated in Fig. 3. This figure depicts a single layer of a transformer model, where each square represents a computation block. The rotations are categorized into fusible rotations ($\boldsymbol{R}_1$ and $\boldsymbol{R}_2$) and online rotations ($\boldsymbol{R}_3$, $\boldsymbol{R}_4$, and $\boldsymbol{R}_5$). Fusible rotations do not add additional computational costs during inference since they can be merged with the model's original parameters. Specifically, we apply $\boldsymbol{R}_1$ to the left side of the token embedding, $\boldsymbol{W}_o$, and $\boldsymbol{W}_d$ within the MHSA and FFN blocks, respectively. The inverse of $\boldsymbol{R}_1$ is applied to the right side of $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$ in the attention block, and $\boldsymbol{W}_{up}$, $\boldsymbol{W}_{gate}$ in the FFN block. Due to the residual connection, the exact same rotation must also be applied across subsequent layers (e.g., $\boldsymbol{X}\boldsymbol{R}_1 + \boldsymbol{Y}\boldsymbol{R}_1$ in one layer and $\boldsymbol{Y}\boldsymbol{R}_1 + \boldsymbol{X}_2\boldsymbol{R}_1$ in the next). The second fusible rotation, $\boldsymbol{R}_2$, is applied to the right side of $\boldsymbol{W}_v$, with its inverse applied to the left side of $\boldsymbol{W}_o$. This transformation improves the distribution of value caches and can vary across layers. The second group of rotations, $\boldsymbol{R}_3$, $\boldsymbol{R}_4$, and $\boldsymbol{R}_5$, are online and can increase computational costs compared to the original model. To mitigate this, we utilize random Hadamard matrices, which are computationally efficient, resulting in minimal overhead. For $\boldsymbol{R}_3$, the transformation is applied after each rotational positional encoding for queries and keys. Since the transpose of any orthogonal matrix equals its inverse, there is no need to add the inverse matrix explicitly. During the computation of attention scores, the term $\boldsymbol{Q}^T\boldsymbol{K}$ simplifies to $\boldsymbol{Q}^T\boldsymbol{R}_3^T\boldsymbol{R}_3\boldsymbol{K}$, effectively nullifying the impact of the rotation. For $\boldsymbol{R}_4$, we introduce the transformation after applying the softmax scores to the values and add the inverse in the subsequent linear layer. Similarly, $\boldsymbol{R}_5$ is implemented in the FFN block using the same approach.

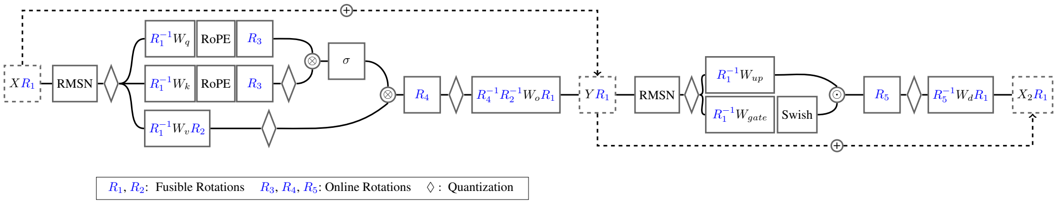

Figure 3: Diagram of a single-layer decoder network after applying rotations. Blocks containing both blue and black indicate that the rotation is fused into the network without adding extra computation. In contrast, blocks with only the rotation signify additional computations during inference.

Learning the Rotations. To discover the optimal rotations, we first run the vanilla model and store the inputs from both the MHSA and FFN blocks. Next, we create a small network consisting of a linear layer and an RMSNorm, designed to simulate the inputs of the MHSA and FFN blocks before quantization (Fig. 3). For optimization, we shuffle the stored input data from all transformer layers and both blocks and then train the rotation using Kurtosis loss. Since the optimization requires the rotations to remain within the orthogonal space, we use the Caley Adam (Li et al., 2020) optimizer to enforce this constraint. We train this small network for 100 iterations using 500 samples from the WikiText (Merity et al., 2016b) training set. After training, the resulting rotation is fused into the original network. For the $\boldsymbol{R}_2$, we did apply a similar approach: we removed the RMSNorm and just optimized the linear layer with Kurtosis loss.

Training Cost. While quantization make the inference of large models feasible on consumer GPUs, finding the optimal rotation still requires substantial computational power. We address this by avoiding end-to-end fine-tuning. Since each multi-head attention and FFN is affected by $\boldsymbol{R}_1$, end-to-end approaches like SpinQuant cannot optimize the rotation layer by layer, and directly optimizing $\boldsymbol{R}_1$ via gradient descent requires loading the entire model, which is memory-intensive. Although SpinQuant reduces training costs by eliminating the need to store weight gradients and states, it still requires loading the full model into GPU memory. Our approach uses layer-wise inference, which eliminates the need to load all the network weight on the GPU at once, store the activations for each layer, and optimize the rotation with a Kurtosis loss. This significantly lowers GPU requirements—at most, a single NVIDIA H100 (or A100) is needed for LLaMA 70B.

Results. To evaluate KurTail we focus on 4-bit quantization for weights, activations and KV-cache, which is a challenging bit-width for LLM quantization. The detail explanation of the experimental setups, models and evaluation dataset provided in § C. Table 1 shows a summary where ”0-shot” means the average performance over 8 tasks. For weight quantization we used GPTQ (Frantar et al., 2022). For all of the result we have better perplexity in all of the models compared to previous methods. At the same time, our method is significantly better that SpinQuant (Liu et al., 2024) and QuaRot (Ashkboos et al., 2024b) in downstream tasks. We provide further results for mixture of experts models in § B.1. We also provide results for math reasoning in § B.2.

Table 1: Comparison of different quantization methods across various models. All the result report for 4 bit quantization for W/A/KV cache. Weights are quantized using GPTQ(Frantar et al., 2022)

Method	**Llama-2-7b**			**Llama-2-13b**			**Llama-3-8b**		
	Wiki (↓)	**0-shot (↑)**	**MMLU (↑)**	**Wiki (↓)**	**0-shot (↑)**	**MMLU (↑)**	**Wiki (↓)**	**0-shot (↑)**	**MMLU (↑)**
16-bit	5.5	64.1	42.1	4.9	66.5	52.7	6.1	67.2	63.2
GPTQ	9600.0s	38.9	23.8	3120.0	33.8	24.8	166.3	39.8	23.3
QuaRot	6.2	60.6	32.3	5.4	64.7	46.83	8.50	60.1	47.4
SpinQuant	6.0	61.0	34.8	5.2	64.8	47.8	7.4	63.8	56.2
Kurtail	**5.9**	**61.3**	**32.9**	**5.2**	**65.2**	**49.1**	**7.2**	**64.6**	**57.3**
Method	**Llama-3-70b**			**Llama-3.2-1b**			**Llama-3.2-3b**		
	Wiki (↓)	**0-shot (↑)**	**MMLU (↑)**	**Wiki (↓)**	**0-shot (↑)**	**MMLU (↑)**	**Wiki (↓)**	**0-shot (↑)**	**MMLU (↑)**
16-bit	2.8	73.1	76.3	9.75	54.9	37.9	7.8	62.7	54.8
GPTQ	452.7	45.5	23.2	108.9	38.0	24.9	178.3	40.3	24.8
QuaRot	6.19	65.1	62.9	17.4	49.0	23.8	10.1	56.1	42.0
SpinQuant	6.2	65.7	59.4	13.6	48.8	25.6	9.2	57.9	44.2
Kurtail	**4.2**	**70.7**	**73.1**	**12.9**	**50.1**	**27.2**	**9.0**	**59.0**	**47.8**

4 CONCLUSION

We introduced KurTail – a novel technique for learning orthogonal transformations that rotate the activation distribution to address the outlier problem. KurTail effectively reduces quantization sensitivity and minimizes quantization error by tackling important challenges, such as the outlier issue, and overcomes the limitations of previous approaches. Compared to QuaRot (Ashkboos et al., 2024b), which uses non-learnable rotation, and SpinQuant (Liu et al., 2024), which requires substantial computational resources for learning rotations, KurTail provides a more efficient and robust solution. These results highlight KurTail ’s ability to deliver efficiency and high performance across large-scale language models.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. *URL https://arxiv.org/abs/2404.14219*, 2024.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefler, and Dan Alistarh. Towards end-to-end 4-bit inference on generative large language models. *arXiv preprint arXiv:2310.09259*, 2023.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024a.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*, 2024b.

Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models, November 2023. URL `https://mobiusml.github.io/hqq_blog/`.

Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment, 2019. URL `https://arxiv.org/abs/1810.05723`.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Michael Boratko, Harsh Padigela, Deepak Mikkilineni, Pavan Yuvraj, Rajarshi Das, Andrew McCallum, Mihai Chang, Achille Fokoue, Pavan Kapanipathi, Nicholas Mattei, et al. Arc: A machine reading comprehension dataset for reasoning over science text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1414–1423, 2018.

Brian Chmiel, Ron Banner, Gil Shomron, Yury Nahshan, Alex Bronstein, Uri Weiser, et al. Robust quantization: One model to rule them all. *Advances in neural information processing systems*, 33:5308–5317, 2020.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

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.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Nelson Elhage, Robert Lasenby, and Christopher Olah. Privileged bases in the transformer residual stream, 2023. URL `https://transformer-circuits.pub/2023/privileged-bases`. Transformer Circuits Thread.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Leo Gao, Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, et al. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2021.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL `https://arxiv.org/abs/2401.04088`.

Jun Li, Li Fuxin, and Sinisa Todorovic. Efficient riemannian optimization on the stiefel manifold via the cayley transform. *arXiv preprint arXiv:2002.01113*, 2020.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100, 2024.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant–llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*, 2024.

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

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Openbookqa: Fact-based open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 268–277, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

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.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019. URL `http://arxiv.org/abs/1910.03771`.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

A Appendix

A.1 Uniform Quantization for k-bit Precision

For a given vector $\boldsymbol{x}$, uniform integer quantization reduces its continuous range of values to a finite set of discrete levels, enabling representation in lower precision. In k-bit quantization, the value range $[x_{\text{min}}, x_{\text{max}}]$ is divided into 2^k equal intervals. Each element x_i in $\boldsymbol{x}$ is mapped to its closest quantization level by $Q(x_i) = \text{round}\left(\frac{x_i - b}{s}\right) \cdot s + b$. Here s is the scale factor or step size and b is the shift. The values of s and b depend on the specific quantization scheme. In symmetric quantization, the range is assumed to be symmetric around zero. Therefore, $b = 0$, and $s = \frac{\max(|x_{\text{max}}|, |x_{\text{min}}|)}{2^{k-1}-1}$. Alternatively, in asymmetric quantization, the range is not assumed to be centered at zero and therefore, $b = \min(\boldsymbol{x}), \quad s = \frac{x_{\text{max}} - x_{\text{min}}}{2^k - 1}$. Given $\boldsymbol{x}$ sampled from distribution f, quantizer Q minimize the error between the quantized and the original values. The expected mean-squared error (MSE), a measure of error, is defined as $\text{MSE}(\boldsymbol{x}, Q) = \mathbb{E}\left[(\boldsymbol{x} - Q(\boldsymbol{x}))^2\right]$.

Definition A.1. Quantization Sensitivity (Chmiel et al., 2020) For a given distribution f and its corresponding vector $\boldsymbol{x}$, let $\tilde{s}$ denote the optimal quantization step size, and let $Q_{\tilde{s}}(\boldsymbol{X})$ represent the optimal quantizer. Quantization sensitivity $\Gamma(\boldsymbol{X}, \epsilon)$ is defined as the increase in the mean squared error (MSE) caused by a small perturbation $\epsilon > 0$ in the quantization step size s around $\tilde{s}$, such that $|s - \tilde{s}| = \epsilon$. Specifically, the sensitivity is given by:

$$\Gamma(\boldsymbol{X}, \epsilon) = |\text{MSE}(\boldsymbol{X}, s) - \text{MSE}(\boldsymbol{X}, \tilde{s})|\,, \tag{1}$$

Theorem A.2. *(Chmiel et al., 2020) Considering $\boldsymbol{x}_U$ and $\boldsymbol{x}_N$ be continuous random variables with uniform and normal distributions. Then, for any given $\varepsilon > 0$, the quantization sensitivity $\Gamma(\boldsymbol{X}, \varepsilon)$ satisfies $\Gamma(\boldsymbol{X}_U, \varepsilon) < \Gamma(\boldsymbol{X}_N, \varepsilon)$,*

This theorem indicates that, compared to the typical normal distribution, the uniform distribution is more robust to changes in the quantization step size s. Therefore, it becomes apparent that there is great benefit in adjusting he distribution of the activations and weight to get closer to uniform distribution. It can also be shown for the uniform distribution the optimal scaling,$\tilde{s}$ is equal to $b = \min(\boldsymbol{x}), \quad s = \frac{x_{\text{max}} - x_{\text{min}}}{2^k - 1}$. Chmiel et al. (2020) also show that the optimal step size for a uniform distribution closely approximates the most robust quantization (less sensitive) step size.

A.2 Related Works

With the emergence of large foundation models and the continuous scaling of model sizes to billions and even trillions of parameters, interest in quantization of LLMs also increased. Some previous work introduced weight-only quantization (Frantar et al., 2022; Lin et al., 2024; Egiazarian et al., 2024; Tseng et al., 2024). These methods project the weights into a lower precision such as 4 bits or 3 bits or even less and then de-quantize to higher precision before actual computation, which resulted in all computations still being done in high precision. Several studies (Xiao et al., 2023; Ashkboos et al., 2024b; Liu et al., 2024) attempted to introduce quantization methods for both weight and activation. They showed that uniform quantizing is not practical for quantizing large language models since they suffer from the existence of large outliers. To address this issue Dettmers et al. (2022) proposed a mixed-precision approach for handling outliers at higher precision. Others (Xiao et al., 2023; Lin et al., 2024) proposed trading outliers between weights and activations by introducing a re-scaling paradigm. Tseng et al. (2024) introduced an incoherence processing method using random rotation matrices and applying vector quantization on the weights for compression, which also added overhead to inference. QuaRot Ashkboos et al. (2024b) was inspired by (Tseng et al., 2024) and taking the advantage of the previous the invariance compotation introduced by (Ashkboos et al., 2024a) introduced a rotation-based approach to compress and remove outliers from the space of activation using a random Hadamard rotation. Later SpinQuant (Liu et al., 2024) employing a similar technique improves the results of QuaRot(Ashkboos et al., 2024b) by learning some of this rotation using end-to-end training of quantization. SpinQuant improved the results compared to QuaRot. In this work, we introduce KurTail similar to SpinQuant learns the rotation by compressing outliers in the space of activation in contrast we use a unsupervised one-shot approach which reduce the computational cost of finding the rotation, further empirically we show that it also reduced over-fitting and improve the performance on downstream tasks.

A.3 Kurtosis of Different Distirbution

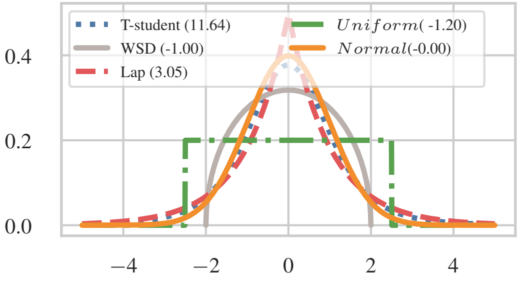

Figure 4: Kurtosis of well known distribution

A.4 Optimization in the Orthogonal Space

As discussed in § 3, in order to be consistent with a computational invariance framework, the transformation needs to be optimized in the orthogonal space. Therefore we optimize all of the transformation matrix within the Stiefel Manifodl (Li et al., 2020) i.e, the space of orthonormal matrices, using Caley Stochastic Gradient Descent (SGD) or Caley Adam (Li et al., 2020). The Caley SGD algorithm efficiently updates a transformation matrix T at each iteration. The updated transformation T' is defined as follows:

$$T' = \Delta_T(Y)T := \left(I - \frac{\alpha}{2}Y\right)^{-1}\left(I + \frac{\alpha}{2}Y\right)T \tag{2}$$

where :

$$\Delta_T(Y) := \left(I - \frac{\alpha}{2}Y\right)^{-1}\left(I + \frac{\alpha}{2}Y\right) \tag{3}$$

is referred to as the *Cayley Transform* of the skew-symmetric matrix Y (i.e., $Y^\top = -Y$). The matrix Y is derived from a projection $\widehat{G}$, which in turn is related to the gradient $G := \nabla_T \mathcal{L}_Q$ of the loss function:

$$Y = \widehat{G} - \widehat{G}^\top, \quad \widehat{G} := GT^\top - \frac{1}{2}TT^\top GT^\top. \tag{4}$$

It can be shown that the operator $\Delta_T(Y)$ preserves orthonormality, ensuring that T' remains orthonormal ($T'^\top T' = I$) as long as T is orthonormal. While Eq. 3 involves a matrix inversion, the updated transformation T' can also be efficiently computed using a fixed-point iteration method. This approach can achieve orthonormality with comparable computation time per iteration compared to a standard SGD update.

A.5 Further Evaluation of KurTail on Channel Outliers

In dynamic per-token quantization, the maximum value of a token's vector plays a critical role in determining the quantization step size and range. Larger maximum values increase the quantization range, which results in larger quantization steps and greater precision loss. Alternatively reducing the maximum value allows for smaller quantization steps, which result in more efficient representation of token values with minimal degradation of information. Therefore, lowering the maximum values across tokens is directly connected to overall quantization error and model performance. To evaluate how well different methods achieve this goal, we measure the success rate of our proposed method, KurTail , compared to its un-rotated counterpart (baseline vector) and an alternative rotation method, QuaRot. A "success" is defined as a case where the maximum value of a token's vector after applying a rotation-based transformation (KurTail or QuaRot) is smaller than that of the baseline vector. The success rate defined the percentage of tokens where the rotated version achieves this reduction. In Table 2, we present the average success rates for LLAMA3-8B. KurTail consistently produces smaller maximum values across all layers, samples, and tokens, achieving a higher success rate compared to the baseline vector in nearly all cases. Additionally, it outperforms QuaRot in approximately 63.29% in MSHA, 62.99% in FFN on average.

Table 2: The success rate of minimum max values per-token under different rotations.

Comparison	Success Rate (%)
MHSA Input	
Vanilla vs. KurTail	0.26% - 99.74%
Vanilla vs. QuaRot	0.57% - 99.43%
Kurtail vs. QuaRot	63.29% - 36.71%
FFN Input	
Vanilla vs. KurTail	0.04% - 99.96%
Vanilla vs. QuaRot	0.04% - 99.96%
Kurtail vs. QuaRot	62.99% - 37.01%

B Further Evaluations

B.1 Experiment on Mixture of Experts

Given the growing popularity of the Mixture of Experts (MoE) models, we also explore the idea of applying rotation within the mixture of experts. For this purpose, we utilize Mixtral (Jiang et al., 2024), which employs the exact same attention block. However, for the mixture of experts component, we apply rotation across all the experts. Table 3 presents the results for 4-bit quantization, where we used rounding to the nearest value. In principle, other quantization methods, such as GPTQ, HQQ, and similar approaches, can also be employed to further enhance performance.

Table 3: Comparison of different quantization methods for Mixtral. All the result report for 4 bit quantization for W/A/KV cache. For weight quantization we used RTN.

Method	Mixtral-8x7B		
	Wiki (↓)	Avg. 0-shot (↑)	MMLU (↑)
16-bit	3.8	71.2	68.8
QuaRot	8.7	55.7	36.8
Kurtail	6.5	59.4	44.8

B.2 Evaluating Mathematical Reasoning

To explore more complex reasoning tasks, we further provide the performance quantized model on tasks involving mathematical reasoning in Table 4 by reporting results on the MathQA(Amini et al., 2019) dataset. MathQA is a benchmark designed to test problem-solving and quantitative reasoning abilities. The dataset consists of real-world mathematical problems covering topics such as arithmetic, algebra, probability, and geometry. Each problem is accompanied by a natural language description, multiple-choice answers, and an annotated solution program that outlines the reasoning steps required to reach the correct answer.

C Setup

C.1 Setup

We developed KurTail using the Hugging Face library (Wolf et al., 2019) integrated with the PyTorch framework (Paszke et al., 2019). For learning the transformation, we used 512 calibration samples for all model except the Mixteral and LLAMA 70B models which we use 256 calibration sample from the WikiText-2 (Merity et al., 2016b) training set, each with a sequence length of 2048. For storing the activation we used layer-wise inference and cpu offloading to reduce the GPU memory requirement. For optimizing the rotation, we use AdamG (Li et al., 2020) optimizer to find the rotation in the Stiefel manifold i.e., the set of all orthonormal matrices. The activation quantization

Table 4: Comparison of different quantization methods across various Llama model families and Phi3 model for mathematical reasoning. All the result report for 4 bit quantization for W/A/KV cache. For weight quantization we used GPTQ(Frantar et al., 2022)

Method	**MathQA**		
	16-bit	**QuaRot**	**KurTail**
Llama-2-7b	28.24	26.70	**26.77**
Llama-2-13b	31.76	28.81	**30.35**
Llama-2-70b	38.39	33.97	**35.68**
Llama-3-8b	40.30	31.36	**34.71**
Llama-3-70b	51.79	35.54	**45.76**
Llama-3.2-1b	28.94	25.29	**26.00**
Phi3-mini	39.93	31.89	**34.81**

was achieved through per-token dynamic symmetric quantization, where a single scale was applied to each row, and all them clip with a quantile of 0.98 in all experiments. For the KV caches, we employed asymmetric quantization. For the Weight quantization, we use round-to-nearest (RTN), HQQ (Badri & Shaji, 2023), and GPTQ (Frantar et al., 2022), using per-column (or per-channel) symmetric quantization. The clipping ratio was optimized via a linear search to reduce squared error. For GPTQ quantization, we uses 128 calibration samples from the WikiText-2, each with a sequence length of 2048. Learning the transformation and Transforming LLAMA3-70B with KurTail on an NVIDIA H100 GPU took around one hour which compare the SpinQuant it uses way less memory. (8 GPU and 2 hours).

C.2 Model

We evaluate KurTail on the LLAMA-2 (Touvron et al., 2023), LLAMA-3 (Dubey et al., 2024), Phi-model family (Abdin et al., 2024) on both language generation and zero-shot tasks. We further also target the mixture of expert model and provide the result for Mixtral (Jiang et al., 2024). We implement an memory efficient of training the the transformation by CPU offloading layers and saving the activation and optimizing the Kurtosis of the layer by implementing a simple linear layer later. Our implementation can be run a 70 billion model on the consumer gpus which can be useful for the user with private fine-tuned model that use consumer gpus. Due to similarity of the architecture with (Ashkboos et al., 2024b), we didn't re-implement the low level kernel for 4 bit matrix multiplication since the same performance result expected.

C.3 Evaluation Setting

To compare the performance of the model after quantization, we report the perplexity (PPL) score on the WikiText-2 (Merity et al., 2016a) test set. We also report results on zero-shot tasks since, while perplexity is a standard measure of language modeling performance, it may not be sufficient for evaluating the model's effectiveness after quantization. For zero-shot reasoning, we assess performance using the lm-evaluation-harness (Gao et al., 2024), testing the models on eight tasks: BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-Easy, and ARC-Challenge (Boratko et al., 2018) (Avg 0-shot). Additionally, to assess the model on more complex tasks, we benchmark its language comprehension and general understanding using the MMLU benchmark (Hendrycks et al., 2021) and for mathematical reasoning we utilize MathQA(Amini et al., 2019). We report the average performance in Table 1.