Fast and Accurate Fisher-Guided Quantization via Efficient Kronecker Factorization

Viktoriia Chekalina[1,2], **Timofey Gerasin**[2,3], **Andrey Kuznetsov**[1,4], **and Evgeniy Frolov**[5]
[1]FusionBrain Lab, [2]MSU, [3]Russian Research Institute,
[4]Innopolis University, [5]AXXX
Correspondence: sayankotor1@gmail.com

Abstract

Quantization has shown strong results in preserving model quality under compression. However, under aggressive bit-width reductions, even quantization may require additional information to prevent performance degradation. A natural source of it is the second-order curvature information, captured by the Hessian. Since the Hessian of the model layers is prohibitively large, direct computation is infeasible, making structured parameterizations and approximations crucial in practice.

In this work, we propose an efficient Kronecker-factored approximation yielding state-of-the-art performance when integrated into existing quantization schemes. Evaluations on the LLaMA and Qwen model families show near-baseline quality at 4-bit compression and only a 5–6% degradation at 2-bit for models with 7–8B parameters. Moreover, our method substantially accelerates the most expensive component in second-order quantization – Hessian parameterization – achieving up to a 10× speedup over prior approaches. Quantized model checkpoints are available at https://huggingface.co/collections/timo13113/fastkron-collection.

1 Introduction

Large language models (LLMs) have accelerated progress across a wide range of downstream applications. However, their size and computational demands remain prohibitive, making LLM compression a critical research direction.

Within this framework, we focus on quantization as the compression method. The post-training quantization (PTQ) problem can be formulated as minimizing the second-order Taylor expansion of the loss around the optimum:

$$\arg\min_{\mathbf{W}\in C} \approx \frac{1}{2}(\mathbf{W}-\mathbf{W}^\star)^T\left(\nabla^2_{\mathbf{W}^\star}L\right)(\mathbf{W}-\mathbf{W}^\star) \quad (1)$$

where $\nabla^2_{\mathbf{W}^\star}L$ is a Hessian, $\mathbf{W}$ denotes the obtained low-precision layer weights after quantization, $\mathbf{W}^\star$ the original high-precision weights, and C the set of possible quantization algorithms. Since the first-order derivative vanishes at the optimum, effectiveness of compression must rely on second-order information, i.e., the Hessian of the loss.

Assuming that the layer weights follow a multivariate normal (MVN) distribution, the Fisher information matrix $\mathcal{I}_F$ – and consequently the Hessian – can be expressed as a Kronecker product of the inverses of the row and column covariance matrices, $\mathbf{\Sigma}_{\text{row}}$ and $\mathbf{\Sigma}_{\text{col}}$:

$$\nabla^2_{\mathbf{W}^\star}L \approx \mathcal{I}_F(\theta) = \mathbf{\Sigma}^{-1}_{\text{col}} \otimes \mathbf{\Sigma}^{-1}_{\text{row}} = \boldsymbol{H}_I \otimes \boldsymbol{H}_O \quad (2)$$

We can rewrite the standard layer-wise quantization algorithm Q applied to weight $\mathbf{W} \in \mathbb{R}^{m\times n}$ to incorporate second-order information:

$$\mathbf{W} = Q(\mathbf{W}^\star + \mathbf{L_O}^\top \Delta\mathbf{W}\,\mathbf{L_I} + \mathbf{L_O}^\top \Delta\mathbf{W} + \Delta\mathbf{W}\,\mathbf{L_I}) \quad (3)$$

where $\Delta\mathbf{W} = \mathbf{W}^\star - \mathbf{W}$ corresponds to the quantization error of already quantized blocks, and $\mathbf{L_O}$ and $\mathbf{L_I}$ are the matrices obtained from the LDL decomposition of $\mathbf{H_O}$ and $\mathbf{H_I}$ from Eq. 2, respectively. In the Eq. 3 we split weight matrix $\mathbf{W}$ into small blocks and the quantization is performed iteratively, block by block, until the entire matrix is quantized.

As can be seen, in second-order-based quantization the overall procedure naturally decomposes into two parts: (i) the computation of factor matrices that capture second-order information, and (ii) the rounding algorithm, which takes these matrices as parameters.

Given the size of the Hessian matrix, the first part is quite time-consuming. Although quantization is typically a one-time process, in practical applications, it may be performed multiple times. For

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 38925–38935

example, in continual learning scenarios, models are repeatedly updated with new data streams, necessitating re-quantization after each fine-tuning stage. In addition, target-specific optimization may require applying quantization multiple times to adapt a model to different hardware platforms, each with distinct precision constraints. As a result, most second-order compression methods rely on computationally lighter diagonal-only approximations (Hsu et al., 2022; Hua et al., 2022). However, some approaches (Tseng et al., 2025b) perform a full Hessian decomposition taking into account off-diagonal interactions. This raises a natural question: which decomposition of the full Hessian is optimal for LLMs in terms of convergence speed?

We further investigate the following research questions: How critical is second order in quantization for different compression levels? How important are off-diagonal Hessian components, and how does their inclusion or omission affect downstream performance?

In our method, we adopt a standard rounding procedure while modifying the target weights to explicitly incorporate second-order curvature information, as shown in Eq. 3. To obtain the factor matrices, we decompose the full Hessian using the Lanczos algorithm. Our factor computation explicitly accounts for off-diagonal Hessian elements, which leads to improved compression quality. Moreover, as shown in Section 3, the spectral structure of the Hessian allows the proposed decomposition to converge faster than those used in methods that also employ generic full (off-diagonal) Hessian factorizations.

The method achieved state-of-the-art (SOTA) results across LLaMA and Qwen model families of various scales, achieving a 10× reduction in the computational overhead of factor computation. Compared to existing curvature-aware baselines, it preserves downstream performance, showing negligible degradation at 4-bit relative to 16-bit and only a small average performance drop at 2-bit precision.

To summarize, we make the following contributions:

- We propose FastKron, a strong layer-wise quantization method that factorizes the Hessian and incorporates it into the layers' weight rounding algorithm.
- We demonstrate that the proposed factorization is optimal for LLMs in terms of convergence rate and has up to 10× empirical speedup among methods in the same category.

Notably, faster factor computation is a *drop-in* improvement in second-order quantization pipelines: faster factor computation can be incorporated into every layer-wise quantization scheme.

2 Related Work

Post-Training Quantization has become a standard approach for compressing large language models. Early PTQ methods primarily relied on scalar quantization with adaptive rounding. For example, QuIP (Tseng et al., 2025a) performs per-weight scalar quantization with adaptive rounding guided by a reconstruction objective, while NWQ (Wang et al., 2022) learns rounding policies for weights by accounting for inter-layer dependencies.

More recent work goes beyond scalar rounding by quantizing blocks of weights as vectors using structured or learned codebooks. QuIP# (Tseng et al., 2024) extends QuIP-style preprocessing and introduces vector quantization with lattice codebooks to improve fidelity in extreme low-bit regimes. Similarly, AQLM (Egorov et al., 2024) revisits additive multi-codebook quantization and represents weight groups as sums of codebook vectors, achieving strong accuracy–compression tradeoffs. Methods such as TCQ (Marcellin and Fischer, 1990) and QTIP (Tseng et al., 2025a) investigate trellis-based and structured search quantization methods, which cast quantization as a constrained optimization or path-search problem.

Regarding the use of second-order information for quantization, HAWQ and HAWQ-V2 (Yao et al., 2020) allocate mixed precision by analyzing Hessian spectra, GPTQ (Frantar et al., 2022) shows that efficient blockwise approximations of the Hessian are sufficient for scaling LLMs to 3–4 bits. YAQA (Tseng et al., 2025b) provides an adaptive rounding rule that consumes Kronecker-factored layerwise Hessians.

Notably, obtaining a precise factorization of second-order information remains a challenging problem, as the Hessian is prohibitively large. K-FAC (Martens and Grosse, 2015) approximates Hessian as a product of two smaller diagonal covariance matrices that capture input activations and output gradients of the layer. FWSVD (Hsu et al., 2022) and TFWSVD (Hua et al., 2022) adopt a diagonal approximation of Hessian for low-rank compression, aligning the factorization objective with

parameter importance. GFWSVD (Chekalina et al., 2025) extends this approach to non-diagonal part by exploiting the Hessian structure and introducing an efficient factor computation. The model (Eschenhagen et al., 2023) proposes an optimization-driven approach for estimating factors - they are treated as learnable parameters.

3 Methodology

We compress the model via layer-wise vector quantization while incorporating curvature information from the loss surface. The first step of the method is to obtain full Hessian factor matrices $\mathbf{H_I}$ and $\mathbf{H_O}$, including both diagonal and off-diagonal components, which capture correlations across feature and output dimensions. Specifically, second-order information is collected on a calibration dataset $|D|$ by accumulating gradient outer products for each layer, yielding an empirical approximation of the Fisher information matrix.

Following (Loan and Pitsianis, 1992), Kronecker factors can be defined as a reshaped leading triplet of a permuted $\mathcal{I}_F$. The leading triplet is typically computed using the Lanczos (Lanczos, 1950) algorithm. This procedure, which effectively performs an iterative SVD, is computationally expensive when applied to the Hessian of a linear layer. To mitigate this bottleneck, we leverage the permutation properties alongside the properties of the Kronecker product. The pseudocode for computing $\mathbf{H}_I$ and $\mathbf{H}_O$ is provided in Algorithm 1. This allows us to avoid constructing the full $(m^2 \times n^2)$ Hessian and to reduce each SVD iteration to the sequential multiplication of three matrices with sizes of linear layer (as in line 7,8 of Algorithm 1), making decomposition tractable for LLMs. For every considered experimental setup, per-batch gradients $\mathbf{G}_i$ are collected and processed individually. This per-batch treatment is necessary because minibatch averaging does not commute with iterative SVD – these two operations cannot be interchanged. To collect gradients, we use the FineWeb calibration dataset (Penedo et al., 2024), chosen for its high quality and diversity, and batch size 128.

Only a limited number of methods refine weight matrices using full non-diagonal Hessian approximations. One representative example is YAQA (Tseng et al., 2025b), which naturally assumes the use of the power iteration method (Golub and Van Loan, 2013) to obtain factors. However, precomputing factors in this way can require up to a day for models with 7–8B parameters, rendering the approach hardly applicable for time-sensitive workflows. Both these algorithms — Power Iteration and Lanczos — have distinct advantages and limitations, and their efficiency varies depending on the properties of the data to which they are applied.

Algorithm 1 FastKron Factorization

Require: List of matrices $\{\mathbf{G}_i\}_{i=1}^{|D|}$, $|D|$ – calibration dataset, K – Lanczos iterations

1: $\mathbf{H} \leftarrow \frac{1}{|D|}\sum_{i=1}^{|D|} \mathrm{vec}(\mathbf{G}_i)\mathrm{vec}(\mathbf{G}_i)^T$ ▷ Never materialized
2: $\tilde{\mathbf{H}} = \mathcal{R}\mathbf{H} \leftarrow \frac{1}{|D|}\sum_{i=1}^{|D|} \mathbf{G}_i \otimes \mathbf{G}_i^T$ ▷ Never materialized
3: $\tilde{\mathbf{H}}\mathbf{z} = \frac{1}{|D|}\sum_{i=1}^{|D|} \mathrm{vec}(\mathbf{G}_i^\top \mathrm{reshape}(\mathbf{z})\mathbf{G}_i)$ ▷ $\mathbf{H}$ to vector multiplication
4: Truncated SVD:
5: $\mathbf{v} \leftarrow$ random unit vector, $\beta \leftarrow 0, \mathbf{v}_{prev} \leftarrow \mathbf{0}$
6: **for** $j = 1$ **to** K **do**
7: $\quad \mathbf{y} \leftarrow \frac{1}{|D|}\sum_{i=1}^{|D|} \mathrm{vec}(\mathbf{G}_i^\top \mathrm{reshape}(\mathbf{v})\mathbf{G}_i)$ ▷ $\tilde{\mathbf{H}}\mathbf{v}$
8: $\quad \mathbf{w} \leftarrow \frac{1}{|D|}\sum_{i=1}^{|D|} \mathrm{vec}(\mathbf{G}_i \mathrm{reshape}(\mathbf{y})\mathbf{G}_i^\top)$ ▷ $\tilde{\mathbf{H}}^T\mathbf{y}$
9: $\quad \alpha_j \leftarrow \mathbf{w} \cdot \mathbf{v}$
10: $\quad \mathbf{w} \leftarrow \mathbf{w} - \alpha_j \mathbf{v} - \beta \mathbf{v}_{prev}$
11: $\quad \beta \leftarrow \|\mathbf{w}\|_2$
12: $\quad \mathbf{v}_{prev} \leftarrow \mathbf{v}, \mathbf{v} \leftarrow \mathbf{w}/\beta$
13: **end for**
14: $\sigma \leftarrow \sqrt{\lambda_{max}(\mathbf{T})}$, where $\mathbf{T}$ is constructed from $\{\alpha_j, \beta\}$
15: $\mathbf{v} \leftarrow$ eigenvector corresponding to $\lambda_{max}(\mathbf{T})$
16: $\mathbf{u} \leftarrow \left(\frac{1}{|D|}\sum_{i=1}^{|D|} \mathrm{vec}(\mathbf{G}_i^\top \mathrm{reshape}(\mathbf{v})\mathbf{G}_i)\right)/\sigma$
17: $\mathbf{b} \leftarrow \mathbf{u} \cdot \sigma$ ▷ $\mathbf{b} = \mathrm{vec}(\mathbf{H_I})$
18: $\mathbf{a} \leftarrow \mathbf{v}$ ▷ $\mathbf{a} = \mathrm{vec}(\mathbf{H_O})$
19: $\mathbf{H_I} \leftarrow \mathrm{reshape}(\mathbf{b}, (m, m))$
20: $\mathbf{H_O} \leftarrow \mathrm{reshape}(\mathbf{a}, (n, n))$
21: **return** $(\mathbf{H_I}, \mathbf{H_O})$

In the following Theorem, we demonstrate that the Lanczos method converges faster in norm than Power iteration. Using LLaMA-2-7B as a case study, we show that in a real-world scenario, norm convergence can be replaced by pointwise convergence for the majority of LLM layers.

Theorem 3.1 (Convergence of power iteration and Lanczos in LLM Curvature Estimation)**.** *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be symmetric positive semidefinite with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ and define the normalized spectral gap $q = \frac{\lambda_2}{\lambda_1}, \quad 0 < q < 1, q \in (0, 1)$.*

Let v_k denote the k-th iterate of power iteration, and let w_k denote the Ritz vector obtained after k steps of the Lanczos method, both initialized with nonzero projection onto the leading eigenvector u_1. Then:

- *For all $k \geq 1$, the Lanczos error bound has strictly smaller $L^2(0, 1)$-norm:*

$$\|f_k\|_{L^2(0,1)} \leq 2^{k+1} \|g_k\|_{L^2(0,1)},$$

where $f_k(q) = \left(\frac{1-q}{1+q}\right)^k, g_k(q) = q^k$.

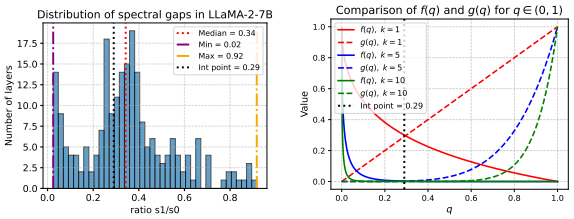
- *In empirical LLM Hessians, the Lanczos method also converges faster pointwise.*

Proof. **Step 1. Power iteration convergence.** The power iteration bound follows from standard spectral decomposition arguments (Golub and Van Loan, 2013; Trefethen and Bau, 1997).

$$\sin \angle(v_k, u_1) \leq C(v_0)\, q^k. \tag{4}$$

Step 2. Lanczos convergence. The Lanczos bound follows from the theory of Krylov subspace methods and Chebyshev polynomial approximation (Parlett, 1980; Saad, 2003).

$$\sin \angle(w_k, u_1) \leq 2 \left(\frac{1-q}{1+q}\right)^k. \tag{5}$$



(a) Distribution of spectral gaps. (b) Comparison of $f(q)$ and $g(q)$.

Figure 1: (a) Histogram of spectral gaps across layers in LLaMA-2-7B. In more than half of the layers, the spectral gap exceeds 0.29, and on these layers we observe that $f(q) < g(q)$. (b) Curves $f_k(q)$ and $g_k(q)$ for $k \in \{1, 5, 10\}$. As k increases, the curve $f^k(x)$ becomes tightly compressed against the horizontal axis, causing the area under it to shrink rapidly.

Step 3. Norm comparison. We now compare the convergence rates of the two methods by analyzing the L_2 norms of their respective error functions over $q \in (0, 1)$:

$$I_k = \|f_k\|^2_{L_2(0,1)} = \int_0^1 \left(\frac{1-\sqrt{q}}{1+\sqrt{q}}\right)^{2k} dq, \tag{6}$$

$$J_k = \|g_k\|^2_{L_2(0,1)} = \int_0^1 q^{2k}\, dq = \frac{1}{2k+1}. \tag{7}$$

With the substitutions $q = t^2$ and $u = \frac{1-t}{1+t}$, we obtain

$$I_k = 4 \int_0^1 u^{2k} \frac{1-u}{(1+u)^3}\, du. \tag{8}$$

Since $\frac{1}{(1+u)^3} \leq 1$ for $u \in (0, 1)$, it follows that

$$I_k \leq \frac{4}{(2k+1)(2k+2)} = \frac{2}{k+1} \underbrace{\frac{1}{2k+1}}_{J_k}. \tag{9}$$

Thus,

$$\|f_k\|_{L_2(0,1)} \leq \sqrt{\frac{2}{k+1}} \|g_k\|_{L_2(0,1)}. \tag{10}$$

Hence,

$$\|f_k\|_{L_2} - \|g_k\|_{L_2} \leq 0 \qquad \forall\, k \geq 1, \tag{11}$$

and the difference between the two norms increases as k grows. This can be seen by analyzing the area under the family of curves f and g in Figure 1(b).

Step 4. Practical pointwise convergence in LLM Figure 1(b) further indicates that the intersection point at which f becomes smaller than g occurs at approximately $q \approx 0.29$. Figure 1(a) shows the empirical distribution of q across layers of LLaMA-2-7B over one training epoch. Notably, most empirical values lie to the right of this intersection point, where the inequality $f_k(q) < g_k(q)$ is satisfied.

This implies that, for the majority of layers, Algorithm 1 converges faster than power iteration not only in the integral sense but also pointwise. □

After Hessian estimation, we perform the weight rounding algorithm. We employ vector quantization based on Trellis-Coded Quantization (TCQ), where the matrix is quantized iteratively and the current quantization step compensates for the quantization error of the previous ones. For a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and a quantization block of size $td_x \times td_y$, the quantization step k is determined using the error accumulated from the previous k-1 steps:

$$\begin{aligned}\mathbf{W}_k = \mathcal{Q}\big(&\mathbf{W}^\star_k + \mathbf{L_O}^\top \Delta\mathbf{W}_{k-1}\mathbf{L_I} \\ &+ \mathbf{L_O}^\top \Delta\mathbf{W}_{k-1} + \Delta\mathbf{W}_{k-1}\mathbf{L_I}\big)\end{aligned} \tag{12}$$

where $\mathbf{W}_k = \mathbf{W}[r \cdot td_y : (r+1) \cdot td_y, c \cdot td_x : (c+1) \cdot td_x]$ - weights quantized at the current step; $\Delta\mathbf{W}_{k-1} = \Delta\mathbf{W}[0 : (r+1) \cdot td_y, 0 : c \cdot td_x]$ - the error accumulated from the previous steps; $r = \lfloor k / \lceil n/td_x \rceil \rfloor$ is the row index of the current weight quantization block; $c = k \pmod{\lceil n/td_x \rceil}$ is the column index of the current weight quantization block.

Our TCQ implementation is based on the code provided in the QTIP repository[1]. All quantization hyperparameters and the full method pipeline are available on GitHub[2]. Classical TCQ does not scale well to large language models due to its high memory requirements and inherently sequential decoding, stemming from the need to store explicit trellis structures and codebooks. Using a bitshift trellis and on-the-fly generation of random Gaussian codes, the codebase employed addresses these limitations.

Prior to quantization, we transform the weight matrix to approximately follow a Gaussian distribution as preprocessing. This satisfies the multivariate normal (MVN) assumption in Eq. 2 and enables the use of QTIP's Fast Random Gaussian Codes for on-the-fly codebook generation.

For all methods compared below, the vector quantization algorithm, TCQ hyper-parameters, and preprocessing are kept consistent and unified — only the factors $\mathbf{H_I}, \mathbf{H_O}$ vary.

4 Experiments

To implement **FastKron**, we compute the Kronecker factors using Algorithm 1 and update the model weights according to Equation (3). We choose the number of calibration tokens so that factors obtained from a calibration dataset with a fixed size yield downstream performance comparable to the baselines.

As our primary baseline, we employ **YAQA**, which similarly utilizes a full Hessian approximation but differs in its algorithm for computing factor matrices. YAQA offers two calibration schemes: Sketch A (20M tokens) and Sketch B (128M tokens). To align with fast quantization objectives, we use Sketch A, as it performs within approximately 1% of Sketch B while requiring significantly less convergence time. For all YAQA experiments, we utilize the authors' official implementation[3].

We also compare our approach against methods using diagonal approximations. Specifically, the original **QTIP** method accounts for loss surface curvature using a diagonal Hessian approximation. To isolate the specific contribution of second-order information to compression quality, we include an additional baseline that employs TCQ but omits curvature data entirely (**NoHess** setup). We also conducted an ablation study to evaluate the significance of off-diagonal information by retaining only the diagonal elements of the Hessian factors computed via FastKron (**OnlyDiag** setup).

We quantized LLaMA-2-7B[4], LLaMA-3-8B (Instruct)[5], and the Qwen3[6] series (8B, 4B, 1.7B, and 0.6B) models. In all experiments, we performed one-shot quantization without any additional fine-tuning. Performance was measured via perplexity and zero-shot accuracy on common-sense reasoning benchmarks. To further account for reasoning and instruction-following capabilities, we additionally validated on GSM8K (Cobbe et al., 2021) and IFEval (Zhou et al., 2023) using LLaMA-3-8B. For the LLaMA models, we employed a sequence length of 4096 tokens, for Qwen, of 2048 tokens. Runtime metrics were captured using Python's built-in profiler.

5 Results and Discussion

The results for zero-shot downstream tasks for LLaMA-2-7B are shown in Table 1, for LLaMA-3-8B – in Table 2, for Qwen3 8B in Table 3. Results for Qwen models of different sizes are presented in Table 5.

Experimental results across all evaluated models demonstrate that FastKron matches or exceeds the quality of established curvature-aware methods, despite operating with a significantly lower budget for loss surface parameterization. A notable observation occurs within the LLaMA family: while FastKron slightly trails YAQA in terms of perplexity, it consistently delivers superior performance on downstream tasks. This discrepancy vanishes for the Qwen family, where FastKron yields better perplexity across all scales.

At 4-bit precision, FastKron and YAQA perform comparably; for instance, on Qwen-1.7B, FastKron shows a marginal gap of less than 0.1%. However, in the 2-bit regime, FastKron consistently outperforms baselines, with accuracy gains ranging from 0.3% (LLaMA-2) to 12.8% (Qwen-4B) across different architectures. This robust performance suggests that FastKron achieves better convergence, with the degree of improvement being highly sensitive to the specific model architecture and its underlying loss landscape.

[1] https://github.com/Cornell-RelaxML/qtip
[2] https://github.com/sayankotor/FastKron_QTIP
[3] https://github.com/Cornell-RelaxML/yaqa-quantization
[4] https://huggingface.co/meta-llama/Llama-2-7b
[5] https://huggingface.co/meta-Llama/Llama-3.1-8B-Instruct
[6] https://huggingface.co/Qwen

Table 1: Zero-shot accuracy for different quantization methods on **LLaMA-2-7B**. Lower is better for resource usage and perplexity (↓), higher is better for accuracy (↑). "M" denotes millions and "K" denotes thousands of tokens.

Method	Wiki↓	C4 ↓	Arc_c ↑	Boolq ↑	Piqa ↑	Arc_e ↑	HSwag ↑	AVG ↑	GPU h ↓	Tokens ↓
16 bit	**5.11**	**6.63**	**0.4325**	**0.7767**	**0.7774**	**0.7617**	**0.5721**	**0.6640**	–	–
4 bit NoHess	5.25	6.74	0.4270	0.7610	0.7669	0.7609	0.5628	0.656	–	0
4 bit QTIP	5.19	6.71	**0.4351**	0.7615	0.7708	0.7604	**0.5686**	0.659	3	16 M
4 bit YAQA	**5.17**	**6.69**	0.4274	0.7688	0.7752	**0.7613**	0.5672	0.659	50	16 M
4 bit FastKron	5.18	6.71	0.4283	**0.7792**	**0.7802**	0.7610	0.5660	**0.663** ↑0.6%	5	0.7 M ↓96%
2 bit NoHess	39.44	43.07	0.2210	0.6355	0.6306	0.5152	0.3422	0.469	–	0
2 bit QTIP	6.55	8.50	0.3677	0.7314	0.7502	0.7113	0.5064	0.613	3	33 M
2 bit YAQA	**6.18**	**8.00**	0.3805	0.7333	0.7562	**0.7192**	**0.5227**	0.622	50	16 M
2 bit FastKron	6.40	8.31	**0.3843**	**0.7510**	**0.7600**	0.7112	0.5144	**0.624**↑0.3%	6	1.4 M ↓92%

Table 2: Zero-shot accuracy for different quantization methods on **LLaMA-3-8B**. Lower is better for resource usage and perplexity (↓), higher is better for accuracy (↑). "M" denotes millions and "K" denotes thousands of tokens.

Method	Wiki↓	C4 ↓	Arc_c ↑	Boolq ↑	Piqa ↑	Arc_e ↑	HSwag ↑	AVG ↑	GPU h ↓	Tokens↓
16 bit	**5.11**	**6.63**	**0.5171**	**0.8409**	**0.7986**	**0.8177**	**0.5908**	**0.7131**	–	—
4 bit NoHess	7.11	10.28	0.5119	0.8415	0.7959	0.8097	0.5829	0.708	–	0
4 bit OnlyDiag	6.90	9.96	0.4958	0.8425	0.8020	0.8178	0.5856	0.708	–	0.7 M
4 bit QTIP	6.93	10.05	0.5136	0.8376	0.7959	0.8118	0.5852	0.708	3	16 M
4 bit YAQA	**5.17**	**6.69**	**0.5136**	**0.8443**	0.7997	0.8198	**0.5865**	0.712	92	16 M
4 bit FastKron	5.18	6.71	0.5116	0.8438	**0.8025**	**0.8207**	0.5863	**0.713**↑0.14%	9.5	0.7 M ↓96%
2 bit NoHess	44.79	51.10	0.2363	0.6336	0.6554	0.5108	0.3620	0.509	–	0
2 bit OnlyDiag	9.96	14.40	0.3800	0.7612	0.7503	0.7445	0.5048	0.632	–	0.9 M
2 bit QTIP	10.5	14.54	0.4002	**0.8175**	0.7492	0.7424	0.5015	0.642	3	16 M
2 bit YAQA	**8.98**	**12.79**	**0.4312**	0.7567	0.7647	0.7391	**0.5259**	0.643	92	16 M
2 bit FastKron	9.11	12.98	0.4277	0.8088	**0.7661**	**0.7468**	0.5159	**0.653**↑1.6%	11.5	0.9 M ↓94%

Furthermore, the results lead to the following conclusions. The first conclusion reflects the importance of curvature information in LLM compression. Comparing FastKron, which incorporates curvature information, with NoHess, which performs quantization without it, shows that the contribution of second-order information becomes increasingly critical at higher compression levels across all evaluated models.

At 4-bit precision, the performance gap between curvature-aware and curvature-agnostic methods remains narrow, with accuracy declining by only 1–3% across all tested architectures. However, at 2-bit quantization, the model sensitivity to curvature information becomes far more pronounced. Under the NoHess configuration, LLaMA models exhibit an average accuracy drop of 14–16% compared to full-factor compression, while Qwen models show a decrease of 8–12%, depending on the model scale.

The second conclusion highlights the importance of off-diagonal Hessian components. We compare the rows in Tables 1–3 corresponding to QTIP, which employs a diagonal Hessian approximation, with those of FastKron, which incorporates the full curvature information. While QTIP can yield the best performance on individual commonsense benchmarks for certain architectures – for example, on HellaSwag for 4-bit LLaMA-2 – on average it is 0.5–2% worse than FastKron.

As expected, OnlyDiag has lower performance than the full FastKron approach but still outperforms the NoHess baseline. The most comparable setup is QTIP, which utilizes one diagonal factor and one identity matrix. While QTIP shows slightly better results, it requires a significantly larger number of calibration tokens.

The next aspect concerns behavior on reasoning and instruction-following benchmarks. The Table 4 presents the results for the GSM8K and IFEval benchmarks. It demonstrates that with 4-bit quantization, the model almost entirely preserves its reasoning and instruction-following capabilities. While a performance drop on GSM8K is observed in 2-bit quantization, the YAQA and FastKron methods perform notably better at maintaining quality compared to QTIP.

This performance drop is consistent with recent findings that contextual understanding in transformer models relies on abnormally large activations at specific layer locations (Jin et al., 2025). Preserving these activations is critical for maintain-

Table 3: Zero-shot accuracy for different quantization methods on **Qwen3-8B**. Lower is better for resource usage and perplexity (↓), higher is better for accuracy (↑).

Method	Wiki↓	C4 ↓	Arc_c ↑	Boolq ↑	Piqa ↑	Arc_e ↑	HSwag ↑	AVG ↑	GPU h ↓	Tokens↓
16 bit	**8.99**	**12.48**	**0.5563**	**0.8682**	**0.7677**	**0.8354**	**0.5708**	**0.7197**		
4 bit NoHess	9.45	13.05	0.5417	0.8604	0.4659	0.8312	0.5585	0.711	–	0
4 bit QTIP	9.27	12.73	0.5471	0.8629	0.7598	**0.8329**	0.5605	0.712	3	16 M
4 bit YAQA	9.29	12.72	**0.5503**	0.8611	**0.7612**	0.8324	0.5601	**0.713**	84	16 M
4 bit FastKron	**9.16**	**12.66**	0.5469	**0.8667**	0.7601	0.8287	**0.5637**	**0.713** 0%	42	0.7 M ↓94%
2 bit NoHess	43.00	51.74	0.2713	0.6988	0.6447	0.7003	0.3771	0.538	–	0
2 bit QTIP	17.32	20.20	0.4263	0.8101	0.7427	0.7511	0.4575	0.638	3	16 M
2 bit YAQA	16.04	18.21	0.4536	0.7782	**0.7435**	**0.7797**	0.4611	0.643	84	16 M
2 bit FastKron	**13.36**	**16.86**	**0.4616**	**0.8416**	0.7334	0.7702	**0.4853**	**0.658** ↑2.3%	42	0.7 M ↓94%

Table 4: Results on **LLaMA-3-8B** for reasoning and instruction-following benchmarks GSM8K and IFEval. Lower is better for resource usage (↓), higher is better for accuracy (↑). "M" denotes millions and "K" denotes thousands of tokens.

Method	GSM8K ↑	IFEval ↑	Tokens ↓
16 bit	**0.7498**	**0.7654**	
4 bit QTIP	0.7172	0.7412	16 M
4 bit YAQA	0.7361	**0.7653**	16 M
4 bit FastKron	**0.7385**	0.7612	0.7 M
2 bit QTIP	0.3495	0.5600	16 M
2 bit YAQA	**0.4546**	0.6561	16 M
2 bit FastKron	0.4537	**0.6577**	0.9 M

ing the model's contextual reasoning ability, yet none of the methods considered here are activation-aware — making degradation at extremely low bit-widths expected.

As model compression is ultimately motivated by deployment efficiency, we report inference speed measurements in Table 6.

6 Impact of Calibration Dataset Size

In the previous section, we showed that FastKron achieves comparable downstream performance with a substantially smaller budget due to faster Hessian parameterization. This raises the question of how sensitive FastKron's downstream performance is to the number of tokens used during calibration, and how quickly the performance of the baseline method, YAQA, degrades as the calibration token budget is reduced.

Figure 2 illustrates the relationship between the calibration token count and average performance on commonsense benchmarks for the 2-bit version of LLaMA-2-7B. For each data point, Kronecker factors were computed from a calibration dataset of a given size and then applied within a fixed quantization pipeline. The complete results for perplexity and zero-shot evaluation are provided in Appendix A.

YAQA degrades monotonically as the calibration budget is reduced. In contrast, FastKron reaches its peak performance at approximately 1.4 M tokens and begins to decline thereafter, with a monotonic decrease observed starting around 712 K tokens. Note that the x-axis is shown on a logarithmic scale.

We attribute this non-monotonic behavior to a fundamental property of our algorithm: FastKron computes the SVD operation per mini-batch individually and averages the results across the calibration set. When the number of mini-batches becomes excessively large, this procedure introduces an estimation bias that becomes particularly pronounced at ultra-low bit-widths.

7 Conclusion

In this work, we introduced **FastKron**, an efficient Kronecker-factored approach for incorporating second-order curvature information into post-training quantization of LLMs. It preserves the structure of existing quantization algorithms while providing significantly faster factor computation — theoretically grounded for the spectral regime characteristic of LLM Hessians — and achieves up to a 10× empirical speedup over methods in the same category. FastKron attains the same downstream accuracy as the power iteration baseline, maintaining nearly identical quality at 4-bit compression and incurring only a 5–6% performance drop under 2-bit quantization.

Our experimental results across the LLaMA and Qwen model families further show that second-order information becomes increasingly important as model compression becomes more aggressive. This observation indicates that highly compressed regimes require explicit and efficient parameterization of the Hessian. FastKron addresses this need by delivering the benefits of full second-order meth-

Table 5: Perplexity and zero-shot accuracy for various bit quantizations of Qwen3-0.6B, Qwen3-1.7B and Qwen3-4B. Method YAQA needs 16 M tokens to converge, FastKron - 0.7 M. Lower is better for Perplexity (↓), higher is better for accuracy (↑).

Model	Method	Wiki↓	C4↓	Arc_c↑	Boolq↑	Piqa↑	Arc_e↑	HSwag↑	AVG↑	Tokens↓
0.6B	16 bit	**19.16**	**23.83**	**0.3413**	**0.5585**	**0.6404**	**0.4732**	**0.6746**	**0.537**	
	4 bit FastKron	**20.21**	**24.75**	**0.3191**	**0.5227**	0.6214	**0.4591**	**0.6725**	**0.519**↑1.0%	0.7 M↓94%
	4 bit YAQA	21.05	25.23	0.3046	0.5135	**0.6358**	0.4538	0.6643	0.514	16 M
	4 bit NoHess	23.90	28.85	0.3131	0.4971	0.5627	0.4362	0.6627	0.494	0
	2 bit FastKron	**63.55**	**64.34**	0.2312	**0.3695**	**0.6171**	**0.3354**	**0.5702**	**0.425**↑4.4%	0.7 M↓94%
	2 bit YAQA	88.65	94.24	0.2244	0.3359	0.5988	0.3182	0.5622	0.407	16 M
	2 bit NoHess	41292	46261	**0.2568**	0.2917	0.4615	0.2605	0.5223	0.358	0
1.7B	16 bit	**15.66**	**18.13**	**0.4317**	**0.6957**	**0.7761**	**0.6033**	**0.7220**	**0.645**	
	4 bit FastKron	**16.24**	**18.60**	0.4113	0.6721	0.7728	**0.5927**	**0.7122**	0.632↓0.6%	0.7 M↓94%
	4 bit YAQA	17.23	19.05	**0.4224**	**0.6864**	**0.7765**	0.5873	0.7084	**0.636**	16 M
	4 bit NoHess	17.70	20.92	0.3771	0.5623	0.7511	0.5614	0.6986	0.590	0
	2 bit FastKron	**28.98**	**32.12**	**0.2841**	**0.4920**	**0.6716**	**0.4487**	**0.6491**	**0.509**↑9.7%	0.7 M↓94%
	2 bit YAQA	38.18	40.51	0.2645	0.4162	0.6489	0.3862	0.6088	0.464	16 M
	2 bit NoHess	54399	36222	0.2534	0.2795	0.6061	0.2799	0.5381	0.391	0
4B	16 bit	**12.43**	**15.45**	**0.5350**	**0.7841**	**0.8502**	**0.6838**	**0.7497**	**0.720**	
	4 bit FastKron	12.91	15.77	**0.5307**	**0.7816**	0.8492	**0.6740**	0.7486	**0.717** 0%	0.7 M↓94%
	4 bit YAQA	**12.54**	**15.64**	0.5290	0.7765	**0.8514**	0.6727	**0.7563**	**0.717**	16 M
	4 bit NoHess	13.12	16.17	0.4906	0.7269	0.8446	0.6693	0.7432	0.695	0
	2 bit FastKron	18.56	21.49	**0.4070**	**0.6970**	**0.8015**	**0.5641**	**0.7057**	**0.635**↑12.8%	0.7 M↓94%
	2 bit YAQA	**16.85**	**20.45**	0.3567	0.5787	0.6661	0.5246	0.6915	0.563	16 M
	2 bit NoHess	91.61	85.49	0.2995	0.4600	0.6327	0.4391	0.6300	0.492	0

Table 6: Inference speed comparison across precisions and sequence lengths.

Seq Len	Bit	Total Latency (ms)	Latency per Token (ms)
1024	2	1225.82	1.19
	4	1492.65	1.45
2048	2	1816.42	0.88
	4	2088.50	1.02

ods at a substantially lower cost, making curvature-aware quantization both practical and scalable. Notably, the proposed factor computation serves as a drop-in improvement for existing quantization pipelines.

8 Ethics statement (Potential Risks)

This work focuses on methods for improving the efficiency and practicality of post-training quantization of large language models. Our research does not involve human subjects, personally identifiable information, or sensitive data. All experiments are conducted on publicly available models (LLaMA, Qwen) and benchmarks (ARC, BoolQ, PIQA, HellaSwag, WikiText, C4), ensuring reproducibility and transparency. We do not release any new datasets containing personal or private data. The proposed methods are intended for reducing the computational cost and energy consumption of deploying large models, which we view as a positive contribution to sustainability. We are not aware of any direct negative societal impacts, though—as with any model compression technique—improved efficiency could indirectly facilitate the deployment of large models in settings where misuse is possible.

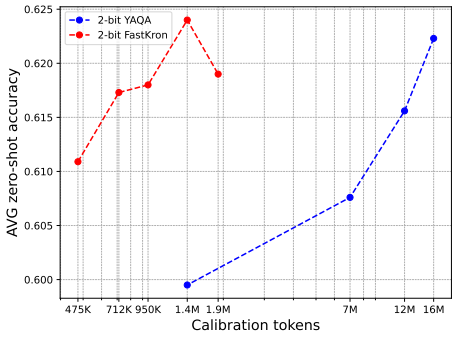


Figure 2: Average zero-shot accuracy versus the number of calibration tokens (logarithmic scale on the x-axis) for 2-bit quantization of LLaMA-2-7B. FastKron consistently outperforms YAQA at comparable or smaller calibration budgets.

Reproducibility Statement

We have made every effort to ensure that our results are fully reproducible. The implementation of the proposed method is provided in the supple-

mentary materials. All theoretical assumptions and proofs are included in the main text. Links to the baseline repositories required to reproduce our experiments are provided in Section 4. Additionally, all model checkpoints used in our experiments will be released publicly in the camera-ready version of the paper.

Limitations

A key limitation of the current FastKron implementation is the need to store gradients for all mini-batches when computing Kronecker factors. For a single LLaMA layer 100 micro-batches of dataset D (which corresponds to 950 K tokens) already imposes 5.4 GB of additional memory. This stems from the fact that the core operation in the Lanczos algorithm—left and right multiplication by a vector—cannot be interchanged with minibatch averaging without degrading downstream performance. Consequently, there is an inherent trade-off: faster computation comes at the cost of higher memory consumption.

Acknowledgements

This work was supported by the The Ministry of Economic Development of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000C313925P4H0002; grant No 139-15-2025-012).

The authors thank Denis Kuznedelev for his invaluable guidance on quantization algorithms, and Albert Tseng for his support in reproducing the Model-Preserving Adaptive Rounding approach.

References

Viktoriia Chekalina, Daniil Moskovskiy, Daria Cherniuk, Maxim Kurkin, Andrey Kuznetsov, and Evgeny Frolov. 2025. Generalized fisher-weighted svd: Scalable kronecker-factored fisher approximation for compressing large language models. *Preprint*, arXiv:2505.17974.

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

Vadim Egorov, James Hooper, Dan Alistarh, Denis Kuznedelev, and Artem Babenko. 2024. Extreme compression of large language models via additive quantization. In *International Conference on Learning Representations (ICLR)*.

Runa Eschenhagen, Alexander Immer, Richard Turner, Frank Schneider, and Philipp Hennig. 2023. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36:33624–33655.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*.

Gene H. Golub and Charles F. Van Loan. 2013. *Matrix Computations*, 4 edition. Johns Hopkins University Press, Baltimore, MD.

Yen-Chang Hsu, Ting Hua, Sung-En Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112*.

Ting Hua, Yen-Chang Hsu, and 1 others. 2022. Numerical optimizations for weighted low-rank estimation: The case of transformer language model compression. In *Proceedings of EMNLP*.

Mingyu Jin, Kai Mei, Wujiang Xu, Mingjie Sun, Ruixiang Tang, Mengnan Du, Zirui Liu, and Yongfeng Zhang. 2025. Massive values in self-attention modules are the key to contextual knowledge understanding. *arXiv preprint arXiv:2502.01563*.

Cornelius Lanczos. 1950. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282.

Charles Van Loan and Nikos Pitsianis. 1992. Approximation with kronecker products.

Michael W Marcellin and Thomas R Fischer. 1990. Trellis coded quantization. *IEEE Transactions on Communications*, 38(1):82–93.

James Martens and Roger Grosse. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*.

Beresford N. Parlett. 1980. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, NJ.

Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin A. Raffel, Leandro von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems*, 2nd edition. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Lloyd N. Trefethen and David Bau. 1997. *Numerical Linear Algebra*. SIAM, Philadelphia, PA.

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

Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. 2025a. Qtip: Quantization with trellises and incoherence processing. *Preprint*, arXiv:2406.11235.

Albert Tseng, Zhaofeng Sun, and Christopher De Sa. 2025b. Model-preserving adaptive rounding. *Preprint*, arXiv:2505.22988.

Changbao Wang, Dandan Zheng, Yuanliu Liu, and Liang Li. 2022. Leveraging inter-layer dependency for post-training quantization. In *Advances in Neural Information Processing Systems*, volume 35, pages 6818–6831.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jichao Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yosuang Wang, and Kurt Keutzer. 2020. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 18518–18529.

Jeffrey Zhou, Tianhao Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Table 7: Perplexity and zero-shot accuracy for 2-bit FastKron quantization of LLaMA-2-7B, comparing factors derived via power YAQA. Lower is better for Perplexity (↓), higher is better for accuracy (↑).

Method	Wiki↓	C4↓	Arc_c↑	Boolq↑	Piqa↑	Arc_e↑	HSwag↑	AVG↑	Steps	Tokens
16 bit	**5.11**	**6.63**	**0.4325**	**0.7767**	**0.7774**	**0.7617**	**0.5721**	**0.6640**	–	–
2 bit YAQA	6.18	8.00	0.3805	0.7333	0.7562	0.7192	0.5227	0.6223	–	16 M
2 bit FastKron	6.59	8.69	0.3599	0.7192	0.7473	0.7030	0.5116	0.6082	35	330K
2 bit FastKron	6.44	8.40	0.3658	0.7152	0.7568	0.7032	0.5135	0.6109	50	475K
2 bit FastKron	6.44	8.30	0.3720	0.7320	0.7579	0.7112	0.5137	0.6173	70	712K
2 bit FastKron	**6.43**	8.36	0.3677	0.7393	0.7578	0.7128	0.5127	0.6180	100	950K
2 bit FastKron	6.40	**8.31**	**0.3843**	**0.7510**	**0.7600**	**0.7112**	**0.5139**	**0.6240**	150	1400K
2 bit FastKron	6.47	8.39	0.3618	0.7486	0.7540	0.7115	0.5091	0.6190	200	1900K

Table 8: Perplexity and zero-shot accuracy for 2-bit YAQA quantization of LLaMA-2-7B under different calibration token budgets (Sketch A). Lower is better for Perplexity (↓), higher is better for accuracy (↑).

Method	Wiki↓	C4↓	AVG↑	Arc_c↑	Arc_e↑	Boolq↑	Piqa↑	HSwag↑	Tokens
16 bit	**5.11**	**6.63**	**0.6640**	**0.4325**	**0.7617**	**0.7767**	**0.7774**	**0.5721**	–
2 bit YAQA	6.18	8.00	0.6223	0.3805	0.7191	0.7333	0.7562	0.5227	16 M
2 bit YAQA	–	–	0.6156	0.3762	0.7175	0.7186	0.7538	0.5121	12 M
2 bit YAQA	6.19	8.38	0.6076	0.3626	0.7083	0.7011	0.7513	0.5145	7 M
2 bit YAQA	6.24	8.97	0.5995	0.3443	0.7008	0.7002	0.7435	0.5090	1.4 M

A Appendix A

We report extended quantization results for Token Count ablation in Tables 7, 8.

B Appendix B: LLM Usage Statement

We used large language models (LLMs) only as a general-purpose writing assistant for grammar checking and text polishing. The research ideas, implementation, analysis, and conclusions are entirely our own.