CSQUANT: CROSS-BLOCK SMOOTHING FOR ACCURATE AND EFFICIENT LOW-BIT LLM QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-bit quantization is essential for efficient large language model inference, but aggressive quantization remains vulnerable to activation outliers that amplify quantization error. Orthogonally rotating activations alleviates quantization error but incurs additional online computation, particularly in adaptive methods that use intermediate permutations, which prevent multiple rotations from being combined and introduce explicit data movement. We propose **CSQuant**, an accurate and efficient post-training quantization framework comprising two key modules: **Cross-Block Smoothing** and **LiteBoA**. Cross-Block Smoothing replaces intermediate permutations with a normalized cross-block Hadamard transform, enabling cross-block outlier redistribution while allowing two adaptive rotations to be merged into a single online transformation. This separable structure further enables CUDA kernel fusion with adjacent operations, reducing intermediate tensor materialization and global-memory traffic. LiteBoA selectively applies attention-aware second-order compensation to value projection matrices and employs head-shared Hessians to reduce calibration cost while preserving quantization accuracy. Experiments on LLaMA-1/2/3 demonstrate improved quantization accuracy across W4A4KV4 and weight-only settings, with particularly strong performance at 2-bit weights. CSQuant achieves a 2.25× prefill speedup over FP16 under W4A4KV4 quantization with a sequence length of 2048 and a batch size of 4, while our fusion strategy further accelerates existing Hadamard-based rotation methods. CSQuant also substantially reduces peak GPU memory during post-training quantization.

1 INTRODUCTION

Large language models (LLMs) (Le Scao et al., 2022; Radford et al., 2019; Devlin et al., 2019; Liu et al., 2019) have demonstrated remarkable capabilities across a wide range of applications, but their substantial computation and memory requirements remain a major obstacle to efficient deployment. Low-bit quantization has therefore become one of the most practical approaches for reducing model size and accelerating inference. However, aggressively quantizing both weights and activations remains challenging, primarily because activation distributions in LLMs are highly non-uniform and often contain large outliers (Xiao et al., 2023). These extreme values enlarge the quantization range and substantially increase the rounding error of low-bit representations.

To address this challenge, rotation-based quantization methods exploit rotational invariance to redistribute activation outliers while preserving the original full-precision computation. Existing methods can be broadly divided into two categories: fusion-based rotations and layer-adaptive online rotations. Fusion-based methods leverage Transformer computational invariances (Ashkboos et al., 2024a; Vaswani et al., 2017) to absorb globally coordinated rotations into model weights or cancel them between adjacent operators (Ashkboos et al., 2024b; Liu et al., 2025b; Hu et al., 2025). However, this requires shared or coordinated rotations across coupled layers, limiting layer-specific adaptation. In contrast, layer-adaptive online methods, such as DuQuant (Lin et al., 2024) and RUQuant (Liu et al., 2026), construct layer-specific rotations from calibration activations, enabling training-free adaptation to layer-wise outliers. However, these rotations generally cannot be absorbed offline and must be executed online, introducing additional inference overhead.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

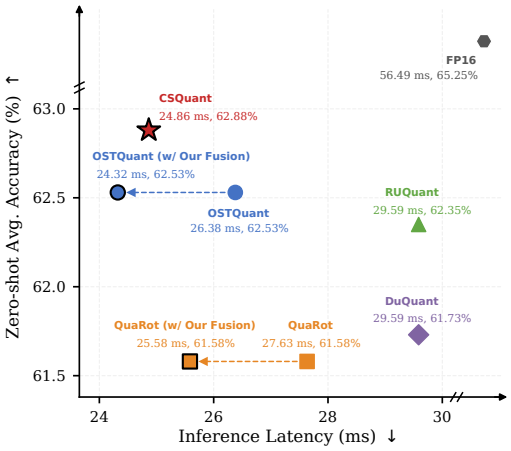

Figure 1: Accuracy and efficiency trade-off on LLaMA-2 7B under W4A4KV4. Inference latency is measured for a single Transformer block with batch size 4 and sequence length 2048.

These observations motivate a natural question: *Can we retain the quantization accuracy benefits of layer-adaptive online rotations while achieving the inference efficiency of fusion-based methods?* We observe that a major source of inference overhead in existing online rotation methods lies in their rotation-permutation-rotation (RPR) pipeline, which combines compact block-shared rotations with cross-block permutation. Although this pipeline reduces the computational cost of online rotations, the intermediate permutation prevents the two rotations from being merged and requires explicit data movement, resulting in non-negligible inference overhead. To address this limitation, we propose **CSQuant** [1], an accurate and efficient framework for low-bit LLM quantization comprising two key modules: **Cross-Block Smoothing** and **LiteBoA**. Cross-Block Smoothing enables the two local rotations in the RPR pipeline to be merged while supporting efficient kernel fusion, whereas LiteBoA selectively applies attention-aware second-order compensation to value projection matrices and employs head-shared Hessians to reduce calibration cost while preserving quantization accuracy. For Cross-Block Smoothing, we replace the permutation in the conventional RPR pipeline with a normalized cross-block Hadamard transform. The cross-block transform mixes the same coordinate across different local blocks, while the adaptive rotation operates within each block. Because these two transformations act on complementary dimensions, they commute exactly, allowing the two local rotations surrounding the Hadamard transform to be merged into a single adaptive rotation. More importantly, this algebraic structure enables cross-block mixing to be fused with neighboring operations. We develop memory-efficient CUDA kernels that fuse it with INT4 quantization and packing before quantized linear layers, and with SiLU and gating multiplication before `down_proj`. These fused schedules reduce intermediate activation materialization and global-memory traffic, substantially lowering online transformation overhead. We further find that the online full-width Hadamard transforms used by methods such as QuaRot (Ashkboos et al., 2024b) admit a similar factorized execution structure. By evaluating the power-of-two factor with a fast Hadamard transform and fusing it with neighboring memory-bound operations, the same execution principle can reduce the overhead of existing online Hadamard transformations.

Beyond improving inference efficiency, CSQuant further enhances quantization accuracy through attention-aware second-order weight quantization. BoA (Kim et al., 2025) extends GPTQ (Frantar et al., 2022) by incorporating downstream attention dependencies into the second-order quantization objectives for Query, Key, and Value projections, improving quantization accuracy at the cost of increased calibration time and memory usage. However, we observe that after Cross-Block Smoothing, the BoA-specific objectives for Query and Key provide negligible additional accuracy gains, while the attention-aware objective for Value remains effective. Motivated by this observation, we introduce **LiteBoA**, which applies standard GPTQ to Query and Key projections while retaining attention-aware second-order compensation for Value projections. Furthermore, LiteBoA shares a single Value Hessian across attention heads, reducing Hessian storage and calibration overhead. This selective design preserves the accuracy benefits of attention-aware Value quantization while substantially reducing quantization time and peak memory usage.

As illustrated in Figure 1, CSQuant achieves a favorable accuracy and efficiency trade-off, attaining the highest average zero-shot accuracy among the evaluated quantized methods with a single-block prefill latency of only 24.86 ms. Moreover, our fusion strategy (w/ our fusion) generalizes to existing Hadamard-based rotation methods that retain an online transform before `down_proj`, reducing the prefill latency of QuaRot from 27.63 ms to 25.58 ms and that of OSTQuant from 26.38 ms to 24.32 ms without affecting their accuracy. These results demonstrate that our approach combines

[1]The source code is available at: https://anonymous.4open.science/r/CSQUANT

accurate low-bit quantization with efficient inference while extending its execution benefits to existing rotation-based methods. Our main contributions are summarized as follows:

- We propose **Cross-Block Smoothing**, which replaces the permutation in the conventional RPR pipeline with a normalized cross-block Hadamard transform, allowing two adaptive local rotations to be exactly merged into a single online transformation. The rotations are constructed analytically without gradient-based optimization. We further develop fused CUDA kernels to reduce online transformation overhead and extend the underlying execution principle to existing Hadamard-based rotation methods.
- We introduce **LiteBoA**, which retains standard GPTQ Hessians for Query and Key, applies attention-aware curvature only to Value, and shares a single Value Hessian across attention heads, substantially reducing calibration time and memory usage while preserving quantization accuracy.
- Extensive experiments on LLaMA-1/2/3 demonstrate that CSQuant improves quantization accuracy under W4A4KV4, W4A16, W3A16, and W2A16 settings, with particularly strong robustness at 2-bit weights. CSQuant also achieves competitive prefill efficiency while substantially reducing peak GPU memory during post-training quantization.

2 Related Work

Large language model quantization reduces the precision of model parameters and intermediate representations to lower memory and computational costs (Zhu et al., 2023). Existing methods are broadly categorized into quantization-aware training (QAT) and post-training quantization (PTQ): QAT (Liu et al., 2024) adapts the model to quantization errors through additional training or fine-tuning, whereas PTQ (Nagel et al., 2020; Liu et al., 2025a) quantizes pretrained models with limited calibration data and without extensive retraining. Due to its lower adaptation cost, PTQ is widely adopted for efficient LLM inference and is the focus of this work. PTQ methods are broadly divided into weight-only and weight-activation quantization.

Weight-only quantization. Weight-only quantization reduces model storage and memory bandwidth while keeping activations in higher precision (Li et al., 2025; Zhang & Shrivastava, 2025). GPTQ (Frantar et al., 2022) uses approximate Hessian information for sequential error compensation, while AWQ (Lin et al., 2023) protects salient weight channels based on activation statistics. SpQR (Dettmers et al., 2024) preserves outlier weights in higher precision, whereas SqueezeLLM (Kim et al., 2024) combines sensitivity-aware non-uniform quantization with sparse outlier preservation. QuIP (Chee et al., 2023) improves ultra-low-bit quantization through incoherence processing, and QuIP# (Tseng et al.) further incorporates randomized Hadamard transformations and lattice-based codebooks. AQLM (Egiazarian et al., 2024) enhances compression through additive vector quantization. BoA (Kim et al., 2025) extends GPTQ by incorporating downstream attention dependencies into the Hessian of Query, Key, and Value projections, improving attention-aware quantization at the cost of higher calibration time and memory.

Weight–activation quantization. Joint weight-activation quantization is more challenging because activation outliers enlarge the quantization range and increase low-bit error (Shao et al., 2024). SmoothQuant (Xiao et al., 2023) mitigates this issue through channel-wise rescaling that shifts quantization difficulty from activations to weights. OmniQuant (Shao et al., 2024) further optimizes quantization parameters through learnable clipping and equivalent transformations, while RPTQ (Yuan et al., 2023) reorders and groups activation channels with similar ranges for finer-grained quantization. QServe (Lin et al., 2025) combines progressive quantization with system-level optimizations to reduce low-bit inference overhead. Rotation-based methods further reshape activation distributions with orthogonal transformations while preserving full-precision equivalence (Ashkboos et al., 2024b; Liu et al., 2025b). QuaRot (Ashkboos et al., 2024b) adopts randomized Hadamard rotations, while SpinQuant (Liu et al., 2025b) and OSTQuant (Hu et al., 2025) optimize globally coordinated rotations. In contrast, DuQuant (Lin et al., 2024) and RUQuant (Liu et al., 2026) construct layer-specific adaptive rotations from calibration activations, improving distribution adaptation at the cost of additional online transformations and channel redistribution. Therefore, effectively adapting activation distributions while minimizing the online overhead of layer-specific transformations remains an important challenge.

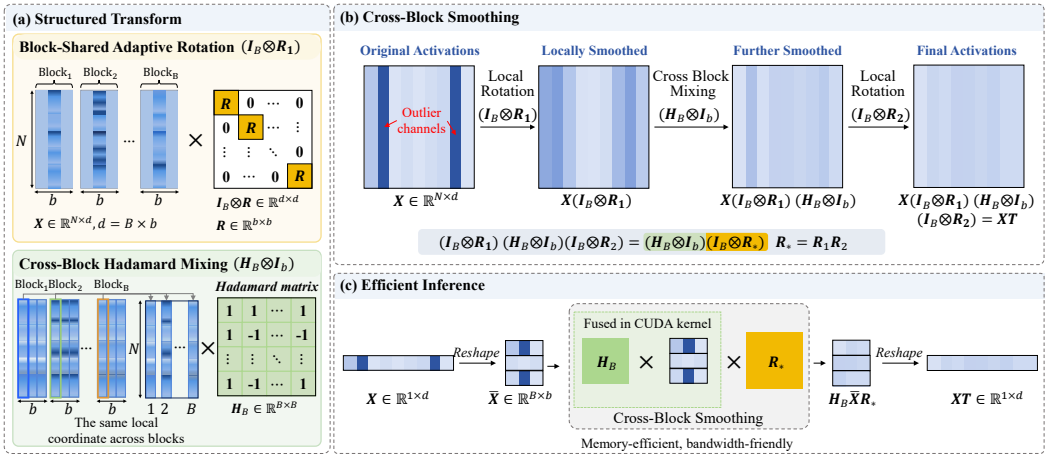

Figure 2: Overview of Cross-Block Smoothing and its efficient execution. (a) Block-shared local rotations smooth within-block activation outliers, while cross-block Hadamard mixing redistributes residual outliers across blocks. (b) The resulting structured transform enables the two local rotations surrounding the cross-block mixing to be exactly merged into a single rotation. (c) During inference, cross-block mixing is fused with neighboring CUDA operations, while the merged local rotation is executed as a standard GEMM.

3 METHOD

CSQuant consists of two key modules: Cross-Block Smoothing and LiteBoA. Before detailing them, we first introduce the basic notation. For a linear layer $\mathbf{Y} = \mathbf{X}\mathbf{W}^\top$, let $\mathbf{X} \in \mathbb{R}^{N\times d}$ denote the input activations and $\mathbf{W} \in \mathbb{R}^{m\times d}$ denote the weight matrix, where N is the number of input tokens, and d and m are the input and output dimensions, respectively. We apply an orthogonal transformation $\mathbf{T} \in \mathbb{R}^{d\times d}$ to both activations and weights:

$$\mathbf{X}\mathbf{W}^\top = (\mathbf{X}\mathbf{T})(\mathbf{W}\mathbf{T})^\top, \qquad \mathbf{T}^\top\mathbf{T} = \mathbf{I}. \tag{1}$$

Quantization is then performed as $\widehat{\mathbf{Y}} = \mathcal{Q}_a(\mathbf{X}\mathbf{T})\mathcal{Q}_w(\mathbf{W}\mathbf{T})^\top$, where $\mathcal{Q}_a(\cdot)$ and $\mathcal{Q}_w(\cdot)$ denote activation and weight quantization, respectively.

3.1 CROSS-BLOCK SMOOTHING

Cross-Block Smoothing consists of two components: a block-shared adaptive rotation for suppressing within-block outliers, and a cross-block Hadamard mixing transform for redistributing residual outliers across blocks. The overall workflow is illustrated in Figure 2.

Block-shared adaptive rotation. We start from the block-shared adaptive rotation used in RUQuant (Liu et al., 2026). Given hidden dimension d, the channels are partitioned into B blocks of width b, where $d = Bb$. A shared data-adaptive orthogonal matrix $\mathbf{R} \in \mathbb{R}^{b\times b}$ is applied independently to all blocks, corresponding to $\mathbf{I}_B \otimes \mathbf{R}$ in the original d-dimensional representation, where $\otimes$ denotes the Kronecker product. Following RUQuant, $\mathbf{R}$ is constructed from calibration activations $\mathbf{X} \in \mathbb{R}^{N\times d}$. After partitioning each sample into B blocks of width b, we reshape $\mathbf{X}$ into local vectors $\bar{\mathbf{X}} \in \mathbb{R}^{NB\times b}$ and iteratively apply

$$\bar{\mathbf{X}}^{(i)} = \bar{\mathbf{X}}^{(i-1)}\mathbf{Q}_i, \qquad \mathbf{R} = \prod_{i=1}^{N_r}\mathbf{Q}_i, \tag{2}$$

where $N_r = 16$ is the number of rotation iterations and each $\mathbf{Q}_i$ is orthogonal. At each step, the local vector with the largest current peak magnitude is selected, and the corresponding transformation rotates it toward an equal-norm target with a smoother coordinate distribution.

The shared local rotation effectively suppresses within-block outliers but does not exchange information across blocks, leaving residual scale imbalance after local smoothing. DuQuant (Lin et al., 2024)

and RUQuant (Liu et al., 2026) address this limitation with a zigzag permutation that redistributes channels across blocks before a second local adaptive rotation, yielding

$$\mathbf{T} = (\mathbf{I}_B \otimes \mathbf{R}_1)\mathbf{P}(\mathbf{I}_B \otimes \mathbf{R}_2), \tag{3}$$

where $\mathbf{P}$ denotes the permutation. Although permutation improves cross-block redistribution, it has an unfavorable execution structure. First, a general $\mathbf{P}$ does not commute with block-shared rotations, preventing the two local rotations from being merged. Second, explicit channel rearrangement introduces additional memory traffic and irregular accesses. We therefore replace permutation with a structured orthogonal transform that directly mixes corresponding coordinates across blocks.

Cross-block Hadamard mixing. For simplicity, we describe the transformation for a single activation vector $\mathbf{X} \in \mathbb{R}^{1\times d}$. We partition its channel dimension into B blocks of width b. For each within-block coordinate $i \in \{1, \dots, b\}$, we collect the corresponding entries from all blocks into

$$\mathbf{x}_i = [\mathbf{X}_{1,i} \quad \mathbf{X}_{2,i} \quad \cdots \quad \mathbf{X}_{B,i}]^\top \in \mathbb{R}^B, \tag{4}$$

where $\mathbf{X}_{j,i}$ denotes the i-th coordinate of the j-th block. We then apply a normalized Walsh–Hadamard matrix $\mathbf{H}_B \in \mathbb{R}^{B\times B}$:

$$\mathbf{x}_i \longmapsto \mathbf{H}_B\mathbf{x}_i, \qquad i = 1, \dots, b. \tag{5}$$

In this way, values at the same within-block coordinate are mixed across all blocks. To support an efficient fast Hadamard transform, we choose the number of blocks to be a power of two. Using a nominal local block width $b_0 = 128$, we select

$$B = \arg\min_{B' \in \mathcal{B}_d} \left| \frac{d}{B'} - b_0 \right|, \qquad \mathcal{B}_d = \left\{ 2^k : k \in \mathbb{Z}_{>0},\ 2^k \mid d \right\}, \tag{6}$$

and set $b = d/B$. This keeps the local rotation dimension close to b_0, balancing smoothing effectiveness and computational cost.

Commutativity and rotation merging. The cross-block Hadamard mixing is represented by $\mathbf{H}_B \otimes \mathbf{I}_b$, while the shared local rotation is represented by $\mathbf{I}_B \otimes \mathbf{R}_k$. By the mixed-product property of the Kronecker product,

$$(\mathbf{H}_B \otimes \mathbf{I}_b)(\mathbf{I}_B \otimes \mathbf{R}_k) = \mathbf{H}_B \otimes \mathbf{R}_k = (\mathbf{I}_B \otimes \mathbf{R}_k)(\mathbf{H}_B \otimes \mathbf{I}_b), \tag{7}$$

so the two transformations commute exactly. During calibration, we retain the rotation–mixing–rotation procedure: the first adaptive rotation suppresses local outliers, cross-block mixing redistributes the remaining block-level imbalance, and the second rotation further adapts the mixed representation. Using the commutativity above, the overall transformation becomes

$$\mathbf{T} = (\mathbf{H}_B \otimes \mathbf{I}_b)\big(\mathbf{I}_B \otimes \mathbf{R}_1\mathbf{R}_2\big) = \mathbf{H}_B \otimes \mathbf{R}_*, \tag{8}$$

where $\mathbf{R}_* = \mathbf{R}_1\mathbf{R}_2$. Thus, although two adaptive local rotations are used during calibration, they merge exactly into a single local rotation for deployment, while preserving cross-block redistribution.

Canonical deployment form. The full $d \times d$ transformation in Eq. (8) is never explicitly materialized. We reshape the activation vector $\mathbf{X}$ into $\bar{\mathbf{X}} \in \mathbb{R}^{B\times b}$. Applying $\mathbf{T} = \mathbf{H}_B \otimes \mathbf{R}_*$ is then equivalent to

$$\mathrm{reshape}_{B\times b}(\mathbf{X}\mathbf{T}) = \mathbf{H}_B\bar{\mathbf{X}}\mathbf{R}_*, \tag{9}$$

where $\mathbf{H}_B$ mixes corresponding coordinates across blocks and $\mathbf{R}_*$ performs the shared local rotation within each block. Activations are therefore transformed online in this factorized form, avoiding explicit construction of $\mathbf{T}$. For weights, the same transformation is absorbed offline according to Eq. (1), i.e., $\mathbf{W} \mapsto \mathbf{W}\mathbf{T}$. The transformed weights are then quantized and stored, introducing no additional weight-side computation during inference.

3.2 Memory-Efficient Kernel Fusion

The separable structure of cross-block smoothing allows the dense local rotation $\mathbf{R}_*$ and the lightweight cross-block mixing to be scheduled separately. We implement $\mathbf{R}_*$ as a batched FP16 Tensor-Core GEMM, while fusing the Hadamard transform with adjacent memory-bound operations. The resulting fusion schemes and their extension to full-width Hadamard transforms are described below. Detailed kernel procedures are provided in Appendix A.

Table 1: Perplexity, quantization time, and peak memory comparison of attention-aware quantization variants after cross-block smoothing.

Method	PPL ↓	Time (s) ↓	Peak Memory (GB) ↓
GPTQ	5.93	815.43	12.38
BoA-Q	5.93	3284.76	23.21
BoA-K	5.93	3303.48	23.21
Full BoA	5.89	6260.26	23.21
LiteBoA	5.89	1494.93	12.62

Kernel fusion. Before quantized linear layers, we fuse the cross-block Hadamard transform with row-wise scale estimation, INT4 quantization, and packing, keeping transformed values on chip to avoid additional FP16 intermediates in global memory. Before `down_proj`, we commute the cross-block transform ahead of the local rotation and fuse it with SiLU and gating multiplication, keeping intermediates in registers to reduce kernel-launch overhead and global-memory traffic.

Extension to full-width Hadamard transforms. The same execution principle applies to methods such as QuaRot (Ashkboos et al., 2024b), which retain an online full-width Hadamard transform before `down_proj`. When the normalized Hadamard transform admits a Kronecker factorization $\mathbf{H}_d = \mathbf{H}_B \otimes \mathbf{H}_b$, the cross-block factor $\mathbf{H}_B$ can be fused with neighboring memory-bound operations, while $\mathbf{H}_b$ is evaluated separately.

3.3 LITEBOA

BoA (Kim et al., 2025) demonstrates that attention-aware second-order information can improve weight quantization by incorporating the attention dependencies of Query, Key, and Value projections. But this improvement comes with substantial calibration and memory overhead. We revisit BoA on models transformed by Cross-Block Smoothing from an accuracy–efficiency perspective and identify two opportunities for simplification: projection-level selectivity and head-level Hessian sharing.

Projection-Level Selectivity. BoA introduces attention-aware objectives for all QKV projections. For Query and Key, however, their perturbations affect the attention map through the nonlinear softmax operation, and the corresponding objectives rely on relaxed approximations. While these objectives are beneficial for low-bit quantization, their contribution becomes marginal after cross-block smoothing, which already makes the transformed representations substantially easier to quantize. As shown in Table 1, applying the BoA-specific Hessian to Query or Key yields almost no perplexity improvement over standard GPTQ (Frantar et al., 2022), while considerably increasing calibration time and peak memory. We therefore retain the standard GPTQ Hessian for Query and Key and preserve attention-aware curvature only for the Value projection, whose objective remains exactly quadratic once the attention map is fixed.

Head-Level Hessian Sharing. For attention head h, the attention-aware Value Hessian is

$$\mathbf{H}_{V,h} = 2(\mathbf{A}_h \mathbf{X})^\top (\mathbf{A}_h \mathbf{X}), \tag{10}$$

where $\mathbf{A}_h$ denotes the full-precision attention map. Directly following BoA requires maintaining a separate $d \times d$ Hessian for each head. LiteBoA instead aggregates these head-wise statistics into a single shared Hessian:

$$\overline{\mathbf{H}}_V = \frac{1}{N_h} \sum_{h=1}^{N_h} \mathbf{H}_{V,h}, \tag{11}$$

which is shared by the entire Value projection during standard GPTQ quantization. The head-wise statistics are accumulated sequentially, avoiding persistent storage of separate Hessians and reducing the Hessian storage complexity from $O(N_h d^2)$ to $O(d^2)$, while leaving the subsequent GPTQ rounding and error-compensation procedure unchanged.

Together, these two simplifications form LiteBoA. As shown in Table 1, LiteBoA achieves the same perplexity of 5.89 as full BoA conditioning, while reducing calibration and quantization time from 6260.26 s to 1494.93 s and peak memory from 23.21 GB to 12.62 GB.

Table 2: Comparison of perplexity on WikiText2 and average accuracy across nine zero-shot tasks.

#Bits W-A-KV	Method	LLaMA-3 8B		LLaMA-3 70B		LLaMA-2 7B		LLaMA-2 13B		LLaMA-2 70B		LLaMA 7B		LLaMA 13B		LLaMA 30B	
		0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)	0-shot Avg.(↑)	Wiki (↓)
16-16-16	FP16	68.15	6.14	73.85	2.85	65.25	5.47	67.63	4.88	71.56	3.32	64.65	5.68	66.66	5.09	69.86	4.10
4-4-4	QuaRot	62.47	8.06	67.00	6.12	61.58	6.07	64.74	5.39	69.75	3.78	61.89	6.22	64.83	5.53	68.24	4.61
	DuQuant	62.82	7.94	67.83	5.84	61.73	6.11	65.67	5.30	70.41	3.74	61.96	6.24	64.97	5.59	67.47	4.59
	RUQuant	63.28	7.90	63.57	7.36	62.35	6.09	65.73	5.30	70.50	3.74	61.87	6.22	65.13	5.51	68.37	4.59
	SpinQuant	63.83	7.36	64.44	6.38	62.01	5.94	65.69	5.24	70.10	3.70	-	-	-	-	-	-
	OSTQuant	**64.92**	7.40	OOM	OOM	62.53	5.94	66.09	5.27	OOM	OOM	62.21	6.09	64.58	5.44	68.38	4.52
	CSQuant	63.82	**7.35**	**70.87**	**4.30**	**62.88**	**5.89**	**66.17**	**5.19**	**70.60**	**3.66**	**62.27**	**6.08**	**65.18**	**5.40**	**68.54**	**4.45**
4-16-16	QuaRot	67.15	6.53	73.26	3.53	64.23	5.59	66.79	5.00	70.95	3.45	63.67	5.82	65.91	5.23	**69.95**	4.26
	DuQuant	65.63	6.82	65.62	5.89	63.81	5.70	67.19	5.03	71.19	3.47	63.35	5.92	65.82	5.24	69.41	4.25
	RUQuant	66.75	6.80	64.42	6.49	64.00	5.68	66.77	5.03	**71.30**	3.46	63.85	5.89	66.28	5.25	69.43	4.28
	SpinQuant	66.85	6.50	73.02	3.49	63.52	5.59	67.17	4.96	71.25	3.42	-	-	-	-	-	-
	OSTQuant	**67.71**	6.54	OOM	OOM	64.32	5.61	**67.80**	4.99	OOM	OOM	**63.91**	5.82	66.36	5.18	69.66	4.22
	CSQuant	67.24	**6.47**	**73.26**	**3.29**	**64.37**	**5.57**	67.27	**4.96**	70.96	**3.42**	63.67	**5.78**	**66.58**	**5.17**	69.57	**4.20**
3-16-16	QuaRot	65.05	7.61	70.95	5.25	62.27	6.16	65.15	5.36	70.64	3.71	61.86	6.22	64.65	5.44	**68.62**	4.58
	DuQuant	55.23	11.22	56.97	15.92	60.16	6.86	63.86	5.75	69.31	4.16	59.43	7.02	63.42	5.92	66.96	4.96
	RUQuant	55.37	11.09	60.49	12.71	59.87	6.78	64.69	5.79	69.03	4.17	59.39	6.97	62.68	5.95	66.88	5.02
	SpinQuant	64.10	7.52	71.26	5.18	61.85	6.07	66.20	5.36	70.49	3.73	-	-	-	-	-	-
	OSTQuant	63.65	7.52	OOM	OOM	61.94	6.11	**66.47**	5.35	OOM	OOM	**62.20**	6.18	64.71	5.47	68.43	4.55
	CSQuant	**64.26**	**7.25**	**72.21**	**4.42**	**62.27**	**5.89**	65.77	**5.21**	**70.86**	**3.71**	62.03	**6.06**	**65.00**	**5.39**	68.10	**4.50**
2-16-16	QuaRot	37.27	27.52	37.85	28.90	38.57	24.75	44.45	12.66	62.72	5.99	45.15	12.70	51.34	9.53	55.47	7.87
	DuQuant	32.01	9.19e5	31.03	3.02e4	30.76	2.18e4	31.01	9.24e3	31.06	NaN	30.99	8.24e3	31.09	7.44e3	31.32	6.62e3
	RUQuant	32.80	4.00e5	31.06	1.99e4	31.28	7.18e3	30.91	7.95e3	30.89	4.90e2	31.14	7.18e3	31.00	2.43e3	32.02	1.90e3
	SpinQuant	38.01	36.73	35.66	1.61e2	36.82	38.82	45.01	12.27	59.93	8.51	-	-	-	-	-	-
	OSTQuant	35.34	30.62	OOM	OOM	37.28	35.59	39.20	15.78	OOM	OOM	38.97	17.91	35.62	31.06	48.68	9.64
	CSQuant	**44.55**	**15.43**	**62.03**	**8.57**	**50.13**	**9.99**	**54.60**	**8.07**	**64.21**	**5.59**	**50.65**	**9.75**	**55.34**	**8.04**	**59.91**	**6.62**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Models and Datasets. We evaluate CSQuant on representative models across three generations of the LLaMA family, including LLaMA (7B-30B) (Touvron et al., 2023a), LLaMA-2 (7B-70B) (Touvron et al., 2023b), and LLaMA-3 (8B-70B) (Team, 2024). For language modeling evaluation, we report perplexity (PPL) on the WikiText2 dataset (Merity et al., 2017). Since perplexity alone does not necessarily capture the degradation of downstream capabilities under aggressive quantization, we further evaluate zero-shot accuracy on nine commonly used benchmarks using `lm-evaluation-harness` (version 0.4.4) (Gao et al., 2024): BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), LAMBADA (Radford et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-Easy, and ARC-Challenge (Boratko et al., 2018). For calibration, we randomly sample 128 sequences of 2,048 tokens from the WikiText2 dataset.

Baselines. We compare our method with several strong weight and activation quantization methods. (1) QuaRot (Ashkboos et al., 2024b), (2) DuQuant (Lin et al., 2024), (3) RUQuant (Liu et al., 2026), (4) SpinQuant (Liu et al., 2025b), (5) OSTQuant (Hu et al., 2025).

Implementation Details. Following DuQuant (Lin et al., 2024) and RUQuant (Liu et al., 2026), we apply a fixed SmoothQuant (Xiao et al., 2023) transformation before constructing the adaptive rotations. This transformation can be absorbed into the model parameters without introducing additional inference overhead. In contrast, OSTQuant (Hu et al., 2025) learns its scaling transformation through calibration-based optimization, jointly with orthogonal transformations. We use GPTQ for weight reconstruction with symmetric per-output-channel quantization without weight grouping.. All quantization experiments are conducted on a single NVIDIA A800 GPU, while perplexity and zero-shot accuracy evaluations for 70B models are performed on two NVIDIA A800 GPUs. Further implementation and evaluation details are provided in Appendix B.

4.2 MAIN RESULTS

Table 2 reports WikiText2 perplexity and average accuracy across nine zero-shot tasks. Extended results for individual tasks are provided in Appendix C.3. Under W4A4KV4 quantization, CSQuant achieves the lowest perplexity across all evaluated models while maintaining competitive zero-shot accuracy. For weight-only quantization, CSQuant maintains competitive performance at W4A16 and

Table 3: The ablation of rotation schemes and LiteBoA on LLaMA-2 7B and LLaMA-2 13B. Local Rotation applies only a block-shared adaptive rotation. RPR denotes rotation-permutation-rotation. CBS denotes our Cross-Block Smoothing. We report WikiText2 perplexity under W4A4KV4 and W2A16 quantization.

Rotation Scheme	LiteBoA	LLaMA-2 7B		LLaMA-2 13B	
		W4A4KV4	W2A16	W4A4KV4	W2A16
Local Rotation	–	7.75	14.07	5.77	9.92
Local Rotation	✓	7.68	13.77	5.73	8.83
RPR	–	5.99	12.04	5.25	8.78
RPR	✓	5.94	10.13	5.22	8.11
CBS	–	5.93	10.96	5.22	8.51
CBS	✓	**5.89**	**9.99**	**5.19**	**8.07**

Table 4: Prefill speedup under W4A4KV4 relative to FP16 on an RTX 3090. Measurements are conducted on a single Transformer block with a batch size of 4 and a sequence length of 2048.

Rotation Scheme	Prefill Speedup ↑	
	LLaMA-2 7B	LLaMA-2 13B
Local Rotation	2.25×	2.35×
RPR	1.87×	1.96×
CBS	2.25×	2.35×

Table 5: Decode speedup under W4A4KV4 relative to FP16 on an RTX 3090. Measurements are conducted on a single LLaMA-2 7B Transformer block with a context length of 4096.

Method	BS=8	BS=16	BS=32	BS=64
QuaRot	0.64×	1.08×	1.69×	2.36×
DuQuant	0.84×	1.08×	1.68×	2.29×
OSTQuant	0.68×	1.14×	1.78×	2.38×
CSQuant	0.76×	1.28×	1.98×	2.43×

achieves more pronounced improvements at lower bit-widths, particularly at W2A16. On LLaMA-3 70B, CSQuant achieves a perplexity of 8.57 and a zero-shot accuracy of 62.03% under W2A16, compared with 28.90 and 37.85% for QuaRot, and 161 and 35.66% for SpinQuant, respectively. These results indicate that CSQuant better preserves model accuracy under aggressive low-bit quantization.

4.3 Ablation Studies

Table 3 evaluates the contributions of Cross-Block Smoothing and LiteBoA under W4A4KV4 and W2A16 quantization. On LLaMA-2 7B, CBS reduces perplexity from 5.99 to 5.93 under W4A4KV4 and from 12.04 to 10.96 under W2A16 compared with RPR. LiteBoA further improves these results to 5.89 and 9.99, respectively. Similar improvements are observed on LLaMA-2 13B, demonstrating the complementary benefits of CBS and LiteBoA, particularly under aggressive 2-bit quantization.

Table 4 evaluates the prefill efficiency of different rotation schemes. On LLaMA-2 7B, RPR reduces the speedup from $2.25\times$ to $1.87\times$ due to additional online rotations and permutation. Our fused CUDA kernels effectively hide the overhead of cross-block Hadamard mixing, allowing CBS to match the measured prefill speedup of a single local rotation. The same trend holds on LLaMA-2 13B, where CBS achieves a $2.35\times$ speedup compared with $1.96\times$ for RPR.

4.4 Inference Efficiency and Memory Usage

Table 6 compares the prefill speedup and memory saving factor of different methods relative to FP16. The measurements are conducted using the e2e benchmark framework provided in the official QuaRot (Ashkboos et al., 2024b) codebase. Since the limited memory capacity of the RTX 3090 prevents full-model evaluation at long sequence lengths, we report the results on a single Transformer block in the main text. Full-model measurements at shorter sequence lengths are provided in Appendix C.1. CSQuant consistently achieves the highest prefill speedup, ranging from $2.35\times$ at sequence length 256 to $2.12\times$ at 8192, outperforming all baselines. Meanwhile, CSQuant preserves the memory efficiency of 4-bit inference, achieving memory saving factors comparable to the other

Table 6: Prefill speedup and memory-saving factor of LLaMA-2 7B under W4A4KV4 relative to FP16 across different sequence lengths. All tests are conducted on a single Transformer block with a batch size of 4 on an RTX 3090 GPU.

Method	Prefill Speedup (Seqlen)						Memory Saving Factor				
	256	512	1024	2048	4096	8192	Static	256	512	1024	2048
QuaRot	2.11×	2.20×	2.09×	2.04×	1.98×	1.95×	3.40×	2.26×	1.79×	1.43×	1.19×
DuQuant	1.98×	2.04×	1.92×	1.87×	1.81×	1.73×	3.40×	2.25×	1.78×	1.42×	1.19×
OSTQuant	2.23×	2.30×	2.18×	2.13×	2.07×	2.02×	3.40×	2.26×	1.79×	1.43×	1.19×
CSQuant	**2.35×**	**2.43×**	**2.32×**	**2.25×**	**2.18×**	**2.12×**	**3.40×**	**2.26×**	**1.79×**	**1.43×**	**1.19×**

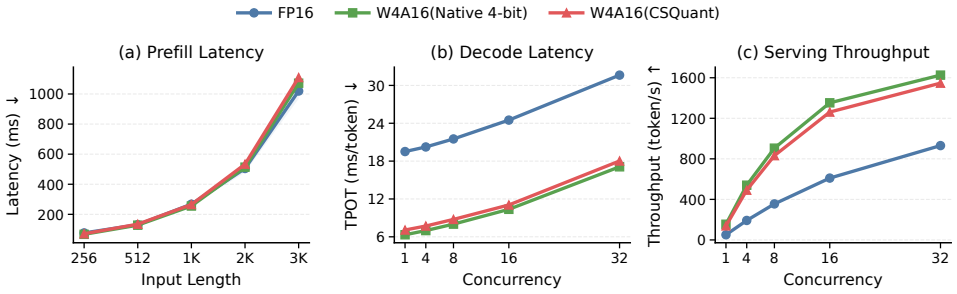

Figure 3: End-to-end inference performance of LLaMA-2 7B, comparing FP16, native W4A16, and CSQuant (W4A16). (a) Prefill-plus-first-token latency at batch size 1 across input lengths; (b) mean TPOT excluding the first token; and (c) aggregate output-token throughput across request concurrencies, with 128 input and 256 output tokens per request.

methods. These results show that CSQuant effectively reduces online rotation overhead without introducing additional memory cost.

Table 5 further evaluates decode performance under high-concurrency settings with a context length of 4096 and varying batch sizes (BS). Since full-batch prefill at BS=64 exceeds GPU memory capacity, we populate the KV cache token by token before measuring decode latency. CSQuant achieves speedups of $1.28\times$, $1.98\times$, and $2.43\times$ at batch sizes of 16, 32, and 64, respectively, outperforming all evaluated baselines. At BS=8, all quantized methods are slower than FP16, consistent with the observation in QuaRot (Ashkboos et al., 2024b) that the benefits of reduced memory traffic may not offset quantization overhead at smaller batch sizes. As batch size increases, CSQuant achieves greater decode acceleration, demonstrating its efficiency in high-concurrency inference scenarios.

Figure 3 further evaluates the end-to-end inference efficiency of CSQuant. We generate W4A16 checkpoints using `LLM Compressor` v0.12.1 and evaluate them with `vLLM` v0.23.1 (Kwon et al., 2023), using Marlin (Frantar et al., 2024), FlashAttention (Dao et al., 2022), `torch.compile`, and CUDA Graph. All experiments are conducted on a single NVIDIA RTX 3090. Under W4A16 quantization, CSQuant achieves prefill latency close to native 4-bit weight-only inference while substantially reducing decode TPOT compared with FP16. It also consistently improves serving throughput across different concurrency levels, approaching the native W4A16 baseline. Overall, CSQuant introduces limited runtime overhead while preserving most of the practical inference acceleration of 4-bit weight-only quantization.

5 Conclusion

We presented CSQuant, an efficient low-bit LLM quantization framework comprising Cross-Block Smoothing and LiteBoA. Cross-Block Smoothing combines structured Hadamard mixing with fused CUDA kernels to reduce online overhead, while LiteBoA preserves attention-aware second-order information at lower calibration cost. Experiments demonstrate robust low-bit quantization with efficient inference and calibration. CSQuant consistently maintains strong accuracy across model scales and aggressive quantization settings. Future work will explore more aggressive low-bit quantization and further optimize fused kernels for end-to-end LLM inference.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, improving manuscript readability, and providing feedback on experimental design and result interpretation. All research ideas, methodological innovations, experimental implementations, empirical observations, and scientific conclusions were independently developed, obtained, or drawn by the authors. All AI-assisted content was critically reviewed and verified by the authors, who take full responsibility for the final manuscript.

REFERENCES

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari Do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations (ICLR)*, 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. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024b.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *Conference on Artificial Intelligence (AAAI)*, pp. 7432–7439. AAAI Press, 2020.

Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, Ryan Musa, Kartik Talamadupula, and Michael Witbrock. A systematic classification of knowledge, reasoning, and context within the ARC dataset. *CoRR*, abs/1806.00358, 2018.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. Quip: 2-bit quantization of large language models with guarantees. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 2924–2936, 2019.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 16344–16359, 2022.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless LLM weight compression. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 4171–4186, 2019.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *International Conference on Machine Learning (ICML)*, volume 235, pp. 12284–12303, 2024.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022.

Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. *arXiv preprint arXiv:2408.11743*, 2024.

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, 2024.

Xing Hu, Yuan Cheng, Dawei Yang, Zhixuan Chen, Zukang Xu, Jiangyong Yu, Chen Xu, Zhihang Yuan, Zhe Jiang, and Sifan Zhou. Ostquant: Refining large language model quantization with orthogonal and scaling transformations for better distribution fitting. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Junhan Kim, Ho-Young Kim, Eulrang Cho, Chungman Lee, Joonyoung Kim, and Yongkweon Jeon. Boa: Attention-aware post-training quantization without backpropagation. In *International Conference on Machine Learning (ICML)*, volume 267, 2025.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. In *International Conference on Machine Learning (ICML)*, volume 235, pp. 23901–23923, 2024.

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 *Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100, 2022.

Yuhang Li, Ruokai Yin, Donghyun Lee, Shiting Xiao, and Priyadarshini Panda. GPTAQ: efficient finetuning-free quantization for asymmetric calibration. In *Forty-second International Conference on Machine Learning (ICML)*, 2025.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized llms. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: activation-aware weight quantization for LLM compression and acceleration. *CoRR*, abs/2306.00978, 2023.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4A8KV4 quantization and system co-design for efficient LLM serving. In *Proceedings of the Eighth Conference on Machine Learning and Systems (MLSys)*, 2025.

Han Liu, Haotian Gao, Xiaotong Zhang, Changya Li, Feng Zhang, Wei Wang, Fenglong Ma, and Hong Yu. SEPTQ: A simple and effective post-training quantization paradigm for large language models. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 812–823, 2025a.

Han Liu, Haotian Gao, Changya Li, Feng Zhang, Xiaotong Zhang, Wei Wang, and Hong Yu. Ruquant: Towards refining uniform quantization for large language models. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 843–854, 2026.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. LLM-QAT: data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics(ACL)*, 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. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025b.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, volume 119, pp. 7197–7206, 2020.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Conference on Artificial Intelligence (AAAI)*, pp. 8732–8740, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4462–4472, 2019.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

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*, 2023b.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *International Conference on Machine Learning (ICML)*, volume 235, pp. 48630–48656.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008, 2017.

Guangxuan Xiao, Ji Lin, Mickaël 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 (ICML)*, pp. 38087–38099, 2023.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. RPTQ: reorder-based post-training quantization for large language models. *CoRR*, abs/2304.01089, 2023.

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

Tianyi Zhang and Anshumali Shrivastava. Leanquant: Accurate and scalable large language model quantization with loss-error-aware grid. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *CoRR*, abs/2308.07633, 2023.

A Fused Kernel Procedures

This appendix describes the CUDA-level execution of the two fused operators introduced in Section 3.2. Let an activation contain N rows and $d = Bb$ channels. After reshaping, the channel dimension is viewed as B blocks with b local features:

$$\mathbf{X} \in \mathbb{R}^{N \times d} \quad \longleftrightarrow \quad \mathbf{X}^{\mathrm{blk}} \in \mathbb{R}^{N \times B \times b}. \tag{12}$$

For a fixed row n and local feature i, the cross-block transform operates on

$$\left[X^{\mathrm{blk}}_{n,0,i}, X^{\mathrm{blk}}_{n,1,i}, \ldots, X^{\mathrm{blk}}_{n,B-1,i}\right]^{\mathsf{T}}. \tag{13}$$

A.1 Fused Hadamard and INT4 Quantization

Algorithm 1 describes the fused operator following the local rotation. One CUDA thread block processes one activation row, and thread i processes local feature i. Each thread performs a length-B Hadamard butterfly in registers. Compared with a sequence of standalone Hadamard, reduction, quantization, and packing kernels, this implementation avoids writing and rereading the FP16 Hadamard output in global memory.

Algorithm 1 Fused cross-block Hadamard transform and INT4 quantization

Require: Reshaped Activation $\mathbf{Y} \in \mathbb{R}^{N \times B \times b}$ in FP16
Ensure: Packed INT4 activation $\mathbf{Q}$ and per-token quantization parameters $\boldsymbol{\theta}$

for all rows n in parallel **do**
  Assign one CUDA thread block to row n
  for all local features $i \in \{0, \ldots, b-1\}$ in parallel **do**
    Load $\{Y_{n,j,i}\}_{j=0}^{B-1}$ from global memory into registers
    for $\ell = 0, \ldots, \log_2 B - 1$ **do**
      Apply the stage-ℓ Hadamard butterfly in registers
    end for
    Normalize the transformed values by $\sqrt{B}$
    Store the transformed values in shared memory
    Compute thread-local quantization statistics
  end for
  Synchronize threads within the block
  Reduce the thread-local statistics across the block
  Compute per-token quantization parameters $\boldsymbol{\theta}_n$
  for all channel pairs $(2c, 2c+1)$ in parallel **do**
    Load the transformed values from shared memory
    Quantize both values to INT4 using $\boldsymbol{\theta}_n$
    Pack the two INT4 values into one byte
  end for
  Write the packed INT4 row and $\boldsymbol{\theta}_n$ to global memory
end for

A.2 Fused MLP Activation and Hadamard Transform

Algorithm 2 describes the fused MLP operator. One CUDA thread block processes one token row. Thread i handles one local feature and keeps its B cross-block values in registers throughout the elementwise activation and Hadamard butterfly. The subsequent local rotation is executed separately using an FP16 Tensor-Core GEMM.

The fused kernel does not materialize either $\mathrm{SiLU}(\mathbf{G})$ or $\mathrm{SiLU}(\mathbf{G}) \odot \mathbf{U}$ in global memory. It reads $\mathbf{G}$ and $\mathbf{U}$ once and writes only the transformed activation $\mathbf{Z}$. The subsequent local rotation is applied to $\mathbf{Z}$ using a separate FP16 Tensor-Core GEMM.

Fusion of power-of-two Hadamard factors. The same fusion principle can also be applied to online full-width Hadamard transforms used by existing rotation-based quantization methods. When

Algorithm 2 Fused SiLU, gating multiplication, and cross-block Hadamard transform

Require: $\mathbf{G}, \mathbf{U} \in \mathbb{R}^{N\times B\times b}$ in FP16
Ensure: Hadamard-transformed activation $\mathbf{Z} \in \mathbb{R}^{N\times B\times b}$ in FP16
for all rows n in parallel **do**
  for all local features $i \in \{0, \dots, b-1\}$ in parallel **do**
    for $j = 0, \dots, B-1$ **do**
      Load $g_j \leftarrow G_{n,j,i}$ and $u_j \leftarrow U_{n,j,i}$
      $a_j \leftarrow \mathrm{SiLU}(g_j)\, u_j$
    end for
    for $\ell = 0, \dots, \log_2 B - 1$ **do**
      Apply the stage-ℓ Hadamard butterfly to $\{a_j\}_{j=0}^{B-1}$
    end for
    $a_j \leftarrow a_j/\sqrt{B}$ for all j
    Write $Z_{n,j,i} \leftarrow a_j$ for all j
  end for
end for

a d-dimensional transform admits a Kronecker factorization

$$\mathbf{H}_d = \mathbf{H}_m \otimes \mathbf{H}_{2^k}, \qquad d = m2^k, \tag{14}$$

up to a fixed layout transformation, the power-of-two factor $\mathbf{H}_{2^k}$ can be fused with neighboring memory-bound operations, while the remaining factor $\mathbf{H}_m$ is evaluated separately. Thus, the proposed fusion strategy can also reduce the overhead of existing online Hadamard-based rotation pipelines.

B Additional Experimental Details

B.1 Models, Calibration Data, and Evaluation

We evaluate CSQuant on the LLaMA model family, including LLaMA-1 7B–30B, LLaMA-2 7B–70B, and LLaMA-3 8B–70B. All experiments use post-training quantization and do not update the original pretrained model parameters.

For calibration, we randomly sample 128 sequences of 2,048 tokens from the WikiText2 training set. We use a fixed random seed of 444 and a calibration batch size of one. Unless otherwise specified, the same calibration-set size, sequence length, and method-specific hyperparameters are shared across model sizes.

We report language-modeling perplexity on the WikiText2 dataset. We additionally evaluate zero-shot performance using `lm-evaluation-harness` version 0.4.4 on BoolQ, HellaSwag, LAMBADA, OpenBookQA, PIQA, SocialIQA, WinoGrande, ARC-Easy, and ARC-Challenge. We use the task-specific accuracy or normalized-accuracy metric provided by the evaluation harness and report the average over the nine tasks.

B.2 Quantization Settings

We primarily evaluate CSQuant under W4A4KV4 quantization, following the same bit-width configuration as OSTQuant, and additionally consider weight-only settings at W4A16, W3A16, and W2A16.

Weight quantization. For a weight matrix $\mathbf{W}$, we use symmetric per-output-channel quantization. Each row of the weight matrix has an independently determined quantization scale. We do not employ weight grouping; that is, the weight group size is set to -1. The weight group size should not be confused with either the GPTQ processing block size or the local rotation block size, both of which have a target value of 128. The final per-output-channel weight ranges are selected using the implementation's MMSE range estimator.

Activation quantization. Linear-layer input activations use asymmetric dynamic per-token quantization without activation grouping. Given $\mathbf{X}$, the quantization range, scale, and zero point are

computed independently for each batch element and token along the hidden dimension C. We apply an activation range-shrinkage ratio of $\gamma_a = 0.9$.

Thus, if the original dynamic range of a token is $[x_{\min}, x_{\max}]$, the range used by the activation quantizer is $[0.9x_{\min},\ 0.9x_{\max}]$.

B.3 Fixed Smoothing

Before constructing the adaptive rotations, CSQuant applies a fixed SmoothQuant transformation. For each input channel j, the smoothing coefficient is computed as

$$s_j = \frac{(\max |\mathbf{X}_{:,j}|)^{\alpha}}{(\max |\mathbf{W}_{:,j}|)^{1-\alpha}}. \tag{15}$$

The transformation transfers part of the activation quantization difficulty to the corresponding weight channels while preserving the full-precision linear operation.

We use $\alpha = 0.6$ for LLaMA-1 and LLaMA-2 and $\alpha = 0.55$ for LLaMA-3. These values are fixed across model sizes within each model family and are not learned during quantization. Because the normalization layers in the evaluated LLaMA models do not contain bias terms, we use multiplicative smoothing without an additional channel-wise shift.

B.4 Baseline Configurations

We preserve the native smoothing and rotation strategy of each baseline rather than adding the same fixed SmoothQuant (Xiao et al., 2023) preprocessing to every method. Results for QuaRot (Ashkboos et al., 2024b), DuQuant (Lin et al., 2024), RUQuant (Liu et al., 2026), and OSTQuant (Hu et al., 2025) are obtained using their official implementations. For the evaluated LLaMA-2 and LLaMA-3 models, the SpinQuant (Liu et al., 2025b) results use officially released rotated weights.

QuaRot. QuaRot uses its native randomized Hadamard rotations. We do not add fixed SmoothQuant preprocessing to QuaRot.

SpinQuant. SpinQuant uses its officially learned rotation matrices. We do not add fixed SmoothQuant preprocessing to SpinQuant.

DuQuant. The evaluated DuQuant configuration applies fixed SmoothQuant with $\alpha = 0.6$ before its native adaptive rotation–permutation–rotation pipeline.

RUQuant. RUQuant is evaluated using its official layer-adaptive local rotations and cross-block redistribution procedure, with the method-specific hyperparameters provided by its implementation.

OSTQuant. OSTQuant jointly learns its smoothing and orthogonal rotation parameters from calibration data. OSTQuant is optimized for 100 steps using its KL-top objective without early stopping. The optimization batch size is four for the 7B model and one for the 13B and 30B models. The approximate learning rates for the rotation and smoothing parameters are 1.69×10^{-2} and 1.79×10^{-3}, respectively, with corresponding momentum values of zero and 0.9. Optimization uses Riemannian Adam.

OSTQuant weights use symmetric per-output-channel GPTQ with a weight group size of -1, a GPTQ block size of 128, a damping ratio of 0.01, and activation ordering disabled. Its activations use asymmetric dynamic per-token quantization without activation grouping. Its Key and Value tensors use asymmetric dynamic group-wise quantization with a group size of 128 and no additional clipping.

Because the methods use different transformation families, we keep method-specific hyperparameters such as rotation-training steps, smoothing learning rates, and KV quantization groups at their native or recommended values. The common experimental conditions are the target bit width, calibration-set size, calibration sequence length, and evaluation datasets.

B.5 Hardware and Efficiency Measurements

All quantization experiments are conducted on a single NVIDIA A800 GPU. Perplexity and zero-shot accuracy evaluations are performed on a single A800 GPU for models smaller than 70B and on two A800 GPUs for 70B models. The inference microbenchmarks and end-to-end serving experiments are performed on a single NVIDIA RTX 3090 GPU.

Transformer-block microbenchmark. We measure the prefill performance of a single Transformer block using the end-to-end benchmark framework from the official QuaRot implementation. The batch size is 4, and the sequence lengths are $\{256, 512, 1024, 2048, 4096, 8192\}$. For each configuration, we perform 20 warm-up iterations followed by 50 timed iterations. Latency is measured using CUDA events with device synchronization, and we report the arithmetic mean over the 50 iterations. Prefill speedup is calculated relative to the mean latency of the corresponding FP16 implementation under the same model, batch size, and sequence length. For memory evaluation, we warm up the model before resetting the CUDA peak-memory statistics and report the peak allocated GPU memory. The memory-saving factor is calculated relative to the corresponding FP16 peak allocated memory.

For decoder measurements, we first populate the KV cache with the corresponding context sequence and then generate 32 one-token decoding steps. Before each timed trial, the logical cache length is reset to the context length while preserving the populated KV cache. Latency is measured using CUDA events with device synchronization over the complete 32-step decode loop. We report the mean latency per decoding step, where each step processes one token for every sequence in the batch, together with the aggregate output-token throughput. Decoder speedup is calculated relative to the FP16 latency under the same batch size, context length, and decode length. For the largest batch-size configurations, where one-shot FP16 prefill exceeds the available GPU memory, the KV cache is populated using an OOM-safe single-token procedure.

The latency measurements in Figure 1 and the speedup results in Table 6 were obtained from separate benchmark runs under the same hardware and workload configurations. Minor differences between the reported latencies and the corresponding speedup ratios may arise from run-to-run performance variability.

End-to-end inference. For the LLaMA-2 7B W4A16 experiment, we generate quantized checkpoints using LLM Compressor version 0.12.1 and evaluate them using vLLM version 0.23.1. The inference stack uses Marlin, FlashAttention, `torch.compile`, and CUDA Graphs. We report prefill-plus-first-token latency at batch size one, mean time per output token excluding the first token, and aggregate output-token throughput. The serving-throughput experiments use 128 input tokens and 256 generated tokens per request and vary the request concurrency.

Quantization cost. Quantization time includes adaptive-rotation construction or baseline-specific optimization, Hessian-statistics collection, and final weight quantization. PPL computation and zero-shot evaluation are excluded. Peak GPU memory is measured using both the framework's internal peak-memory statistics and process-level `nvidia-smi` sampling at 0.2-second intervals. We report the peak memory of the target quantization process rather than total device memory. External wall-clock measurements that also include model loading and environment-startup overhead are used only as an auxiliary consistency check.

C More results

C.1 Full Model Speedup

We evaluate the end-to-end prefill efficiency of different 4-bit inference methods on the full LLaMA-2 7B model with batch size 1. Table 7 reports the prefill speedup over FP16 and the peak allocated GPU memory saving factor. Across all evaluated sequence lengths, CSQuant consistently achieves the highest prefill speedup. The advantage becomes more pronounced as the sequence length increases, reaching $2.25\times$ speedup at sequence length 2048, while introducing negligible additional memory overhead. The 4-bit implementations reduce the static memory footprint by approximately $3.48\times$. The memory saving factor decreases slightly with longer sequences as activations and the KV cache account for a larger fraction of memory, but remains at $3.26\times$ at sequence length 2048.

Table 7: Prefill speedup and memory saving factors of different 4-bit inference methods over FP16 on the full LLaMA-2 7B model. All results are measured with batch size 1. GPU memory is reported using peak allocated memory.

Method	Prefill Speedup (Sequence Length)				Memory Saving Factor					
	256	512	1024	2048	Static	128	256	512	1024	2048
QuaRot	1.14×	1.99×	2.04×	2.08×	3.48×	3.47×	3.45×	3.42×	3.37×	3.27×
DuQuant	1.14×	1.86×	1.90×	1.94×	3.47×	3.46×	3.44×	3.41×	3.36×	3.26×
OSTQuant	1.20×	2.05×	2.10×	2.15×	3.48×	3.47×	3.45×	3.42×	3.37×	3.27×
CSQuant	**1.36**×	**2.15**×	**2.19**×	**2.25**×	**3.48**×	**3.46**×	**3.45**×	**3.42**×	**3.36**×	**3.26**×

Table 8: Quantization cost of different methods across model scales. We report the wall-clock quantization time and peak GPU memory usage for OSTQuant and CSQuant on LLaMA-2 7B and LLaMA 30B.

Model	Method	Quant. Time (s) ↓	Peak GPU Mem (GB) ↓
LLaMA-2 7B	OSTQuant	1725.80	23.75
	CSQuant	1494.93	12.62
LLaMA 30B	OSTQuant	6735.88	75.89
	CSQuant	7987.37	25.00

C.2 Quantization Cost

Table 8 compares the quantization cost of OSTQuant and CSQuant. CSQuant substantially reduces peak GPU memory, from 23.75 GB to 12.62 GB on LLaMA-2 7B and from 75.89 GB to 25.00 GB on LLaMA 30B. In terms of quantization time, CSQuant is faster on LLaMA-2 7B but slower on LLaMA 30B. Overall, CSQuant provides a substantially more memory-efficient post-training quantization pipeline, making large-scale models practical under limited GPU memory.

C.3 Zero-shot results

Table 9: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA-2 7B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA-2 7B	16-16-16	FP16	46.33	74.62	77.71	76.00	73.92	44.20	79.11	46.11	69.22	65.25	5.47
	4-4-4	QuaRot	41.04	69.70	74.31	73.22	71.07	39.60	76.77	43.09	65.43	61.58	6.07
		DuQuant	42.83	70.92	71.80	72.75	70.42	40.80	76.28	43.24	66.54	61.73	6.11
		RUQuant	42.41	71.38	73.70	73.16	70.39	42.20	77.04	44.47	66.38	62.35	6.09
		SpinQuant	42.49	69.99	73.76	73.17	72.15	41.00	76.17	44.27	65.11	62.01	5.94
		OSTQuant	43.94	71.25	74.40	72.99	70.70	40.40	77.26	45.24	66.61	62.53	5.94
		CSQuant	43.60	72.31	74.01	73.50	70.33	43.00	77.69	44.58	66.93	62.88	5.89
	4-16-16	QuaRot	44.28	72.90	76.67	74.97	73.63	43.20	78.40	44.58	69.46	64.23	5.59
		DuQuant	43.86	72.35	74.28	75.44	72.37	42.60	78.84	44.78	69.77	63.81	5.70
		RUQuant	44.11	72.43	78.29	74.88	72.27	42.80	78.29	44.78	68.19	64.00	5.68
		SpinQuant	43.17	72.39	72.84	74.83	73.30	43.40	77.97	45.55	68.19	63.52	5.59
		OSTQuant	44.28	73.19	76.54	74.77	72.35	44.40	78.02	46.57	68.75	64.32	5.61
		CSQuant	45.56	74.28	76.36	75.05	73.16	41.80	78.40	45.45	69.30	64.37	5.57
	3-16-16	QuaRot	41.81	69.82	75.14	71.36	72.33	41.60	76.28	44.68	67.40	62.27	6.16
		DuQuant	40.10	67.09	70.00	70.01	69.51	38.60	76.61	42.73	66.77	60.16	6.86
		RUQuant	40.02	67.09	70.76	70.32	67.36	37.80	76.61	43.09	65.75	59.87	6.78
		SpinQuant	40.78	68.69	74.13	71.47	71.55	41.20	77.31	43.86	67.64	61.85	6.07
		OSTQuant	40.36	68.06	76.79	71.34	70.87	40.80	77.42	43.71	68.11	61.94	6.11
		CSQuant	40.27	69.61	74.34	72.63	72.66	40.40	77.42	44.27	68.82	62.27	5.89
	2-16-16	QuaRot	24.15	36.83	61.68	33.17	18.92	27.20	55.77	35.01	54.38	38.57	24.75
		DuQuant	27.39	26.14	38.59	26.05	0	24.60	51.52	34.44	48.15	30.76	2.18e4
		RUQuant	26.02	25.97	45.50	25.84	0	24.40	49.84	34.95	49.01	31.28	7.18e3
		SpinQuant	24.15	32.95	61.62	32.60	8.34	26.60	56.86	35.77	52.49	36.82	38.82
		OSTQuant	21.33	34.18	61.50	31.26	15.72	28.60	55.98	35.52	51.46	37.28	35.59
		CSQuant	28.84	53.16	63.58	50.99	51.87	34.40	69.42	38.89	59.98	50.13	9.99

Tables 9–16 report the complete per-task zero-shot results and WikiText2 perplexity across all evaluated LLaMA-1/2/3 models and quantization settings. These detailed results show that CSQuant maintains competitive performance under W4A4KV4 and higher-bit weight-only quantization, while showing increasingly clear advantages as the weight bit-width decreases.

Table 10: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA-2 13B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA-2 13B	16-16-16	FP16	49.06	77.53	80.55	79.38	76.75	45.20	80.52	47.34	72.38	67.63	4.88
	4-4-4	QuaRot	46.25	74.24	76.97	75.54	73.86	42.80	78.73	44.73	69.53	64.74	5.39
		DuQuant	47.44	75.04	76.91	77.39	74.99	45.20	78.45	46.01	69.61	65.67	5.30
		RUQuant	45.65	75.13	78.35	77.31	75.04	43.40	79.43	45.96	71.27	65.73	5.30
		SpinQuant	47.35	75.55	78.10	77.08	74.02	44.20	78.94	46.26	69.69	65.69	5.24
		OSTQuant	47.95	74.79	79.60	77.14	74.97	44.00	79.65	47.49	69.22	66.09	5.27
		CSQuant	48.21	76.09	79.42	77.12	75.59	43.60	79.38	47.08	69.06	66.17	5.19
	4-16-16	QuaRot	48.63	77.02	79.42	78.09	76.77	45.40	79.87	45.55	70.32	66.79	5.00
		DuQuant	49.49	76.81	80.34	78.82	76.56	43.80	80.25	46.47	72.22	67.19	5.03
		RUQuant	47.44	76.18	79.36	78.56	75.86	45.00	80.09	46.21	72.22	66.77	5.03
		SpinQuant	48.38	77.02	80.43	78.76	77.18	44.40	80.52	46.42	71.43	67.17	4.96
		OSTQuant	48.89	76.56	81.90	78.55	76.38	45.60	80.74	48.46	73.16	67.80	4.99
		CSQuant	48.72	76.68	80.34	78.75	77.12	45.20	79.49	46.88	72.30	67.27	4.96
	3-16-16	QuaRot	47.10	74.92	73.43	76.36	76.32	42.60	78.40	45.65	71.59	65.15	5.36
		DuQuant	43.17	70.88	77.68	75.27	72.25	43.00	78.73	44.83	68.90	63.86	5.75
		RUQuant	47.27	72.90	77.00	75.45	72.09	44.40	79.00	44.52	69.61	64.69	5.79
		SpinQuant	48.29	77.44	77.25	75.82	76.87	43.80	78.94	47.03	70.32	66.20	5.36
		OSTQuant	47.78	75.29	80.12	76.32	74.81	45.00	78.89	46.52	73.48	66.47	5.35
		CSQuant	45.39	74.54	77.61	76.25	77.02	43.80	78.56	46.52	72.22	65.77	5.21
	2-16-16	QuaRot	26.37	46.89	62.23	41.94	34.64	31.80	63.00	37.15	56.04	44.45	12.66
		DuQuant	27.82	26.94	38.90	26.52	0	25.00	49.13	34.70	50.12	31.01	9.24e3
		RUQuant	26.62	25.97	40.34	25.70	0	25.40	49.24	35.01	49.96	30.91	7.95e3
		SpinQuant	26.02	43.48	62.54	44.11	42.34	31.00	64.09	36.80	54.70	45.01	12.27
		OSTQuant	26.28	41.29	58.07	38.33	13.80	28.40	57.62	37.05	51.93	39.20	15.78
		CSQuant	33.02	61.70	64.37	57.87	64.49	33.80	70.51	40.99	64.64	54.60	8.07

Table 11: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA 7B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA 7B	16-16-16	FP16	44.80	72.94	75.14	76.19	73.57	44.40	79.16	45.65	70.01	64.65	5.68
	4-4-4	QuaRot	40.53	69.65	73.18	72.94	70.60	40.80	77.86	44.32	67.09	61.89	6.22
		DuQuant	41.38	67.21	72.81	73.08	69.92	43.60	76.82	44.37	68.43	61.96	6.24
		RUQuant	42.58	68.14	72.63	73.01	69.88	42.40	76.22	44.27	67.72	61.87	6.22
		OSTQuant	40.78	68.56	72.97	73.64	71.82	41.60	78.13	45.09	67.32	62.21	6.09
		CSQuant	42.32	69.23	72.26	73.64	71.49	43.00	76.61	43.91	67.96	62.27	6.08
	4-16-16	QuaRot	42.49	71.04	74.98	74.82	73.16	43.00	79.00	44.37	70.17	63.67	5.82
		DuQuant	43.52	70.45	73.91	74.98	71.78	41.60	79.43	45.09	69.38	63.35	5.92
		RUQuant	43.43	72.22	72.66	75.42	71.86	43.60	79.05	46.11	70.32	63.85	5.89
		OSTQuant	43.77	71.38	76.70	74.69	73.20	41.40	78.73	45.85	69.46	63.91	5.82
		CSQuant	43.86	71.34	73.46	75.20	73.34	42.20	78.84	45.19	69.61	63.67	5.78
	3-16-16	QuaRot	40.61	69.57	71.59	71.76	71.94	40.60	78.07	44.58	68.03	61.86	6.22
		DuQuant	38.48	64.65	69.30	71.02	64.45	38.20	77.31	43.35	68.11	59.43	7.02
		RUQuant	38.31	64.98	70.89	71.11	64.66	37.60	76.06	42.94	67.96	59.39	6.97
		OSTQuant	41.21	67.59	73.70	71.93	71.14	40.80	77.86	45.85	69.69	62.20	6.18
		CSQuant	40.02	70.03	72.11	72.17	72.48	41.00	77.31	45.14	68.03	62.03	6.06
	2-16-16	QuaRot	25.85	45.37	61.74	46.14	36.60	31.20	65.34	37.82	56.27	45.15	12.70
		DuQuant	27.73	25.97	38.10	25.74	0	26.20	50.11	35.01	50.04	30.99	8.24e3
		RUQuant	26.79	26.85	38.10	26.31	0	27.00	50.27	35.21	49.72	31.14	7.18e3
		OSTQuant	23.72	37.75	62.75	39.57	14.17	27.20	58.00	35.36	52.17	38.97	17.91
		CSQuant	28.58	53.87	68.56	52.38	48.86	33.00	69.59	39.71	61.25	50.65	9.75

Table 12: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA 13B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA 13B	16-16-16	FP	47.61	74.75	77.89	79.07	76.17	44.80	80.20	46.72	72.77	66.66	5.09
	4-4-4	QuaRot	46.25	71.89	76.27	77.20	73.34	44.20	79.05	45.39	69.85	64.83	5.53
		DuQuant	46.08	71.42	76.97	76.85	74.42	44.40	79.00	44.68	70.96	64.97	5.59
		RUQuant	46.67	72.47	75.84	77.38	73.69	44.40	79.22	45.39	71.11	65.13	5.51
		OSTQuant	45.56	70.58	75.44	76.93	72.68	44.20	78.78	45.70	71.35	64.58	5.44
		CSQuant	46.16	73.19	75.84	77.29	74.79	44.60	78.94	45.60	70.17	65.18	5.40
	4-16-16	QuaRot	46.59	72.35	77.43	78.18	75.82	45.00	79.65	45.65	72.53	65.91	5.23
		DuQuant	47.27	72.56	77.34	78.39	75.37	44.20	79.16	46.06	72.06	65.82	5.24
		RUQuant	46.84	73.61	77.43	78.44	75.66	45.20	79.98	46.62	72.69	66.28	5.25
		OSTQuant	45.65	73.06	79.45	78.71	75.33	45.60	80.03	46.78	72.61	66.36	5.18
		CSQuant	47.61	74.83	77.31	78.39	76.62	45.00	80.20	46.93	72.30	66.58	5.17
	3-16-16	QuaRot	44.28	72.14	73.30	75.88	76.03	43.60	79.92	44.98	71.74	64.65	5.44
		DuQuant	43.60	71.09	73.85	75.46	71.98	41.80	78.02	43.86	71.11	63.42	5.92
		RUQuant	41.81	69.07	71.10	75.17	72.33	43.20	77.86	44.73	68.82	62.68	5.95
		OSTQuant	43.00	71.17	78.17	75.13	75.04	43.20	79.11	45.65	71.90	64.71	5.47
		CSQuant	45.56	71.59	76.82	76.21	75.86	44.60	78.62	44.98	70.80	65.00	5.39
	2-16-16	QuaRot	31.23	53.41	64.68	53.02	54.73	33.20	68.82	40.58	62.35	51.34	9.53
		DuQuant	26.45	26.81	40.43	26.32	0.02	24.60	50.38	34.49	50.28	31.09	7.44e3
		RUQuant	25.60	27.19	40.03	25.79	0	26.80	50.60	33.88	49.09	31.00	2.43e3
		OSTQuant	21.59	31.73	55.63	30.49	14.07	26.20	54.68	35.36	50.83	35.62	31.06
		CSQuant	34.39	61.49	66.18	58.78	63.50	35.60	71.82	41.04	65.27	55.34	8.04

Table 13: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA 30B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA 30B	16-16-16	FP16	52.90	78.87	82.75	82.62	77.55	48.20	82.26	47.75	75.85	69.86	4.10
	4-4-4	QuaRot	50.68	75.63	81.13	80.74	76.36	48.40	80.20	46.62	74.43	68.24	4.61
		DuQuant	50.09	74.79	80.21	80.51	76.52	46.00	79.60	47.70	71.82	67.47	4.59
		RUQuant	50.51	77.02	81.50	81.23	77.22	46.80	79.98	47.34	73.72	68.37	4.59
		OSTQuant	50.00	77.31	80.09	80.52	76.67	48.00	80.14	48.77	73.88	68.38	4.52
		CSQuant	51.19	76.56	80.73	81.12	77.72	46.80	80.69	47.44	74.59	68.54	4.45
	4-16-16	QuaRot	53.92	78.45	81.87	81.99	77.88	48.40	81.99	48.36	76.72	69.95	4.26
		DuQuant	52.99	78.75	82.29	82.10	77.64	47.40	81.39	47.34	74.82	69.41	4.25
		RUQuant	52.39	78.70	82.94	82.12	77.66	47.40	80.79	47.54	75.30	69.43	4.28
		OSTQuant	51.88	78.07	82.81	81.87	77.53	49.80	81.50	48.31	75.14	69.66	4.22
		CSQuant	53.07	78.75	81.74	82.04	78.24	48.40	81.18	47.59	75.14	69.57	4.20
	3-16-16	QuaRot	51.28	78.07	79.91	80.66	78.79	46.20	80.20	47.13	75.30	68.62	4.58
		DuQuant	49.32	74.58	78.50	79.46	75.51	45.60	79.71	46.57	73.40	66.96	4.96
		RUQuant	48.98	73.48	80.28	79.18	75.59	44.80	80.03	46.62	72.93	66.88	5.02
		OSTQuant	47.78	76.39	81.87	79.78	78.36	48.00	80.58	48.46	74.66	68.43	4.55
		CSQuant	49.83	76.43	79.33	80.07	79.12	45.40	80.47	47.24	74.98	68.10	4.50
	2-16-16	QuaRot	35.07	63.22	65.14	59.85	61.28	37.00	70.73	39.82	67.17	55.47	7.87
		DuQuant	26.37	26.47	42.87	26.50	0	25.60	50.71	34.24	49.09	31.32	6.62e3
		RUQuant	23.98	30.56	44.34	27.95	0.14	25.40	50.44	34.75	50.59	32.02	1.90e3
		OSTQuant	31.93	53.91	65.38	50.74	41.98	31.00	65.13	37.31	60.77	48.68	9.64
		CSQuant	37.54	66.04	71.74	66.27	73.86	38.80	73.50	41.15	70.24	59.91	6.62

Table 14: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA-3 8B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA-3 8B	16-16-16	FP16	53.41	77.69	81.35	79.19	75.99	45.00	80.79	47.19	72.69	68.15	6.14
	4-4-4	QuaRot	45.99	69.23	74.13	73.23	68.62	42.00	75.30	45.34	68.35	62.47	8.06
		DuQuant	45.14	73.48	73.03	74.10	69.14	42.20	76.33	43.60	68.35	62.82	7.94
		RUQuant	47.27	71.46	74.98	74.48	69.55	40.40	76.28	43.86	71.27	63.28	7.90
		SpinQuant	47.70	75.00	76.15	75.18	70.74	40.80	77.04	45.50	66.38	63.83	7.36
		OSTQuant	50.34	74.41	77.89	75.26	71.36	43.00	76.82	44.63	70.56	64.92	7.40
		CSQuant	46.33	71.76	75.41	75.32	72.39	43.40	77.64	43.60	68.51	63.82	7.35
	4-16-16	QuaRot	50.60	77.10	80.67	77.89	73.86	44.00	79.60	46.67	73.95	67.15	6.53
		DuQuant	48.72	73.99	75.69	77.25	72.81	45.20	79.33	45.50	72.22	65.63	6.82
		RUQuant	51.62	77.31	78.01	77.33	73.70	43.00	79.76	46.83	73.16	66.75	6.80
		SpinQuant	50.51	75.46	79.63	77.87	74.46	44.80	79.98	46.06	72.85	66.85	6.50
		OSTQuant	51.88	77.95	81.38	77.91	76.29	42.80	79.43	47.54	74.19	67.71	6.54
		CSQuant	52.56	77.06	80.15	77.97	75.04	43.60	79.71	45.96	73.09	67.24	6.47
	3-16-16	QuaRot	48.55	76.39	78.53	73.90	70.68	42.40	76.99	45.96	72.06	65.05	7.61
		DuQuant	39.42	61.87	51.62	68.47	58.66	38.20	72.03	41.40	65.43	55.23	11.22
		RUQuant	34.98	60.35	69.08	64.83	56.30	36.00	71.98	40.99	63.77	55.37	11.09
		SpinQuant	45.73	71.93	77.65	73.55	71.98	40.20	77.75	45.75	72.38	64.10	7.52
		OSTQuant	44.97	71.76	79.27	73.61	69.77	41.40	76.88	45.29	69.93	63.65	7.52
		CSQuant	46.76	72.05	77.40	74.05	72.95	42.40	77.15	44.78	70.80	64.26	7.25
	2-16-16	QuaRot	21.67	38.30	52.11	34.55	13.93	28.00	58.38	36.03	52.49	37.27	27.52
		DuQuant	25.94	25.08	46.51	26.45	0	29.80	51.09	32.34	50.91	32.01	9.19e5
		RUQuant	25.34	25.21	56.30	26.64	0	27.40	51.74	33.57	49.01	32.80	4.00e5
		SpinQuant	23.21	38.93	61.41	33.84	11.47	27.20	58.27	33.37	54.38	38.01	36.73
		OSTQuant	23.55	32.11	57.40	32.24	9.49	25.00	52.99	33.11	52.17	35.34	30.62
		CSQuant	25.94	40.99	62.75	46.95	41.02	29.80	59.63	37.46	56.35	44.55	15.43

Table 15: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA-2 70B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA-2 70B	16-16-16	FP16	57.25	81.06	83.70	83.80	79.62	48.80	82.70	49.18	77.90	71.56	3.32
	4-4-4	QuaRot	54.95	80.26	80.55	81.69	79.12	46.60	81.18	47.54	75.85	69.75	3.78
		DuQuant	55.89	79.80	82.97	82.78	78.48	49.20	81.34	47.29	75.93	70.41	3.74
		RUQuant	55.72	79.04	82.20	82.30	78.61	48.80	81.94	48.36	77.51	70.50	3.74
		SpinQuant	55.72	79.76	81.22	82.44	79.02	47.00	81.72	48.06	76.01	70.10	3.70
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	55.38	80.22	82.39	82.34	79.16	48.80	82.26	48.06	76.80	70.60	3.66
	4-16-16	QuaRot	57.34	80.39	81.87	83.26	79.74	48.00	82.59	48.72	76.64	70.95	3.45
		DuQuant	57.08	79.97	82.97	83.29	79.14	48.80	82.75	48.67	78.06	71.19	3.47
		RUQuant	56.48	80.43	83.73	83.29	79.41	48.20	83.19	49.03	77.98	71.30	3.46
		SpinQuant	56.74	80.30	83.88	83.05	79.51	49.20	82.54	48.31	77.74	71.25	3.42
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	56.14	79.55	83.03	83.12	80.17	47.80	83.08	48.67	77.11	70.96	3.42
	3-16-16	QuaRot	54.86	80.22	83.33	82.01	80.92	47.60	81.61	47.39	77.82	70.64	3.71
		DuQuant	55.63	76.89	80.34	81.62	78.19	45.60	81.77	47.03	76.72	69.31	4.16
		RUQuant	52.99	77.65	78.84	81.17	77.55	47.00	81.72	47.44	76.87	69.03	4.17
		SpinQuant	55.38	80.30	83.24	82.00	79.88	46.40	82.54	47.24	77.43	70.49	3.73
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	57.59	80.77	82.63	81.76	81.62	47.80	82.54	46.98	76.09	70.86	3.71
	2-16-16	QuaRot	44.97	72.01	74.98	67.22	75.72	38.80	75.24	43.09	72.45	62.72	5.99
		DuQuant	25.60	27.23	42.11	27.85	0.19	22.60	51.20	33.83	48.93	31.06	NaN
		RUQuant	25.68	26.64	40.15	26.93	0.06	24.00	52.34	34.19	48.07	30.89	4.90e2
		SpinQuant	40.27	68.52	63.98	53.46	63.21	39.00	73.50	41.61	68.82	56.93	8.51
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	44.71	74.45	75.93	71.28	78.83	39.80	76.61	44.32	71.98	64.21	5.59

Table 16: Complete comparison of perplexity on WikiText2 and averaged accuracy on zero-shot common-sense reasoning tasks for **LLaMA-3 70B**.

Model	#Bits W-A-KV	Method	ARC-c (↑)	ARC-e (↑)	BoolQ (↑)	HellaS. (↑)	LambA. (↑)	OBQA (↑)	PIQA (↑)	SIQA (↑)	WinoG. (↑)	Avg. (↑)	Wiki2 (↓)
LLaMA-3 70B	16-16-16	FP16	64.33	85.90	85.20	84.89	79.82	48.60	84.55	50.57	80.66	73.85	2.85
	4-4-4	QuaRot	52.39	77.86	78.78	78.52	73.39	43.60	79.38	45.65	73.40	67.00	6.12
		DuQuant	57.59	80.51	75.38	79.49	71.94	42.80	82.15	49.28	71.35	67.83	5.84
		RUQuant	58.87	81.86	69.42	77.48	58.68	35.80	81.99	48.46	59.59	63.57	7.36
		SpinQuant	51.62	74.83	79.17	66.74	69.47	44.00	77.37	43.96	72.77	64.44	6.38
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	57.68	81.86	83.55	82.51	79.70	44.40	82.59	47.95	77.58	70.87	4.30
	4-16-16	QuaRot	62.97	85.61	85.14	84.54	79.78	47.20	84.06	49.95	80.11	73.26	3.53
		DuQuant	62.37	83.92	70.34	77.08	63.85	37.60	83.79	49.74	61.88	65.62	5.89
		RUQuant	61.52	84.81	69.91	77.47	59.48	33.40	83.62	49.90	59.67	64.42	6.49
		SpinQuant	62.63	84.47	85.99	84.88	78.40	48.00	83.30	49.44	80.03	73.02	3.49
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	62.29	84.43	85.14	84.33	80.32	48.00	84.33	50.26	80.27	73.26	3.29
	3-16-16	QuaRot	57.17	82.11	84.65	81.85	78.21	45.40	82.37	47.39	79.40	70.95	5.25
		DuQuant	53.58	77.15	60.03	68.17	38.68	31.60	81.56	45.80	56.20	56.97	15.92
		RUQuant	56.14	80.47	66.70	77.63	43.74	32.20	82.26	47.80	57.46	60.49	12.71
		SpinQuant	57.08	82.37	85.78	81.70	78.19	46.80	82.86	47.08	79.48	71.26	5.18
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	60.07	84.64	84.01	82.99	80.09	46.00	83.08	49.13	79.87	72.21	4.42
	2-16-16	QuaRot	23.04	40.19	53.24	33.97	15.31	27.60	60.17	35.06	52.09	37.85	28.90
		DuQuant	26.37	25.63	39.60	26.10	0.02	26.80	51.20	33.78	49.80	31.03	3.02e4
		RUQuant	25.68	25.21	40.70	26.07	0	28.60	50.87	33.73	48.70	31.06	1.99e4
		SpinQuant	23.29	39.35	51.44	29.49	8.83	26.40	58.32	33.52	50.28	35.66	160.90
		OSTQuant	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
		CSQuant	42.06	68.90	78.90	67.21	73.53	37.40	75.95	43.19	71.11	62.03	8.57