MOSAICQUANT: INLIER–OUTLIER DISAGGREGATION FOR UNIFIED 4-BIT LLM QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

4-bit quantization significantly reduces the memory footprint and accelerates the inference of large language models (LLMs). However, its limited bit-width representation struggles to faithfully capture both dense common values (*inliers*) and rare large-magnitude values (*outliers*), causing substantial accuracy degradation. Existing mixed-precision methods mitigate this by retaining outliers in high precision, but at the cost of breaking the uniformity of low-bit execution, introducing precision conversion and extra data movement that undermine practical speedup. We propose **MosaicQuant**, a unified 4-bit LLM quantization paradigm built on a novel principle of *inlier–outlier disaggregation*. Rather than elevating outlier precision, MosaicQuant quantizes the full weight matrix into a dense 4-bit base component, where inliers are captured faithfully while outlier are inevitably quantized. A sparse 4-bit residual component is then introduced to compensate for these quantization errors, selectively targeting the most error-critical weight blocks where output distortion is shown to be concentrated. However, a unified representation alone is insufficient, as naïvely executing the sparse residual as a separate kernel still breaks the unified low-bit inference pipeline. To bridge this gap, we introduce **ZipperEngine**, which fuses sparse block computation into the dense 4-bit GEMM kernel via an overlapped pipeline, unifying not only the representation but also the execution into a single coherent low-bit inference pipeline. Extensive experiments on LLaMA3 and Qwen3 demonstrate that MosaicQuant preserves near-FP16 accuracy while achieving up to $1.24\times$ speedup over the W16A16 baseline.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed in complex applications, including code generation, autonomous agent workflows, and long-document analysis (Grattafiori et al., 2024; Yang et al., 2025; Team et al., 2023; Barke et al., 2023; Chen et al., 2026; Wang et al., 2026b). This remarkable progress has been driven largely by scaling model size: LLMs have grown from 7B parameters to frontier models such as DeepSeek-V4-Pro with 1.6T parameters. However, this scaling comes at a steep cost in memory footprint and computational demand, as the large-scale matrix multiplications that dominate Transformer inference become increasingly expensive, creating major bottlenecks to practical deployment.

Low-precision computation offers a practical path to alleviating these bottlenecks. Modern GPU architectures have evolved accordingly: NVIDIA's Hopper and Blackwell architectures (NVIDIA, 2024) natively support low-bit integer (INT8, INT4) and floating-point (FP8, FP4) formats, enabling substantially higher matrix-multiplication throughput than traditional 16-bit execution. Quantization, which compresses LLM weights and runtime activations into these low-bit representations, has therefore become a key technique for efficient LLM inference (Ashkboos et al., 2025; Lin et al., 2025; Wang et al., 2025; 2026a).

However, quantizing both weights and activations to low bit-widths often causes substantial accuracy degradation. Low-bit representations have limited dynamic range and resolution, making it difficult to faithfully capture both dense common values (*inliers*) and rare large-magnitude values (*outliers*) (Dettmers et al., 2022). Existing methods often address this via mixed-precision quantization (Liu et al., 2025a; Zhao et al., 2024), preserving outlier-dominated channels or components in high precision while quantizing the remaining inlier-dominated weights to low precision. Although

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.

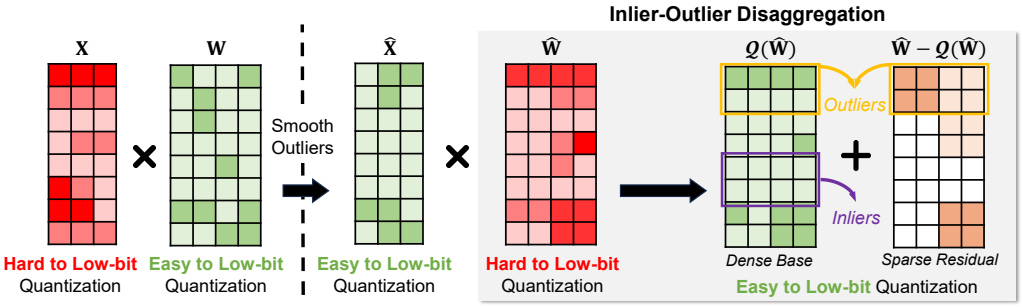


Figure 1: Overview of MosaicQuant workflow.

effective for accuracy recovery, such designs introduce high-precision fallback operations that require extra data movement and format conversion, undermining the practical hardware efficiency of low-bit inference.

Therefore, we propose **MosaicQuant**, a unified 4-bit quantization paradigm that eliminates the need for high-precision fallback through a new idea of *inlier–outlier disaggregation*. As shown in Figure 1, rather than preserving outliers in high precision, MosaicQuant quantizes the full weight matrix into a dense 4-bit base component, where inliers are represented faithfully while outliers are inevitably quantized. A sparse 4-bit residual component is then introduced to compensate for these quantization errors. Since both components operate entirely in 4-bit, this design in principle preserves the regularity and hardware efficiency of low-bit inference, but realizing this potential requires carefully addressing two challenges in how the sparse residual is allocated and executed.

For allocation, we observe that quantization errors exhibit fine-grained block-level concentration rather than spreading uniformly across channels. Compensating entire channels would therefore include many low-impact values, whereas targeting blocks allows more precise error correction with a smaller residual budget. However, naively selecting blocks by largest absolute weight values is ineffective, as the output impact of a block also depends on the input activations. We therefore derive a Hessian-based Δ-score that weights each block's residual error by activation sensitivity, enabling efficient top-K block selection without expensive per-block output evaluations.

For execution, naively computing the selected residual blocks through a separate sparse kernel does not directly translate into low latency: it requires reloading activations and merging partial outputs, adding over 30% latency and offsetting the gains of 4-bit inference. We observe, however, that residual blocks naturally align with the tiled structure of low-bit GEMM. This allows our quantized inference engine **ZipperEngine** to fuse sparse residual computation directly into the dense kernel via an overlapped pipeline, eliminating the overhead of separate kernel dispatch and realizing the hardware efficiency that unified 4-bit representation promises.

We summarize our contributions as follows:

- We propose **MosaicQuant**, a 4-bit quantization paradigm for LLMs that performs block-wise disaggregation of inlier and outlier computation, mitigating the accuracy loss of prior quantization approaches while preserving the efficiency of low-bit inference.
- We introduce **ZipperEngine** which fuses sparse block computation into the 4-bit dense base quantization and computation kernels. This avoids the overhead of computing sparse residual blocks separately and enables practical speedup even with additional residual blocks.
- We conduct extensive experiments on mainstream LLMs and system-level evaluations on the RTX 4090, demonstrating that MosaicQuant preserves near-FP16 accuracy while achieving up to $1.24\times$ speedup over the W16A16 baseline.

2 Preliminary & Related Work

Quantization is a widely used technique to accelerate the linear operators in Transformer layers by reducing their arithmetic precision. For a tensor $\mathbf{X}$, uniform symmetric quantization can be written as

$$\mathbf{Q}_\mathbf{X} = \text{round}\left(\frac{\mathbf{X}}{s_\mathbf{X}}\right), \quad s_\mathbf{X} = \frac{\max(|\mathbf{X}|)}{q_{\max}}, \tag{1}$$

where $\mathbf{Q_X}$ is the low-bit representation of $\mathbf{X}$, $s_\mathbf{X}$ is the scaling factor, and $q_{\max}$ is the maximum representable integer. The dequantized tensor is then given by $\mathcal{Q}(\mathbf{X}) = s_\mathbf{X}\,\mathbf{Q_X}$. For a linear layer with activation $\mathbf{X} \in \mathbb{R}^{N\times d_{\text{in}}}$ and $\mathbf{W} \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$, the quantized computation can be expressed as

$$\mathbf{XW} \approx \mathcal{Q}(\mathbf{X})\,\mathcal{Q}(\mathbf{W}) = s_\mathbf{X} s_\mathbf{W} \cdot \mathbf{Q_X}\mathbf{Q_W}. \tag{2}$$

To exploit low-precision compute units on modern GPUs, $\mathbf{Q_X}$ and $\mathbf{Q_W}$ typically adopt the same bit-width. Otherwise, one operand must be dequantized at runtime to match the other, which undermines the performance advantage of quantized execution.

Outlier Mitigation via Equivalent Transformations. The input activations are often rich in outliers. To mitigate their impact on quantization, a popular way is to apply equivalent transformations that reshape the activation and weight distributions which can be generally written as

$$\hat{\mathbf{X}}\hat{\mathbf{W}} = (\mathbf{X}\mathcal{T})\left(\mathcal{T}^{-1}\mathbf{W}\right), \tag{3}$$

where $\mathcal{T}$ denotes an invertible matrix. Existing methods instantiate $\mathcal{T}$ in different forms: SmoothQuant (Xiao et al., 2023) uses a diagonal channel-wise scaling to migrate activation outliers into weights, while QuaRot (Ashkboos et al., 2025) applies a fixed orthogonal rotation to redistribute outlier across channels. Furthermore, SpinQuant (Liu et al., 2025b) and FlatQuant (Sun et al., 2025) learn rotation matrix and affine transformation to flatten weight and activation distributions, respectively. These methods improve quantization by reshaping tensor distributions, but under aggressive 4-bit quantization, the remaining or migrated outliers can still induce large quantization errors in a small fraction of weight and activation blocks.

Mixed-Precision Quantization. Recent studies observe that outliers in LLM weights are often concentrated in a small subset of channels or structured components (Zhao et al., 2024; Lin et al., 2024). A natural strategy is therefore to preserve these outlier-dominated parts in high-precision, while quantizing the remaining weights to low-precision (Li et al., 2025; Saxena et al., 2025; Huang et al., 2025). For example, Atom (Zhao et al., 2024) identifies outlier activation channels and keeps them in 8-bit, while applying 4-bit quantization to the normal channels. Moreover, SVDQuant (Li et al., 2025) introduces a 16-bit low-rank branch to absorb outlier components, and quantizes the remaining residual branch to 4-bit. Although these methods improve accuracy, they introduce additional high-precision fallback operations which require extra data movement and format conversion, limiting the practical hardware efficiency of low-bit inference. In contrast, MosaicQuant uses a unified sparse low-precision component to compensate outlier-induced quantization errors at the block level, preserving accuracy while maintaining a fully low-bit execution pattern.

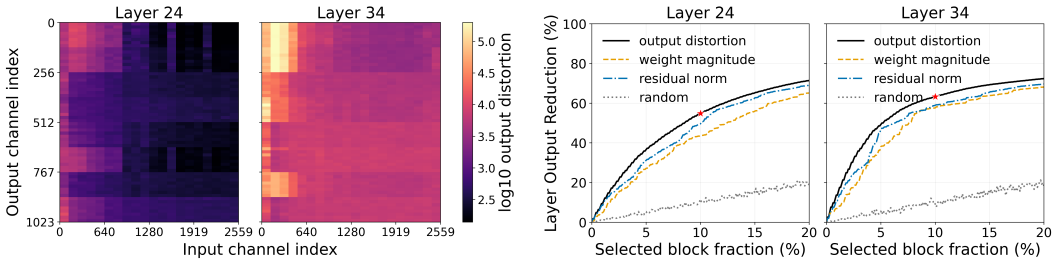
3 MOTIVATION

In this paper, we first fully migrate the outliers from the activation $\hat{\mathbf{X}}$ to the weight $\hat{\mathbf{W}}$ using Equation 3. Our key insight is to represent the migrated weight with a dense 4-bit base component, while introducing an additional sparse 4-bit residual component to compensate for the quantization errors induced by outliers. Formally, we write:

$$\hat{\mathbf{X}}\hat{\mathbf{W}} = \hat{\mathbf{X}}(\mathcal{Q}(\hat{\mathbf{W}}) + \hat{\mathbf{W}} - \mathcal{Q}(\hat{\mathbf{W}})) = \hat{\mathbf{X}}\mathcal{Q}(\hat{\mathbf{W}}) + \hat{\mathbf{X}}\mathbf{R} \approx \underbrace{\mathcal{Q}(\hat{\mathbf{X}})\mathcal{Q}(\hat{\mathbf{W}})}_{\text{dense 4-bit component}} + \underbrace{\mathcal{Q}(\hat{\mathbf{X}})\mathcal{Q}(\mathbf{R})}_{\text{sparse 4-bit component}}, \tag{4}$$

where $\mathbf{R} = \hat{\mathbf{W}} - \mathcal{Q}(\hat{\mathbf{W}})$. This decomposition suggests a residual-compensation path for reducing the error of the dense 4-bit base component. However, applying the residual component densely to all weights would introduce substantial extra computation and storage, weakening the efficiency advantage of low-bit inference. This motivates a sparse residual-compensation strategy, where only a small fraction of residual blocks are selected to recover most of the error-reduction effect.

Observation 1: Block-wise residual compensation is more effective than channel-wise. Prior studies have shown that outliers are often concentrated in a small number of channels (Zhao et al., 2024; Lin et al., 2024); thus, a naive strategy is to apply residual compensation to the corresponding weight channels. However, we observe that the output distortion caused by residual quantization is not channel-uniform, but instead exhibits fine-grained block-level concentration. Figure 2a visualizes



(a) Block-wise concentration of residual-induced output distortion.

(b) Comparison of block selection criteria.

Figure 2: Block-wise residual compensation analysis on the v_{proj} linear layers of Qwen3-4B, including Layer 24 and Layer 34.

the output distortion, measured by $\|\hat{\mathbf{X}}(\mathbf{R} - \mathcal{Q}(\mathbf{R}))\|_F^2$. If the distortion were mainly channel-wise, high-error regions would appear as full vertical stripes in the heatmap; instead, we observe localized rectangular hotspots. This indicates that compensating full channels would include many low-impact values, whereas block-wise compensation can target high-distortion regions more precisely. Therefore, residual compensation should be performed at the block-level rather than the channel-level.

Observation 2: Output-aware selection is crucial for residual compensation. After establishing block-wise compensation as the basic unit, we next ask how to select the blocks to compensate. Since residual compensation aims to preserve the layer output, the most direct criterion is the reduction in full output distortion after compensating a selected block set. For a selected block set $\mathcal{S}$, we measure this reduction as

$$C(\mathcal{S}) = \frac{\|\hat{\mathbf{X}}\mathbf{R}\|_F^2 - \left\|\hat{\mathbf{X}}\left(\mathbf{R} - \mathbf{M}_{\mathcal{S}} \odot \mathcal{Q}(\mathbf{R})\right)\right\|_F^2}{\|\hat{\mathbf{X}}\mathbf{R}\|_F^2}, \tag{5}$$

where $\mathbf{M}_{\mathcal{S}}$ is a block-level binary mask that keeps $\mathcal{Q}(\mathbf{R})$ only on the selected blocks. Figure 2b compares different block selection criteria under the same block budget, including output distortion selection, largest weight magnitude $\max_{(i,j)\in B} |\hat{W}_{ij}|$, residual norm $\|\mathbf{R}_B\|_F^2$, and random selection. The output-distortion criterion is markedly more effective than the other selection criteria. This advantage comes from its activation-aware nature: magnitude- and norm-based selection only measure the size of weights or residuals, ignoring their actual contribution to the layer output. However, directly using output distortion for block selection is expensive, because it greedily chooses the block with the largest marginal increase in $C(\mathcal{S})$ at each step and therefore requires repeated activation-dependent full-output-distortion evaluations over all candidate blocks. This motivates an efficient output-aware approximation for residual block selection.

Observation 3: A 10% sparse residual budget recovers most quantization errors. The curves in Figure 2b also show that the benefit of residual compensation is highly concentrated in a small fraction of blocks. At only a 10% block budget, output-aware selection already attains 55% layer output reduction on Layer 24 and 63% on Layer 34. In other words, compensating just 10% blocks is sufficient to recover more than half of the output distortion. This reveals a clear diminishing-return pattern: the most important error reduction comes from a small subset of highly critical blocks, while compensating the remaining blocks provides much smaller gains. Therefore, only a small sparse residual budget is needed to preserve most of the accuracy benefits of residual compensation, substantially reducing the additional memory and computation overhead.

Thus, our objective is to identify a sparse set of residual blocks whose 4-bit quantized residuals can most effectively compensate for outlier-induced quantization error. Formally, we aim to find a block-wise sparse mask $\mathbf{M}$ that minimizes:

$$E(\hat{\mathbf{X}}, \hat{\mathbf{W}}, \mathbf{M}_{\mathcal{S}}) = \left\|\hat{\mathbf{X}}\hat{\mathbf{W}} - \mathcal{Q}(\hat{\mathbf{X}})\left[\mathcal{Q}(\hat{\mathbf{W}}) + \mathbf{M}_{\mathcal{S}} \odot \mathcal{Q}(\mathbf{R})\right]\right\|_F^2, \tag{6}$$

where $\mathbf{R} = \hat{\mathbf{W}} - \mathcal{Q}(\hat{\mathbf{W}})$, and $\mathbf{M}$ is a block-wise binary mask indicating which residual blocks are selected for sparse 4-bit compensation.

Algorithm 1 Hessian-Guided Residual Block Selection

Input: migrated weight $\hat{\mathbf{W}}$, calibration activation $\hat{\mathbf{X}}$, block partition $\mathcal{B}$, block budget K
Output: selected block set $\mathcal{S}$, block mask $\mathbf{M}_{\mathcal{S}}$, quantized weight $\mathbf{W}_{\mathrm{MQ}}$

$\mathbf{R} \leftarrow \hat{\mathbf{W}} - \mathcal{Q}(\hat{\mathbf{W}})$ // residual after dense 4-bit quantization
$h_i \leftarrow \frac{2}{N}\sum_{n=1}^{N} \hat{\mathbf{X}}_{n,i}^2$ // activation sensitivity of input channel i
for each block $B \in \mathcal{B}$ **do**
  $S_B \leftarrow \left[\sum_{(i,j)\in B} h_i \mathbf{R}_{ij}^2 - \sum_{(i,j)\in B} h_i \left(\mathbf{R}_{ij} - \mathcal{Q}(\mathbf{R})_{ij}\right)^2\right]_+$ // Δ-Hessian score
end for
$\mathcal{S} \leftarrow \mathrm{TopK}(\{S_B\}_{B\in\mathcal{B}}, K)$ // select residual blocks
$(\mathbf{M}_{\mathcal{S}})_B \leftarrow 1$ if $B \in \mathcal{S}$, otherwise 0 // build block mask
$\mathbf{W}_{\mathrm{MQ}} \leftarrow \mathcal{Q}(\hat{\mathbf{W}}) + \mathbf{M}_{\mathcal{S}} \odot \mathcal{Q}(\mathbf{R})$ // dense 4-bit base + sparse 4-bit residual

4 METHOD

In this section, we present **MosaicQuant**, which preserves a unified 4-bit inference pipeline by decomposing the migrated weight into a dense 4-bit base and a sparse 4-bit residual, with output-critical residual blocks selected by an activation-aware Δ-Hessian score. We further introduce **ZipperEngine**, a dense–sparse 4-bit execution engine that integrates these stages into an overlapped pipeline, reducing redundant memory traffic and kernel-launch overhead while improving the accuracy–efficiency trade-off.

4.1 MOSAICQUANT: BLOCK-WISE INLIER–OUTLIER DISAGGREGATION

Building on the observations in section 3, MosaicQuant realizes inlier–outlier disaggregation at the block-level. It represents the migrated weight with a dense 4-bit base for common inlier and a sparse 4-bit residual component for compensating outlier. The key problem is to identify the few residual blocks that most affect the layer output, for which we derive an activation-aware Δ-Hessian score that reduces block allocation to top-K ranking and is optimal under a diagonal-Hessian objective.

Hessian-Guided Block Selection. As discussed in section 3, exact output-distortion selection provides a direct objective for residual block allocation, but it is too expensive to use as a practical algorithm. It requires recomputing the full output distortion for each candidate block, while the effects of different blocks interact in the layer error and cannot be cheaply ranked independently. To retain the output-aware principle of exact selection without its computational cost, we seek a proxy metric that approximates the marginal benefit of residual compensation. Such a proxy should be both output-aware and block-decomposable: it should capture activation-amplified residual errors and assign each block an independent score for direct top-K selection.

Inspired by GPTQ (Frantar et al., 2023), which uses the activation Hessian to estimate the output impact of weight quantization errors, we introduce a Δ-Hessian score for residual block selection. While GPTQ uses Hessian information to reduce the error of quantizing all weights, we use it to rank which blocks deserve the limited residual 4-bit compensation. Specifically, for each input channel i, we use the diagonal Hessian estimate $h_i = \frac{2}{N}\sum_{n=1}^{N} \hat{X}_{n,i}^2$ to measure how strongly errors on this channel affect the layer output. For a block B, the Δ-Hessian score is defined as

$$S_B = \left[\sum_{(i,j)\in B} h_i \mathbf{R}_{ij}^2 - \sum_{(i,j)\in B} h_i (\mathbf{R}_{ij} - \mathcal{Q}(\mathbf{R})_{ij})^2\right]_+ .$$

The score estimates the output error reduction obtained by applying residual 4-bit compensation to block B, using h_i to weight how sensitive the layer output is to errors on each input channel. A block receives a high score only when its residual error is large and occurs on activation-sensitive channels. Thus, S_B favors blocks whose compensation is expected to produce the largest reduction in layer-output distortion, rather than blocks with merely large residual magnitude.

MosaicQuant Block Allocation. With the Δ-Hessian score, residual block allocation becomes a simple score-and-sort procedure. As shown in Algorithm 1, MosaicQuant computes S_B for every

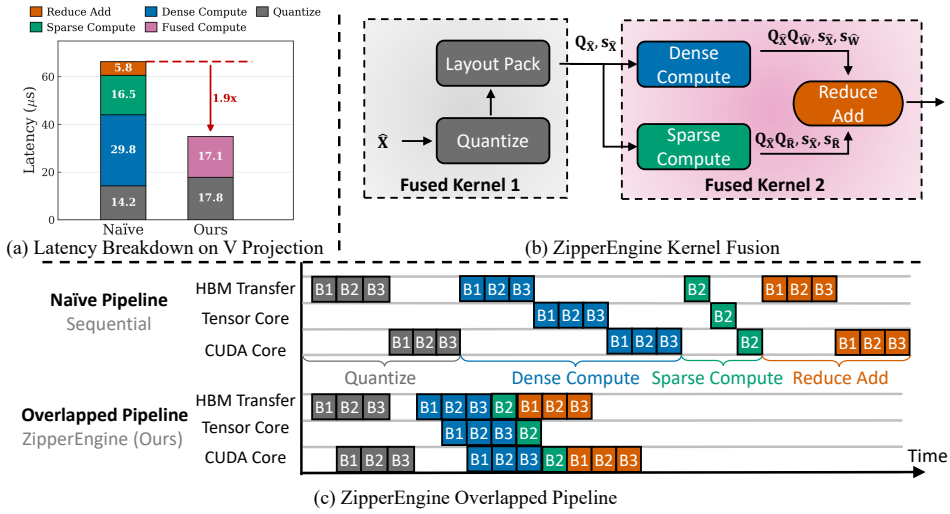


Figure 3: ZipperEngine kernel fusion and overlapped execution on the V projection layer of Qwen3-8B. By fusing dense/sparse 4-bit computation, and reduce add into an overlapped pipeline, ZipperEngine hide memory and auxiliary overheads and reduce latency by 1.9× over the naive pipeline.

block, selects the top-K blocks under the sparse residual budget. Thus, only the selected blocks receive residual 4-bit compensation, while all other blocks remain in dense 4-bit. This replaces expensive exact output distortion selection with one calibration-based scoring and sorting step.

Optimality under the Diagonal Hessian. Exact output-distortion selection is difficult because the full activation Hessian contains off-diagonal terms. These terms make the residual errors of different blocks interact with each other, so the benefit of selecting one block depends on which blocks have already been selected. Therefore, exact selection cannot be solved by ranking blocks independently.

Δ-Hessian simplifies this problem by using only the diagonal Hessian. This removes the interaction terms between different blocks, while still keeping the activation sensitivity h_i. As a result, the output-distortion objective becomes a sum of independent block-wise errors. Under this objective, each block has its own error-reduction gain, which is exactly S_B.

Theorem 4.1 (Optimality of Δ-Hessian ranking)**.** *Under the diagonal-Hessian objective, the optimal solution to*

$$\min_{\mathcal{S}:|\mathcal{S}|\le K} \sum_{i,j} h_i \left(\mathbf{R}_{ij} - (\mathbf{M}_{\mathcal{S}} \odot \mathcal{Q}(\mathbf{R}))_{ij}\right)^2$$

is obtained by selecting the top-K blocks with the largest Δ-Hessian scores S_B.

The proof is provided in Appendix A. This theorem shows that Δ-Hessian is an efficient approximation to exact output-distortion selection: after removing cross-block interaction terms, it exactly optimizes the diagonal-Hessian objective. Thus, MosaicQuant preserves the output-aware principle of exact selection while reducing residual block allocation to one sorting step.

4.2 ZipperEngine: Fusing Dense and Sparse 4-bit Kernels

ZipperEngine is designed to make MosaicQuant's sparse 4-bit residual compensation practical for low-latency inference. Its key goal is to remove the extra memory traffic and kernel-launch overhead introduced by the sparse kernels, and further overlap quantization, dense computation, sparse compensation, and reduction within a unified execution pipeline.

Kernel Fusion. We first observe that executing the sparse residual branch as a separate kernel introduces noticeable overhead. As shown in Figure 3(a), on the V projection layer of Qwen3 8B, sparse compute takes 16.5 μs, about 55% of the dense 4-bit compute latency. Together with reduce-add, the sparse residual path contributes 22.3 μs, nearly one-third of the naive pipeline latency. This overhead mainly comes from extra memory traffic and kernel launches: the sparse kernel reloads the quantized activation from HBM, writes a partial output, and then relies on another reduce-add kernel

Table 1: Comparison of the perplexity score on WikiText2 and averaged accuracy on six zero-shot commonsense reasoning tasks.

Precision	Method	LLaMA3.2 3B		LLaMA3 8B		Qwen3 4B		Qwen3 8B		Qwen3 14B		Qwen3 32B	
		0-shot Avg.	Wiki (↓)	0-shot Avg.	Wiki (↓)	0-shot Avg.	Wiki (↓)	0-shot Avg.	Wiki (↓)	0-shot Avg.	Wiki (↓)	0-shot Avg.	Wiki (↓)
W16A16	–	66.2	10.7	73.5	8.6	69.4	10.0	71.6	9.7	74.2	8.6	75.2	7.6
W4A16	RTN	60.9	18.8	71.2	10.5	67.1	10.6	69.0	12.0	70.3	9.9	68.1	38.5
	GPTQ	61.7	15.2	72.4	9.0	68.6	10.4	70.4	10.8	72.7	9.2	73.5	8.3
	AWQ	63.1	12.7	72.6	10.3	68.4	10.4	70.8	10.2	69.8	9.6	73.8	8.2
W4A8	RTN	60.7	29.0	71.0	10.6	58.8	30.6	64.4	12.3	68.3	10.9	67.4	11.2
	SmoothQuant	59.8	288.5	62.6	13.3	59.9	22.6	63.8	12.5	68.1	11.0	67.1	11.6
	QuaRot	64.2	12.4	71.6	10.6	67.8	11.2	67.1	11.5	70.7	10.4	71.0	10.5
	SpinQuant	65.2	11.5	72.3	10.3	68.4	10.8	70.4	10.8	72.5	9.8	73.1	8.9
	ResQ	64.8	11.8	72.2	10.4	68.4	10.7	70.3	10.7	72.4	12.1	73.0	9.2
	Atom	64.5	11.9	71.9	10.6	67.8	11.0	70.3	10.9	72.1	12.4	73.0	9.3
	MosaicQuant (INT4)	65.3	11.3	72.7	10.2	67.00	12.27	69.91	10.08	73.56	8.79	74.46	7.84
	MosaicQuant (FP4)	–	–	–	–	67.28	12.10	70.76	9.94	73.63	8.78	74.84	7.70
W4A4	RTN	43.2	741.9	40.4	92.9	41.5	8791	40.1	4392	42.1	18749	45.4	1796
	SmoothQuant	44.7	372.3	52.8	19.4	40.1	9910	40.1	3360.1	41.3	21675	44.4	1806
	QuaRot	61.4	16.9	68.0	13.1	61.7	12.5	66.3	15.6	67.2	18.2	67.4	15.6
	SpinQuant	64.1	11.9	70.0	11.4	65.6	11.5	68.3	11.4	71.7	12.0	72.7	10.3
	ResQ	63.0	12.2	69.9	11.6	65.1	12.0	66.8	14.2	71.0	12.8	72.0	10.9
	Atom	62.6	12.5	68.8	12.5	64.8	12.3	66.1	14.5	70.5	14.5	72.2	10.5
	MosaicQuant (INT4)	64.9	11.8	71.2	10.8	62.61	13.22	67.38	10.61	71.86	9.29	72.27	8.51
	MosaicQuant (FP4)	–	–	–	–	66.91	12.25	70.54	10.09	73.50	9.00	74.61	8.02

to merge it with the dense result. To remove this overhead, ZipperEngine fuses the dense and sparse kernels by exploiting their shared dataflow. The sparse branch reuses the same quantized activation tiles and accumulates into the same output tiles as the dense GEMM, as shown in Figure 3(b). ZipperEngine therefore fuses layout packing with quantization, and further integrates sparse residual computation and reduce-add into the dense GEMM kernel. This removes redundant HBM accesses and kernel launches, making sparse residual compensation nearly cost-free.

Overlapped pipeline. Kernel fusion removes redundant memory traffic, but the fused stages can still execute sequentially. As shown in Figure 3(c), quantization, dense compute, sparse compute, and reduce-add are performed one after another, leaving HBM, Tensor Cores, and CUDA Cores idle at different times. To improve utilization, ZipperEngine adopts a tiled overlapped pipeline. It splits each tile into independent sub-blocks and schedules different stages concurrently. For example, while Tensor Cores execute dense MMA for sub-block B1, CUDA Cores can prepare quantization and layout packing for B2, and HBM can prefetch sparse residual blocks for B3. Since each sub-block reads its own input tiles and accumulates into its own output tile, these stages do not introduce cross-tile dependencies. Reduce-add is also folded into the local fused output path instead of launched as a separate kernel. This software pipeline hides auxiliary work behind dense 4-bit GEMM, keeping hardware units busy while preserving the throughput of regular 4-bit inference.

5 Experiments

5.1 Setups

Models and metric. We benchmark against two mainstream LLMs: the LLaMA3 (Grattafiori et al., 2024) models (3B/8B) and Qwen3 (Yang et al., 2025) models (4B/8B/14B/32B). Following previous works (Sun et al., 2025; Liu et al., 2025b), we randomly sample the 128 prompts from WikiText-2 dataset (Merity et al., 2017) for calibration, each sentence with 2048 tokens. To evaluate the commonsense reasoning capability of our method, we use six zero-shot evaluation tasks, including ARC-Challenge, ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), PIQA(Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021). Additionally, we also report the perplexity score on WikiText2 (Merity et al., 2017) datasets. All accuracy metrics are tested on lm-eval (Gao et al., 2024). We also evaluate the memory save and speedup.

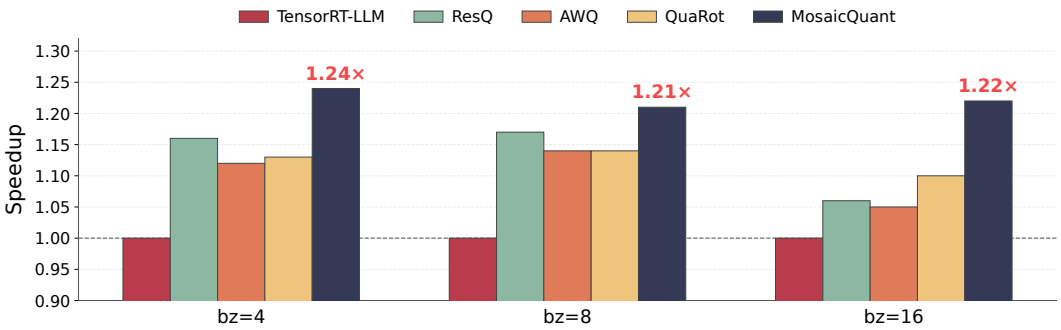


Figure 4: Throughput speedup comparison on Qwen3-8B with input length 2048 and output length 128. MosaicQuant consistently achieves the highest throughput across batch sizes 4 to 16.

Baselines. We compare MosaicQuant against six popular PTQ methods, including the naive quantization method RTN (Yao et al., 2023), the weight-only quantization GPTQ (Frantar et al., 2023) and AWQ (Lin et al., 2024), the mixed-precision quantizaiton ResQ (Saxena et al., 2025) and Atom (Zhao et al., 2024), the weight-activation quantization SmoothQuant (Xiao et al., 2023) and two recent state-of-the-art methods QuaRot (Ashkboos et al., 2025), SpinQuant (Liu et al., 2025b). See Appendix B.1 for more details.

Implementation details. Please refer to Appendix C.1 fore more details.

5.2 Accuracy Results

Table 1 reports WikiText2 perplexity and average accuracy on six zero-shot tasks. MosaicQuant (INT4) is evaluated on all six LLaMA3 and Qwen3 models, and FP4 on the four Qwen3 models. Per-task results are provided in Appendix C.2.

Language generation tasks. Under both W4A8 and W4A4, INT4 achieves lower perplexity than all competing baselines in the corresponding precision group on five of six models. On Qwen3-32B under W4A8, INT4 and FP4 achieve perplexities of 7.84 and 7.70, respectively, versus 9.2 for ResQ and 9.3 for Atom. Under W4A4, MosaicQuant maintains Qwen3 perplexities of 8.51–13.22 with INT4 and 8.02–12.25 with FP4, whereas RTN and SmoothQuant reach 10^3–10^4. Qwen3-4B remains the exception: neither variant surpasses SpinQuant in perplexity under either setting.

Zero-shot tasks. Under W4A8, INT4 leads the competing baselines on both LLaMA models and on Qwen3-14B and Qwen3-32B, but trails SpinQuant on Qwen3-4B and Qwen3-8B. FP4 leads on Qwen3-8B, Qwen3-14B, and Qwen3-32B, but remains below SpinQuant on Qwen3-4B.

Under W4A4, INT4 exceeds SpinQuant on both LLaMA models and Qwen3-14B, but falls below it on the other three models. FP4 achieves the highest average accuracy among the evaluated W4A4 methods on all four Qwen3 models, exceeding SpinQuant by 1.31–2.24 percentage points. The accuracy gaps to W16A16 are 1.30–6.79 percentage points for INT4 across all six models and 0.59–2.49 for FP4 across the four Qwen3 models.

5.3 End-to-end Throughput

Figure 4 compares MosaicQuant with TensorRT-LLM and recent quantization baselines on Qwen3-8B, using an input length of 2048 and an output length of 128. MosaicQuant consistently achieves the highest throughput at all evaluated batch sizes (4, 8, and 16), reaching 1.21–1.24× the throughput of TensorRT-LLM. Compared with ResQ, MosaicQuant achieves 1.03–1.15× the throughput. ResQ is the strongest competing baseline at batch sizes 4 and 8, whereas QuaRot is the strongest at batch size 16. Relative to the strongest competing baseline at each batch size, MosaicQuant improves throughput by 3.4–10.9%.

MosaicQuant also achieves 1.06–1.16× and 1.06–1.11× the throughput of AWQ and QuaRot, respectively. MosaicQuant uses unified dense–sparse INT4 execution, in which the dense 4-bit base and sparse 4-bit residual compensation are computed in a coordinated pipeline. By compensating only output-critical residual blocks and keeping both branches in INT4, this design aims to limit the additional memory traffic and scheduling overhead associated with residual compensation, while

Precision	Method	Qwen3 4B		LLaMA3 8B	
		0-shot Avg.	Wiki (↓)	0-shot Avg.	Wiki (↓)
W16A16	–	69.4	10.0	73.5	8.6
W4A4	w/o Sparse Branch	52.4	108.6	55.7	143.7
	w/ Channel Compensation	60.6	14.5	69.8	13.1
	MosaicQuant(INT4)	62.6	13.2	71.2	10.8

Table 2: Ablation study of MosaicQuant on Qwen3 4B and LLaMA3 8B, reporting the average accuracy on six zero-shot tasks and WikiText-2 perplexity.

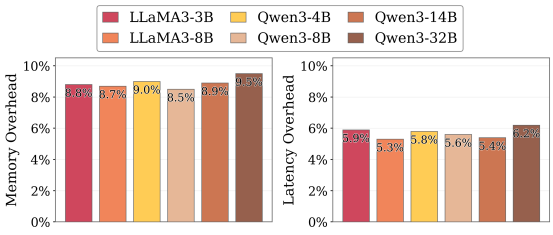


Figure 5: Memory and latency overhead of sparse 4-bit branches across six models under 10% block budget.

avoiding high-precision fallback in the compensation path. Additional kernel profiling results are reported in Appendix C.3.

5.4 Ablation Study

As illustrated in Table 2, we ablate MosaicQuant on Qwen3 4B and LLaMA3 8B under W4A4 quantization. Removing the sparse residual branch causes severe degradation, reducing zero-shot accuracy to 52.4 on Qwen3 4B and 55.7 on LLaMA3 8B, while increasing WikiText-2 perplexity to 108.6 and 143.7, respectively. Adding a 10% channel-level residual budget substantially improves accuracy to 60.6 and 69.8, and reduces perplexity to 14.5 and 13.1, confirming the importance of residual compensation. Under the same INT4 setting, replacing coarse channel-level compensation with our Hessian-guided block selection further improves accuracy to 62.6 and 71.2, and lowers perplexity to 13.2 and 10.8. These results demonstrate that fine-grained, output-aware block selection allocates the residual budget more effectively than channel-level compensation.

5.5 Overhead of Sparse 4-bit Branches

As shown in Figure 5, sparse 4-bit residual branches incur only modest overhead across all six models. The memory overhead stays within 8.5–9.5%, since only a small fraction of selected blocks receive an additional 4-bit residual branch instead of dense or high-precision compensation. Meanwhile, the latency overhead remains lower, ranging from 5.3–6.2%, indicating that sparse compensation does not translate into proportional runtime cost. This is enabled by our fused overlapped pipeline, which co-schedules the dense 4-bit base path and sparse 4-bit residual path to reduce extra memory traffic and kernel-launch overhead.

6 Limitations and Future Work

Our system evaluation currently focuses on INT4 execution on the RTX 4090, while FP4 results are limited to accuracy. Extending performance evaluation of both variants to data-center GPUs such as the H200 would help assess performance portability. Future work also includes MXFP4-compatible packing and block scaling, alongside ZipperEngine kernel adaptation, to exploit native FP4 computation on supported accelerators. Finally, broader validation on mixture-of-experts and multimodal models remains an important direction.

7 Conclusion

In this work, we introduce **MosaicQuant**, a 4-bit post-training quantization paradigm for LLMs. It performs block-level inlier–outlier disaggregation, representing weights with a dense 4-bit base and a sparse 4-bit residual selected by a Hessian-guided output-aware criterion to compensate outlier-containing blocks that dominate layer-output error. **ZipperEngine** further fuses the sparse residual into dense 4-bit GEMM and pipelines quantization, compute, and reduce-add into one overlapped kernel, eliminating extra memory traffic from sparse compensation. Experiments show that MosaicQuant preserves near-FP16 accuracy, while ZipperEngine achieves up to $1.24\times$ speedup over the 16-bit baseline on an NVIDIA RTX 4090 GPU, enabling efficient LLM serving without sacrificing the regularity of uniform 4-bit execution.

AI USE STATEMENT

We used generative AI tools for language refinement, manuscript consistency checks, and coding assistance in experimental workflows, including preparing and debugging experimental scripts. All AI-assisted text and code were reviewed and verified by the authors, who take full responsibility for the research methodology, experimental results, and conclusions presented in this work.

ETHICS STATEMENT

This work studies post-training quantization of pretrained large language models, with experiments on LLaMA3 and Qwen3 using WikiText-2 and six zero-shot evaluation benchmarks. Our evaluations focus on predictive accuracy and inference efficiency; they do not establish the safety or fairness of the quantized models. Quantization can change model outputs, and preserving average benchmark performance does not guarantee preservation of safety-related behavior. Deployment should therefore be subject to application-specific safety evaluation and the applicable usage restrictions of the underlying models and datasets.

REPRODUCIBILITY STATEMENT

Sections 4 and 5 describe MosaicQuant, the ZipperEngine inference engine, and the experimental setup. Appendix A provides the proof of the residual-block selection theorem under the diagonal-Hessian objective. Appendix B.1 describes baseline methods, while Appendix C.1 reports quantization settings, hardware specifications, and integration with vLLM. Appendix C.2 provides detailed per-task results, and Appendix C.3 reports kernel-level latency measurements.We plan to release the quantization and evaluation code, together with the ZipperEngine inference kernels, upon publication.

REFERENCES

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 *International Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proc. ACM Program. Lang.*, 7(OOPSLA1), April 2023. doi: 10.1145/3586030. URL https://doi.org/10.1145/3586030.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 34, pp. 7432–7439, 2020.

Hao Chen, Tao Han, Jie Zhang, Song Guo, and Lei Bai. Stcast: Adaptive boundary alignment for global and regional weather forecasting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

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. The language model evaluation harness, 07 2024.

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

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and XIAOJUAN QI. Slim-LLM: Salience-driven mixed-precision quantization for large language models. In *International Conference on Machine Learning (ICML)*, 2025.

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

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. In *International Conference on Learning Representations (ICLR)*, 2025.

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

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 Machine Learning and Systems (MLSys)*, 2025.

Lian Liu, Long Cheng, Haimeng Ren, Zhaohui Xu, Yudong Pan, Mengdi Wang, Xiaowei Li, Yinhe Han, and Ying Wang. Comet: Towards practical w4a4kv4 llms serving. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 131–146, 2025a.

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 *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.

NVIDIA. Nvidia blackwell architecture technical brief, February 2024. URL `https://resources.nvidia.com/en-us-blackwell-architecture`.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1525–1534, 2016.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, 2021.

Utkarsh Saxena, Sayeh Sharify, Kaushik Roy, and Xin Wang. Resq: Mixed-precision quantization of large language models with low-rank residuals. In *International Conference on Machine Learning (ICML)*, pp. 53095–53114, 2025.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, et al. Flatquant: Flatness matters for llm quantization. In *International Conference on Machine Learning (ICML)*, 2025.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Haodong Wang, Qihua Zhou, Zicong Hong, and Song Guo. D^2 moe: Dual routing and dynamic scheduling for efficient on-device moe-based llm serving. In *Mobile Computing and Networking (MobiCom)*, 2025.

Haodong Wang, Junjie Liu, Zicong Hong, Qianli Liu, Jian Lin, Song Guo, and Xu Chen. Twinquant: Learnable subspace decomposition for 4-bit llm quantization. *arXiv preprint arXiv:2606.01556*, 2026a.

Zile Wang, Qianli Liu, Kaibin Guo, Haodong Wang, Jian Lin, Zicong Hong, and Song Guo. Ppai: Enabling personalized llm agent interoperability for collaborative edge intelligence. *arXiv preprint arXiv:2605.18067*, 2026b.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning (ICML)*, pp. 38087–38099, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. Zeroquant-v2: Exploring post-training quantization in llms from comprehensive study to low rank compensation. *arXiv preprint arXiv:2303.08302*, 2023.

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

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. In *Proceedings of Machine Learning and Systems (MLSys)*, pp. 196–209, 2024.

A Proof of Δ-Hessian Block Selection

We prove that the proposed Δ-Hessian score gives the optimal block selection rule under a diagonal Hessian surrogate of the exact output-distortion objective. We then show how this surrogate approximates the exact output-distortion oracle.

Let $\mathbf{R}$ be the residual after dense 4-bit quantization, and let $\mathcal{Q}(\mathbf{R})$ be the 4-bit residual compensation candidate. For a selected block set $\mathcal{S}$, let $\mathbf{M}_{\mathcal{S}}$ be the corresponding block-wise binary mask. The remaining residual error after compensation is

$$\mathbf{E}_{\mathcal{S}} = \mathbf{R} - \mathbf{M}_{\mathcal{S}} \odot \mathcal{Q}(\mathbf{R}).$$

The exact layer-output distortion is

$$L(\mathcal{S}) = \frac{2}{N} \left\| \hat{\mathbf{X}} \mathbf{E}_{\mathcal{S}} \right\|_F^2,$$

where the constant factor $\frac{2}{N}$ does not affect block selection. Let

$$\mathbf{H} = \frac{2}{N} \hat{\mathbf{X}}^\top \hat{\mathbf{X}}$$

be the activation Hessian. Then

$$L(\mathcal{S}) = \mathrm{Tr}\left(\mathbf{E}_{\mathcal{S}}^\top \mathbf{H} \mathbf{E}_{\mathcal{S}} \right).$$

The exact output-distortion oracle selects blocks by maximizing

$$\Delta(\mathcal{S}) = L(\emptyset) - L(\mathcal{S}),$$

where $L(\emptyset)$ is the output distortion without residual compensation.

However, the full Hessian contains cross-channel interactions. Expanding the trace gives

$$L(\mathcal{S}) = \sum_j \sum_{i,k} H_{i,k} (\mathbf{E}_\mathcal{S})_{i,j} (\mathbf{E}_\mathcal{S})_{k,j}.$$

When $i \neq k$, the term $H_{i,k}$ couples the errors from different input channels. As a result, the contribution of one block depends on the errors left in other blocks, so the exact objective cannot be decomposed into independent block scores.

To obtain a decomposable surrogate, we approximate the Hessian by its diagonal:

$$\mathbf{D} = \mathrm{diag}(h_1, \ldots, h_{d_{\mathrm{in}}}), \qquad h_i = \frac{2}{N} \sum_{n=1}^{N} \hat{X}_{n,i}^2.$$

This gives the surrogate distortion

$$\widetilde{L}(\mathcal{S}) = \mathrm{Tr}\left(\mathbf{E}_\mathcal{S}^\top \mathbf{D} \mathbf{E}_\mathcal{S}\right) = \sum_{i,j} h_i \left(\mathbf{R}_{ij} - (\mathbf{M}_\mathcal{S} \odot \mathcal{Q}(\mathbf{R}))_{ij}\right)^2.$$

For a block B, define

$$A_B = \sum_{(i,j)\in B} h_i \mathbf{R}_{ij}^2$$

as the surrogate error contributed by block B without residual compensation, and

$$B_B = \sum_{(i,j)\in B} h_i \left(\mathbf{R}_{ij} - \mathcal{Q}(\mathbf{R})_{ij}\right)^2$$

as the surrogate error contributed by block B after applying residual 4-bit compensation. The error reduction of selecting block B is

$$\delta_B = A_B - B_B.$$

This is exactly the unclipped Δ-Hessian score:

$$\delta_B = \sum_{(i,j)\in B} h_i \mathbf{R}_{ij}^2 - \sum_{(i,j)\in B} h_i \left(\mathbf{R}_{ij} - \mathcal{Q}(\mathbf{R})_{ij}\right)^2.$$

The score used in the algorithm is

$$S_B = [\delta_B]_+.$$

Now we show that top-K selection is optimal for the surrogate objective. Because the blocks form a disjoint partition of the weight matrix, the surrogate distortion can be decomposed as

$$\widetilde{L}(\mathcal{S}) = \sum_{B \notin \mathcal{S}} A_B + \sum_{B \in \mathcal{S}} B_B.$$

Equivalently,

$$\widetilde{L}(\mathcal{S}) = \sum_B A_B - \sum_{B\in\mathcal{S}} (A_B - B_B) = \widetilde{L}(\emptyset) - \sum_{B\in\mathcal{S}} \delta_B.$$

Since $\widetilde{L}(\emptyset)$ is constant with respect to $\mathcal{S}$, minimizing $\widetilde{L}(\mathcal{S})$ under the budget constraint $|\mathcal{S}| \leq K$ is equivalent to

$$\max_\mathcal{S} \sum_{B\in\mathcal{S}} \delta_B, \qquad \text{s.t.} \quad |\mathcal{S}| \leq K.$$

This is a modular maximization problem under a cardinality constraint. Its optimal solution is to select the blocks with the largest positive gains δ_B, up to the budget K. Equivalently, one selects the top-K blocks according to $S_B = [\delta_B]_+$, omitting zero-score blocks if fewer than K blocks have positive gain. Therefore, Δ-Hessian selection is optimal for minimizing the diagonal Hessian surrogate $\widetilde{L}(\mathcal{S})$.

We further relate this surrogate to the exact output-distortion objective. Let

$$\mathbf{H} = \mathbf{D} + \mathbf{O},$$

where $\mathbf{O}$ contains the off-diagonal Hessian terms. For any selected block set $\mathcal{S}$,

$$L(\mathcal{S}) - \widetilde{L}(\mathcal{S}) = \mathrm{Tr}\left(\mathbf{E}_{\mathcal{S}}^{\top}\mathbf{O}\mathbf{E}_{\mathcal{S}}\right).$$

Using the spectral norm bound,

$$\left|L(\mathcal{S}) - \widetilde{L}(\mathcal{S})\right| \leq \|\mathbf{O}\|_2 \|\mathbf{E}_{\mathcal{S}}\|_F^2.$$

Similarly, for the output-distortion reduction,

$$\Delta(\mathcal{S}) = L(\emptyset) - L(\mathcal{S}), \qquad \widetilde{\Delta}(\mathcal{S}) = \widetilde{L}(\emptyset) - \widetilde{L}(\mathcal{S}),$$

we have

$$\left|\Delta(\mathcal{S}) - \widetilde{\Delta}(\mathcal{S})\right| \leq \|\mathbf{O}\|_2 \left(\|\mathbf{R}\|_F^2 + \|\mathbf{E}_{\mathcal{S}}\|_F^2\right).$$

Define

$$\epsilon_K = \max_{|\mathcal{S}| \leq K} \left|\Delta(\mathcal{S}) - \widetilde{\Delta}(\mathcal{S})\right|.$$

Let $\mathcal{S}^\star$ be the exact output-distortion oracle, i.e.,

$$\mathcal{S}^\star = \arg\max_{|\mathcal{S}| \leq K} \Delta(\mathcal{S}),$$

and let $\widehat{\mathcal{S}}$ be the set selected by Δ-Hessian, i.e.,

$$\widehat{\mathcal{S}} = \arg\max_{|\mathcal{S}| \leq K} \widetilde{\Delta}(\mathcal{S}).$$

Then

$$\Delta(\mathcal{S}^\star) \leq \widetilde{\Delta}(\mathcal{S}^\star) + \epsilon_K \leq \widetilde{\Delta}(\widehat{\mathcal{S}}) + \epsilon_K \leq \Delta(\widehat{\mathcal{S}}) + 2\epsilon_K.$$

Therefore,

$$\Delta(\mathcal{S}^\star) - \Delta(\widehat{\mathcal{S}}) \leq 2\epsilon_K.$$

This shows that the gap between the exact output-distortion oracle and Δ-Hessian selection is controlled by the approximation error between the full Hessian objective and its diagonal surrogate. In particular, if the off-diagonal Hessian term is zero, i.e., $\mathbf{O} = 0$, then $\epsilon_K = 0$, and Δ-Hessian selection exactly matches the output-distortion oracle. When the off-diagonal interactions are small, Δ-Hessian provides a close block-wise approximation while allowing efficient independent ranking.

B Benchmark Details

B.1 Baselines

We benchmark our methods compared with the following six baselines:

- RTN (Yao et al., 2023) employs uniform rounding-to-nearest with fixed scaling to quantize weights and activations without calibration, thereby offering a almost zero-cost baseline at the expense of larger accuracy loss.
- GPTQ (Frantar et al., 2023) employs post-training, Hessian-aware greedy weight quantization while applying error compensation to previously quantized rows, thereby preserving accuracy under W4A16.
- AWQ (Lin et al., 2024) employs activation-aware weight clipping and per-channel scaling while prioritizing salient channels, thereby achieving accurate W4A16 quantization with light calibration.
- SmoothQuant (Xiao et al., 2023) employs channel-wise scaling to shift activation outliers into weights while smoothing activation distributions, thereby enabling W8A8 quantization with minimal degradation.
- QuaRot (Ashkboos et al., 2025) employs online Hadamard-based orthogonal rotations to on-the-fly disperse outliers and quantize in the rotated space, thereby enabling W4A4 quantization with minimal overhead and small loss.
- SpinQuant (Liu et al., 2025b) employs learned orthogonal rotations to homogenize weights and activations magnitudes while reducing outliers, thereby enabling W4A4 quantization with improved stability.

C Additional Results

C.1 Implementation Details

We perform MosaicQuant quantization and all accuracy evaluations on a single NVIDIA H200 GPU. These evaluations include WikiText-2 perplexity and accuracy on the six zero-shot tasks for both the quantized models and the W16A16 references. System performance measurements of MosaicQuant, including end-to-end throughput and kernel latency, use its INT4 implementation on a workstation equipped with an NVIDIA RTX 4090 GPU (24 GB), an Intel Xeon Gold 6430 CPU, and 120 GB of DDR5 host memory. The reported FP4 results are limited to accuracy evaluation.

For W4A8, we use dynamic per-token activation quantization and per-channel weight quantization. For W4A4, activations and weights use per-group and per-channel symmetric quantization, respectively.

We integrate MosaicQuant into vLLM (Kwon et al., 2023), with ZipperEngine serving as a custom execution backend for the quantized linear layers. vLLM handles model execution, request scheduling, and KV-cache management, while ZipperEngine jointly computes the dense 4-bit base and the selected sparse 4-bit residual blocks through its fused dense–sparse execution pipeline. This integration enables end-to-end evaluation of MosaicQuant within the vLLM runtime.

The smoothing factor $\boldsymbol{\lambda} \in \mathbb{R}^m$ is a per-channel vector whose i-th element is computed as $\lambda_i = \max(|\mathbf{X}_{:,i}|)^{\alpha} / \max(|\mathbf{W}_{i,:}|)^{1-\alpha}$ where $\mathbf{X} \in \mathbb{R}^{t \times m}$ and $\mathbf{W} \in \mathbb{R}^{m \times n}$ following SmoothQuant (Xiao et al., 2023). The migration strength α is chosen offline, per layer, by searching for the value that minimizes the layer output mean squared error (MSE) after MosaicQuant on the calibration dataset.

For the baselines, we run AWQ (Lin et al., 2024), FlatQuant (Sun et al., 2025), QuaRot (Ashkboos et al., 2025), and QServe (Lin et al., 2025) using their publicly released implementations. For AWQ, FlatQuant, and QuaRot, we adopt the official PyTorch-based code and recommended W4A4/W4A8 configurations for each model, and ensure that all methods share the same decoding setup (greedy decoding), batch sizes, and sequence lengths on the same GPU. For QServe, we build its C++/CUDA runtime following the official instructions and evaluate the W4A8 configuration on the same hardware and workload as MosaicQuant. All throughput measurements are collected after warm-up runs to avoid initialization overhead.

C.2 Zero-Shot Task Results

In Table 3 and Table 4, we show the complete results of Table 1. including ARC-Challenge, ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), PIQA(Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021). Additionally, we also report the perplexity score on WikiText2 (Merity et al., 2017) datasets. We compare our results with previous works including RTN (Yao et al., 2023), GPTQ (Frantar et al., 2023), AWQ (Lin et al., 2024), SmoothQuant (Xiao et al., 2023), QuaRot (Ashkboos et al., 2025) and SpinQuant (Liu et al., 2025b).

C.3 Kernel Profile

Table 5 shows that kernel fusion consistently reduces the latency of the main GEMM operators on Qwen3-8B. For attention, the fused $\text{qkv}_{\text{fused}}$ operator obtains 2.51–$2.56\times$ prefill speedup and 1.61–$1.79\times$ decode speedup across batch sizes, while o_{proj} improves by 2.39–$2.46\times$ in prefill and 1.45–$1.87\times$ in decode. For MLP layers, fusing $\text{gate}/\text{up}_{\text{proj}}$ gives the largest prefill gain, reaching 2.64–$2.68\times$, with 1.25–$1.45\times$ decode speedup. The $\text{down}_{\text{proj}}$ operator shows a different pattern: its prefill speedup is 1.92–$1.97\times$, but its decode speedup increases with batch size and reaches $2.90\times$ at batch size 32. Overall, these results show that kernel fusion is effective across both attention and MLP projections, with particularly strong gains in prefill and for decode in $\text{down}_{\text{proj}}$.

Table 3: Complete comparison of the perplexity score on WikiText2 and averaged accuracy on six zero-shot tasks on LLaMA3 3B and 8B.

Model	Precision	Method	ARC-C	ARC-E	HellaSwag	PIQA	Winogrande	LAMBADA	Avg.	Wiki2 (↓)
3B	W16A16	–	47.6	69.9	71.0	76.0	66.6	65.9	66.2	10.7
	W4A16	RTN	41.3	60.2	66.2	73.1	63.0	61.6	60.9	18.8
	W4A16	GPTQ	41.4	60.8	65.9	73.6	65.0	63.4	61.7	15.2
	W4A16	AWQ	43.5	66.7	65.8	75.8	62.7	63.8	63.1	12.7
	W4A8	RTN	42.6	60.2	66.2	72.6	62.7	60.1	60.7	29.0
	W4A8	SmoothQuant	40.7	59.8	65.5	73.8	58.5	60.7	59.8	288.5
	W4A8	QuaRot	46.9	67.3	68.1	75.6	64.8	62.2	64.2	12.4
	W4A8	SpinQuant	47.2	68.8	69.2	76.0	66.7	63.2	65.2	11.5
	W4A8	ResQ	44.4	69.4	70.2	75.2	65.2	64.2	64.8	11.8
	W4A8	Atom	44.2	68.8	70.5	74.4	65.6	63.4	64.5	11.9
	W4A8	MosaicQuant(INT4)	46.8	69.0	70.2	75.2	65.8	64.8	65.3	11.3
	W4A4	RTN	29.8	41.0	41.4	57.3	50.9	38.9	43.2	741.9
	W4A4	SmoothQuant	30.5	43.6	37.7	58.0	52.9	45.3	44.7	372.3
	W4A4	QuaRot	42.3	64.9	64.8	72.9	63.8	59.8	61.4	16.9
	W4A4	SpinQuant	44.9	68.3	67.2	75.0	65.5	63.5	64.1	11.9
	W4A4	ResQ	43.1	66.6	68.4	74.1	63.2	62.8	63.0	12.2
	W4A4	Atom	42.8	66.4	68.8	73.2	62.1	62.5	62.6	12.5
	W4A4	MosaicQuant(INT4)	45.8	69.4	69.8	74.5	64.8	64.8	64.9	11.8
8B	W16A16	–	54.86	79.55	79.13	80.74	73.72	72.93	73.5	8.64
	W4A16	RTN	52.5	77.7	77.4	79.6	73.4	66.7	71.2	10.5
	W4A16	GPTQ	54.2	79.4	78.1	80.1	72.1	70.6	72.4	9.0
	W4A16	AWQ	53.8	77.6	78.8	80.0	73.2	72.1	72.6	10.3
	W4A8	RTN	52.7	77.0	77.3	79.4	73.0	66.3	71.0	10.6
	W4A8	SmoothQuant	42.5	66.2	69.6	73.72	66.1	57.5	62.6	13.3
	W4A8	QuaRot	53.4	78.0	77.9	79.7	71.0	69.5	71.6	10.6
	W4A8	SpinQuant	54.0	78.5	78.1	79.6	72.4	71.3	72.3	10.3
	W4A8	ResQ	54.2	78.9	78.1	78.6	71.8	71.5	72.2	10.4
	W4A8	Atom	53.8	78.1	78.3	77.9	72.2	71.2	71.9	10.6
	W4A8	MosaicQuant(INT4)	54.3	79.2	78.5	79.1	72.5	72.3	72.7	10.2
	W4A4	RTN	28.3	40.2	43.8	59.1	50.0	21.0	40.4	92.9
	W4A4	SmoothQuant	35.0	57.3	59.8	67.9	56.8	40.2	52.8	19.4
	W4A4	QuaRot	49.2	73.4	74.1	77.4	68.3	65.8	68.0	13.1
	W4A4	SpinQuant	51.9	75.2	75.9	77.5	70.8	68.4	70.0	11.4
	W4A4	ResQ	51.6	76.5	75.2	76.3	70.5	69.4	69.9	11.6
	W4A4	Atom	51.4	75.5	74.4	76.2	68.8	66.3	68.8	12.5
	W4A4	MosaicQuant(INT4)	52.7	77.1	76.3	78.8	71.3	70.8	71.2	10.8

Table 4: Complete comparison of the perplexity score on WikiText2 and averaged accuracy on six zero-shot tasks on Qwen3 4B/8B/14B/32B.

Model	Precision	Method	ARC-C	ARC-E	HellaSwag	PIQA	Winogrande	LAMBADA	Avg.	Wiki2 (↓)
4B	W16A16	–	58.6	81.1	69.09	76.0	68.11	63.56	69.4	10.04
	W4A16	RTN	56.7	77.3	66.8	73.2	66.8	61.9	67.1	10.6
	W4A16	GPTQ	58.1	80.1	68.4	75.6	67.0	62.1	68.6	10.4
	W4A16	AWQ	57.2	80.7	68.8	75.8	66.1	61.5	68.4	10.4
	W4A8	RTN	42.3	68.7	64.1	68.8	53.9	55.1	58.8	30.6
	W4A8	SmoothQuant	42.2	69.5	63.4	71.5	57.5	55.0	59.9	22.6
	W4A8	QuaRot	55.3	79.5	68.3	74.7	67.2	61.8	67.8	11.2
	W4A8	SpinQuant	56.2	80.4	68.2	75.5	67.6	62.4	68.4	10.8
	W4A8	ResQ	56.4	80.7	67.8	75.2	67.8	62.6	68.4	10.7
	W4A8	Atom	56.0	79.8	67.4	74.8	66.9	62.1	67.8	11.0
	W4A8	MosaicQuant (INT4)	52.45	77.35	68.90	77.50	65.04	60.74	67.00	12.27
	W4A8	MosaicQuant (FP4)	52.36	78.52	69.09	77.21	65.44	61.04	67.28	12.10
	W4A4	RTN	26.7	28.6	44.2	48.9	51.8	48.9	41.5	8791
	W4A4	SmoothQuant	22.6	25.9	42.3	51.6	47.9	50.2	40.1	9910
	W4A4	QuaRot	54.93	70.58	59.07	73.29	58.72	53.46	61.7	12.5
	W4A4	SpinQuant	55.4	76.8	66.5	74.8	63.5	56.7	65.6	11.5
	W4A4	ResQ	54.5	76.8	66.8	73.4	62.8	56.5	65.1	12.0
	W4A4	Atom	54.8	76.2	66.9	73.0	62.5	55.4	64.8	12.3
	W4A4	MosaicQuant (INT4)	45.07	70.30	67.37	74.76	61.39	56.75	62.61	13.22
	W4A4	MosaicQuant (FP4)	51.77	76.48	69.35	77.70	66.25	59.90	66.91	12.25

Continued on next page

Table 4 – continued from previous page

Model	Precision	Method	ARC-C	ARC-E	HellaSwag	PIQA	Winogrande	LAMBADA	Avg.	Wiki2 (↓)
8B	W16A16	–	55.5	83.5	78.8	76.4	68.0	67.4	71.6	9.71
	W4A16	RTN	53.8	79.1	76.3	75.4	63.6	65.6	69.0	12.0
	W4A16	GPTQ	55.6	80.1	77.8	76.0	66.1	66.8	70.4	10.8
	W4A16	AWQ	54.7	83.1	77.1	75.4	67.5	66.7	70.8	10.2
	W4A8	RTN	45.8	73.4	72.3	73.1	62.9	58.7	64.4	12.3
	W4A8	SmoothQuant	44.3	71.0	71.7	73.4	62.1	60.0	63.8	12.5
	W4A8	QuaRot	53.9	77.8	74.8	72.7	62.6	60.5	67.1	11.5
	W4A8	SpinQuant	54.2	81.9	77.5	75.6	66.5	66.8	70.4	10.8
	W4A8	ResQ	54.5	81.8	77.4	75.6	66.2	66.4	70.3	10.7
	W4A8	Atom	55.0	82.5	76.5	75.2	66.8	65.8	70.3	10.9
	W4A8	MosaicQuant (INT4)	55.74	79.64	74.46	77.52	68.22	63.88	69.91	10.08
	W4A8	MosaicQuant (FP4)	56.69	79.78	74.90	78.51	69.50	65.19	70.76	9.94
	W4A4	RTN	22.6	24.9	45.6	48.9	51.8	46.9	40.1	4392
	W4A4	SmoothQuant	25.7	25.5	41.7	50.5	52.2	44.9	40.1	3360.1
	W4A4	QuaRot	49.8	77.8	69.8	74.3	65.7	60.1	66.3	15.6
	W4A4	SpinQuant	53.6	78.5	71.4	76.5	67.3	62.4	68.3	11.4
	W4A4	ResQ	52.1	78.8	73.8	71.5	63.5	61.1	66.8	14.2
	W4A4	Atom	51.8	76.6	72.6	71.2	62.8	61.5	66.1	14.5
	W4A4	MosaicQuant (INT4)	51.01	76.30	72.92	75.80	65.67	62.59	67.38	10.61
	W4A4	MosaicQuant (FP4)	55.96	79.90	74.52	77.19	69.37	66.29	70.54	10.09
14B	W16A16	–	59.0	84.3	80.5	80.0	72.9	68.4	74.2	8.6
	W4A16	RTN	51.8	80.6	78.5	77.9	68.7	64.4	70.3	9.9
	W4A16	GPTQ	57.1	81.9	78.8	78.8	72.5	66.9	72.7	9.2
	W4A16	AWQ	51.7	79.2	79.3	76.0	66.1	66.5	69.8	9.6
	W4A8	RTN	50.7	79.5	75.4	75.3	66.9	62.0	68.3	10.9
	W4A8	SmoothQuant	50.5	79.0	75.8	75.8	66.3	61.3	68.1	11.0
	W4A8	QuaRot	56.4	80.7	76.1	76.9	68.5	65.8	70.7	10.4
	W4A8	SpinQuant	57.8	81.8	77.6	78.3	71.8	67.6	72.5	9.8
	W4A8	ResQ	57.2	82.5	78.4	77.5	72.0	66.6	72.4	12.1
	W4A8	Atom	57.5	81.4	78.2	77.7	71.5	66.2	72.1	12.4
	W4A8	MosaicQuant (INT4)	59.90	82.91	77.96	78.89	72.77	68.95	73.56	8.79
	W4A8	MosaicQuant (FP4)	60.41	81.52	78.51	79.76	72.61	68.97	73.63	8.78
	W4A4	RTN	24.8	23.9	50.6	55.7	46.8	50.7	42.1	18749
	W4A4	SmoothQuant	26.5	25.8	48.7	51.2	50.2	45.3	41.3	21675
	W4A4	QuaRot	52.8	76.4	72.4	73.6	65.0	62.8	67.2	18.2
	W4A4	SpinQuant	56.3	81.0	77.8	76.9	71.4	66.7	71.7	12.0
	W4A4	ResQ	56.7	80.5	76.4	76.1	70.8	65.6	71.0	12.8
	W4A4	Atom	56.1	80.2	75.5	75.2	70.2	65.7	70.5	14.5
	W4A4	MosaicQuant (INT4)	56.88	81.09	77.27	77.79	71.27	66.85	71.86	9.29
	W4A4	MosaicQuant (FP4)	58.96	81.36	78.69	79.00	73.51	69.49	73.50	9.00
32B	W16A16	–	57.8	84.4	84.2	80.9	73.6	70.3	75.2	7.6
	W4A16	RTN	49.7	73.0	82.6	71.9	62.9	68.5	68.1	38.5
	W4A16	GPTQ	57.8	82.6	83.6	79.7	67.6	69.8	73.5	8.3
	W4A16	AWQ	56.9	82.5	83.2	79.8	71.8	68.7	73.8	8.2
	W4A8	RTN	49.5	76.3	78.6	72.9	65.3	62.0	67.4	11.2
	W4A8	SmoothQuant	49.2	75.8	78.9	71.4	63.5	63.7	67.1	11.6
	W4A8	QuaRot	51.3	78.9	79.8	76.8	70.9	68.5	71.0	10.5
	W4A8	SpinQuant	56.6	82.9	80.4	79.4	72.0	67.1	73.1	8.9
	W4A8	ResQ	55.7	82.9	81.2	79.6	71.8	66.8	73.0	9.2
	W4A8	Atom	55.4	82.3	80.9	80.4	72.4	66.6	73.0	9.3
	W4A8	MosaicQuant (INT4)	60.49	82.28	81.71	80.58	73.72	67.98	74.46	7.84
	W4A8	MosaicQuant (FP4)	62.71	83.16	82.10	81.23	71.74	68.10	74.84	7.70
	W4A4	RTN	28.5	27.6	60.8	52.5	50.7	52.0	45.4	1796
	W4A4	SmoothQuant	26.0	28.9	56.7	51.9	54.1	48.6	44.4	1806
	W4A4	QuaRot	49.8	72.2	75.6	73.4	67.9	65.3	67.4	15.6
	W4A4	SpinQuant	56.7	81.6	81.2	78.9	71.8	65.8	72.7	10.3
	W4A4	ResQ	56.9	81.6	80.1	78.4	71.2	63.5	72.0	10.9
	W4A4	Atom	57.5	81.2	80.6	78.2	71.5	63.9	72.2	10.5
	W4A4	MosaicQuant (INT4)	57.56	80.43	80.96	78.76	69.25	66.68	72.27	8.51
	W4A4	MosaicQuant (FP4)	60.76	82.57	82.67	80.52	72.21	68.94	74.61	8.02

Table 5: Latency comparison with/without kernel fusion on Qwen3-8B in Environment 1. The fused operators $\text{qkv}_{\text{fused}}$ and $\text{gate_up}_{\text{fused}}$ merge the original $q/k/v_{\text{proj}}$ and $\text{gate}/\text{up}_{\text{proj}}$ into a single GEMM with concatenated output dimensions.

GEMM Operator	Batch Size	without Kernel Fusion		with Kernel Fusion		Speedup	
		Prefill (ms)	Decode (ms)	Prefill (ms)	Decode (ms)	Prefill	Decode
$\text{qkv}_{\text{fused}}$ (Fused $q/k/v_{\text{proj}}$) 6144×4096	4	4.536	0.067	1.806	0.037	2.51×	1.79×
	8	9.070	0.066	3.581	0.038	2.53×	1.74×
	16	18.167	0.066	7.101	0.038	2.56×	1.74×
	32	36.344	0.062	14.270	0.038	2.55×	1.61×
o_{proj} 4096×4096	4	3.123	0.066	1.307	0.035	2.39×	1.87×
	8	6.301	0.065	2.624	0.038	2.40×	1.73×
	16	12.538	0.055	5.094	0.038	2.46×	1.45×
	32	25.043	0.062	10.228	0.038	2.45×	1.63×
$\text{gate_up}_{\text{fused}}$ (Fused $\text{gate}/\text{up}_{\text{proj}}$) 24576×4096	4	17.171	0.076	6.498	0.060	2.64×	1.26×
	8	34.343	0.077	12.835	0.061	2.68×	1.25×
	16	68.780	0.082	25.949	0.064	2.65×	1.29×
	32	137.811	0.097	52.298	0.067	2.64×	1.45×
$\text{down}_{\text{proj}}$ 4096×12288	4	9.922	0.148	5.161	0.077	1.92×	1.93×
	8	19.824	0.148	10.141	0.059	1.95×	2.49×
	16	39.621	0.158	20.174	0.060	1.96×	2.63×
	32	79.616	0.180	40.376	0.062	1.97×	2.90×