SCVQ: Sparse-Compensated Vector Quantization for Large Language Models

Zixuan Zhou[*], **Yujun Diao, Zicheng Kong, Dehua Ma, Zhenbo Xu, Peipei Li, Zhaofeng He** [†]
Beijing University of Posts and Telecommunications

Abstract

Large language models are primarily constrained by computing and bandwidth limitations during hardware deployment. However, current vector quantization methods incur substantial inference overhead, mainly because large-scale codebook storage and frequent index lookups consume excessive memory and compute resources. To address these challenges, we present *Sparse-Compensated Vector Quantization (SCVQ)*, a salience and sparse-compensated vector quantization framework. By integrating a salience-aware weighted K-means clustering scheme with symmetry constraints, SCVQ significantly reduces codebook size and indexing costs. Crucially, we design a structured sparse representation that unifies outliers, salient weights, and quantization residuals into a single sparse matrix to to maintain model performance with minimal overhead, thereby addressing the challenges of aggressive compression. Extensive experiments across multiple benchmarks validate the strong low-bit quantization performance of SCVQ over current methods. Specifically, SCVQ achieves a perplexity of 5.87 on the WikiText-2 dataset at 2-bit quantization for LLaMA-2-7B, while delivering a $1.4\times$ end-to-end inference speedup on an NVIDIA A100 GPU compared to the baseline.

1 Introduction

Large Language Models (LLMs) have demonstrated transformative potential across diverse NLP applications, ranging from intelligent conversational agents to complex reasoning tasks (Brown et al., 2020; Touvron et al., 2023; Kaplan et al., 2020; Chen et al., 2021). However, their massive scale—often exceeding tens of billions of parameters—imposes prohibitive memory and computational demands, particularly on resource-constrained hardware. This bottleneck not only hinders the democratization of these models but also undermines the performance of latency-sensitive applications. Consequently, post-training quantization (PTQ) (Lin et al., 2024; Nagel et al., 2021; Dettmers et al., 2022; Chee et al., 2023; Dettmers et al., 2023; Kim et al., 2024) has emerged as a pivotal technique to reconcile the immense capabilities of LLMs with the practicalities of real-world deployment.

Conventional PTQ typically employs scalar quantization, treating weight elements independently. Neglecting structural correlations within weight distributions, these approaches suffer from significant performance degradation with extreme quantization settings. To address this, recent studies (Egiazarian et al., 2024; Tseng et al., 2024; Liu et al., 2024a; van Baalen et al., 2025) have explored vector quantization (VQ), which enables superior reconstruction quality by quantizing weight blocks as multi-dimensional vectors.

Our analysis identifies two primary limitations in existing VQ methods. First, standard VQ assigns uniform importance to all weights, neglecting that LLM performance is predominantly driven by a small fraction of salient weights (Lin et al., 2024). Second, explicit-codebook-based approaches rely on Look-Up Table (LUT) operations during decoding, where large-scale codebooks and frequent indexing incur substantial memory bandwidth overhead, thereby bottlenecking end-to-end inference throughput. Consequently, these approaches struggle to achieve an optimal trade-off between codebook storage costs and reconstruction accuracy. Alternatively, methods relying on implicit codebooks (Tseng et al., 2024) often necessitate computationally intensive transformations (e.g., Hadamard transforms), which complicate the deployment pipeline and introduce non-trivial runtime overhead.

[*]Part of this work was done during an internship at Qualcomm AI Research.

[†]Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 8934–8950

To achieve high-quality and hardware-efficient quantization, we propose *Sparse-Compensated Vector Quantization (SCVQ)*. To ensure our method is seamlessly integrable without introducing additional decoding complexity, we build upon the standard K-means-based vector quantization framework. To address the aforementioned limitations, we develop a salience-aware weighted K-means clustering scheme with symmetry constraints. Our key inovation is predicated on the insight that explicitly isolating and compensating for critical weight components enables a more compact and accurate representation. We introduce a unified representation that consolidates outliers, salient weights, and quantization residuals into a single sparse compensate matrix. Our work is the first to **integrate VQ with sparse compensation**, offering two critical advantages. First, we empirically observe that outliers and salient weights tend to be contiguously aligned along the rows of the weight matrix, which perfectly aligns with the block-wise grouping scheme of VQ. Second, leveraging this contiguous distribution, we design a specialized sparse storage format *VSR*, along with optimized CUDA kernels. This design ensures that the inference latency introduced by sparse compensation remains negligible.

2 Background and Related Works

2.1 LLM Quantization

Quantization(Nagel et al., 2021; Gholami et al., 2022) is the process of converting values from high-precision to low-precision counterparts. For LLMs, in the transformer architecture, the weight matrix of a linear layer is $W \in \mathbb{R}^{m \times n}$, the simple Round-to-Nearest (RTN) linear quantization with bit width b can be formulated as:

$$\hat{W} = \text{clamp}\left(\left\lfloor \frac{W}{s} \right\rceil + z, 0, 2^b - 1\right) \quad (1)$$

Where scale factor is formulated as $s = \frac{\max(W) - \min(W)}{2^b - 1}$, and zero point is formulated as $z = -\lfloor \frac{\min(W)}{s} \rceil$. We focus on weight-only post-training quantization(Lin et al., 2024; Frantar et al., 2022; Kim et al., 2024; Chee et al., 2023; Dettmers et al., 2023; Nagel et al., 2020) of LLMs in this work, which means quantizing weights of pre-trained models while keeping the activations in full precision.

2.2 Vector Quantization

Vector quantization (Gray, 1984) operates by partitioning model weights into d-dimensional subvectors, which are then mapped to the nearest centroids stored in a shared codebook $\mathcal{B}$. By replacing high-precision weights with low-bitwidth indices, VQ significantly reduces storage requirements. Unlike conventional PTQ methods that treats weights W_{ij} as independent scalars, VQ captures intra-vector correlations, theoretically yielding a higher signal-to-noise ratio at equivalent bitrates. However, VQ faces the "curse of dimensionality": codebook size 2^{kd} scales exponentially with bitwidth k and dimension d, making LUTs computationally prohibitive and hardware-inefficient.

Recent research has introduced several VQ frameworks tailored for LLM quantization scenarios. In contrast to AQLM and VPTQ(Egiazarian et al., 2024; Liu et al., 2024a), our method obviates the need for additional codebooks by leveraging sparse matrices to store key components, such as quantization residuals. This design not only mitigates the inference latency inherent in multi-codebook schemes but also more effectively prevents performance degradation. Compared to QuIP#(Tseng et al., 2024), although our approach incurs some memory overhead due to the use of explicit codebooks, it eliminates the reliance on Hadamard transforms for outlier suppression during inference, thereby offering superior hardware compatibility.

2.3 Sparse Compensation

LLMs exhibit "outlier" characteristics in both activations and weights—systematic, high-magnitude values that, despite their sparsity, are critical for numerical stability and performance. Originally observed in activations (Dettmers et al., 2022),SpQR and SqueezeLLM (Dettmers et al., 2023; Kim et al., 2024) preserves sensitive weights $< 1\%$ in high-precision sparse formats to prevent perplexity degradation at 3–4 bits. In contrast, our method unifies more diverse compensation terms—including outliers, salient weights and residuals—into a single structured sparse matrix, while fully leveraging the properties arising from its integration with VQ.

3 Methodology

A comprehensive overview of the SCVQ pipeline is presented in Figure 2. To complement this, Algorithm 1 outlines the core algorithmic steps, with

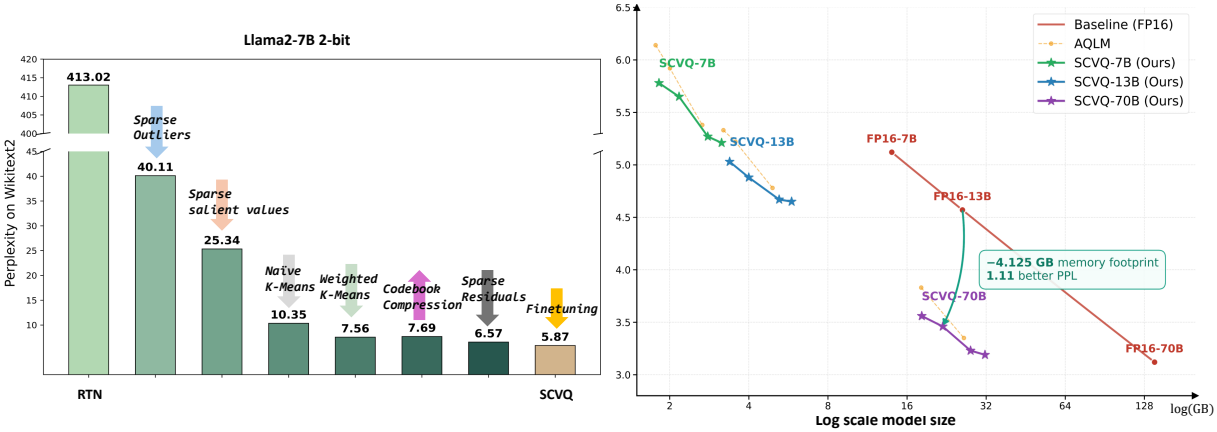

Figure 1: (Left) By applying our methods to LLaMA models of varying sizes, we can achieve improved Pareto-frontier between perplexity and model size than AQLM (Egiazarian et al., 2024) baseline. (Right) Perplexity of the LLaMA-2-7B model across different optimization stages under SCVQ at 2 bits per weight. During codebook compression, we shrink the codebook to 1/4 of its original size, incurring only a minor perplexity increase of 0.13.

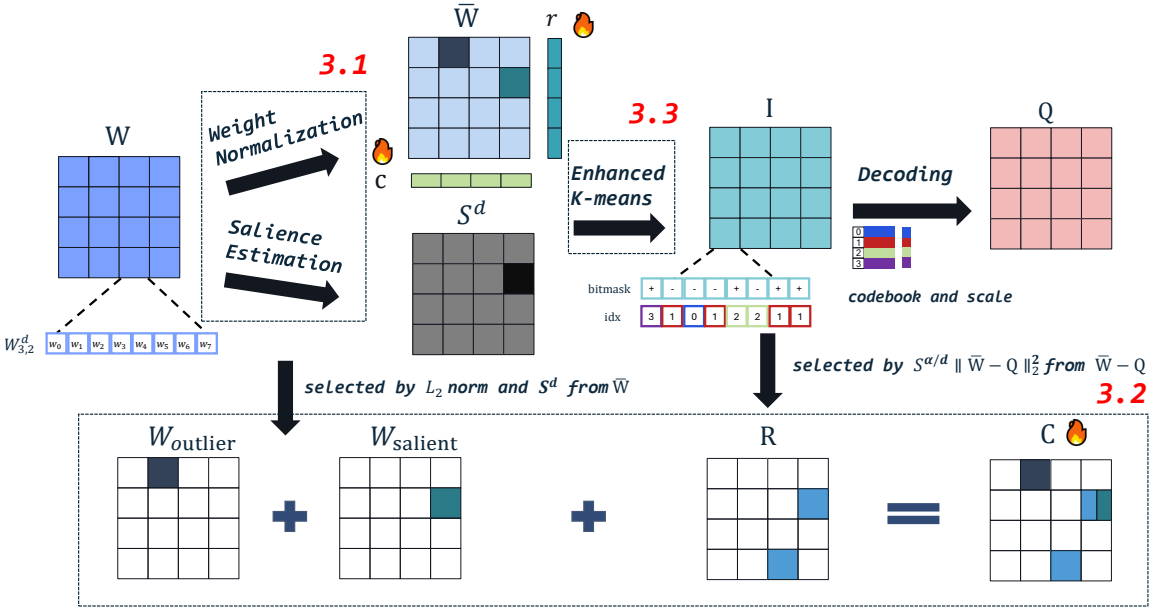

Figure 2: Overview of the SCVQ quantization pipeline. To recover model performance, we treat the normalization scaling vectors and sparse compensation matrices as trainable components during the fine-tuning phase. In the diagram, each square block denotes an individual weight sub-vector when $d = 8$. Furthermore, red annotations enclosed in dashed boundaries refer to the respective sections for each step.

the fine-tuning phase described separately in Section 3.4.

3.1 Normalization and Salience Estimation

Bi-Normalization To align the weight distribution with the geometric assumptions of K-means clustering, we apply *bi-normalization* to $W \in \mathbb{R}^{m \times n}$. By independently rescaling rows and columns, this procedure encourages near-isotropic statistics, approximating the isotropic Gaussian distribution that minimizes quantization distortion (Guo et al., 2021; Bishop, 2006). Formally, the normalized entry $\bar{W}_{ij}$ is given by:

$$\bar{W}_{ij} = \frac{W_{ij}}{r_i \cdot c_j}, \qquad (2)$$

$$r_i = \sqrt{\sum_{j=1}^{n} W_{ij}^2} \quad c_j = \sqrt{\sum_{i=1}^{m} \left(\frac{W_{ij}}{r_i} \right)^2} \qquad (3)$$

By equalizing the scale across both dimensions, bi-normalization mitigates the bias that outliers—typically concentrated along specific rows or columns—would otherwise introduce into subsequent quantization steps (Dettmers et al., 2023, 2022), thereby enhancing quantization robustness. A detailed theoretical analysis of how this normalization aligns with spherical clustering assumptions is provided in Appendix A.

Salience Factor A minute fraction of weights—typically less than 1%—disproportionately impacts model performance (Lin et al.,

Algorithm 1: SCVQ Quantization Pipeline

Input : LLM Π, weights $W \in \mathbb{R}^{m\times n}$, calibration datasets D_1, D_2; sub-vector dim d, iterations n_{iter}, centroid count k, symmetry dims $\bar{d}$, sparse ratios: $\{r_{\text{outlier}}, r_{\text{salient}}, r_{\text{residual}}\}$.

Output : quantized codebook $\hat{\mathcal{B}}$ with scales s, quantized matrix I, sparse matrix C, symmetry axe set D, scaling vectors r and c.

$S^{\text{local}} \leftarrow \text{LocalSalienceFactor}(D_1, \Pi, W)$;
$S^{\text{global}} \leftarrow \text{GlobalSalienceFactor}(D_2, \Pi, W)$;
$S \leftarrow S^{\text{local}} \odot S^{\text{global}}$;
$(r, c, \bar{W}) \leftarrow \text{Bi-normalization}(W)$;
Sub-vectors$\{W^v_{i,b}\} \leftarrow$ Group $\bar{W}$ into matrix of weight sub-vectors $W^v \in \mathbb{R}^{m\times(n/d)}$;
$S^v_{i,b} \leftarrow \sum_{k=1}^{d} S_{i,bd+k-1}$;
$W_{\text{outlier}} \leftarrow \{W^v_{i,b} \mid \|W^v_{i,b}\|_2 \in \text{Top-}r_{\text{outlier}} \text{ of } W^v\}$;
$W_{\text{salient}} \leftarrow \{W^v_{i,b} \mid S^v_{i,b} \in \text{Top-}r_{\text{salient}} \text{ of } S^v\}$;
$D \leftarrow \text{KSTest}(\bar{d}, W^v_{i,b})$;
$(\text{bitmask}, \mathbf{V}_{\text{sym}}) \leftarrow \text{SymmetryTransform}(\{\mathbf{v}_i\}, D)$;
$k' \leftarrow k \cdot 2^{\bar{d}} - 1$;
$(B, \text{idx}) \leftarrow \text{WeightedKMeans}(\mathbf{V}_{\text{sym}}, S_v, k', n_{\text{iter}})$;
$I \leftarrow \text{BitPack}(\text{idx}, \text{bitmask})$;
$(\hat{\mathcal{B}}, s) \leftarrow \text{QuantizeCodebook}(\mathcal{B})$;
$Q \leftarrow \text{Decode}(I, D, \hat{\mathcal{B}}, s)$;
$R \leftarrow \{(W^v_{i,b} - Q_{i,b}) \mid M_{i,b} \in \text{Top-}r_{\text{residual}} \text{ of } M\}$;
$C \leftarrow W_{\text{outlier}} + W_{\text{salient}} + R$
return $\hat{\mathcal{B}}, s, I, C, D, r, c$

2024; Frantar et al., 2022; Kim et al., 2024). Accurately identifying these critical parameters requires a dual perspective: capturing local weight-activation dynamics while integrating global, end-to-end model information. Guided by this principle, we introduce a composite salience factor S_{ij} to quantify the quantization sensitivity of the corresponding weight W_{ij}:

- **Local Salience (S^{local}):** The input to a linear layer for LLM is an activation tensor $\mathbf{X} \in \mathbb{R}^{(N\times L)\times m}$, where N, L, and m denote the batch size, sequence length, and input feature dimension, respectively. To quantify the activation magnitude interacting with weight W_{ij}, we compute the L_2 norm of the j-th feature column averaged across all $N \times L$ tokens:

$$S^{\text{local}}_{ij} = \frac{1}{|D_1|} \sum_{D_1} \|\mathbf{X}_{:,j}\|_2^2 \tag{4}$$

  Note that we get samples from calibration dataset D_1 and $\|\mathbf{X}_{:,j}\|_2^2 = \sum_{k=1}^{N\times L} X^2_{k,j}$

- **Global Salience (S^{global}):** Captures the mean absolute gradient over a dataset D_2, which is typically smaller than D_1 and task-specific:

$$S^{\text{global}}_{ij} = \frac{1}{|D_2|} \sum_{D_2} |g_{ij}|, \quad \text{with} \quad g_{ij} = \frac{\partial L}{\partial W_{ij}} \tag{5}$$

The final element-wise salience is defined as the product $S_{ij} = S^{\text{local}}_{ij} \cdot S^{\text{global}}_{ij}$. Empirically, these salient weights often exhibit strong spatial locality, clustering consecutively along specific rows of the weight matrix. Consequently, we group them into contiguous "salient vectors", which naturally aligns with the block-wise partitioning required for vector quantization. Specifically, each row is partitioned into non-overlapping blocks of dimension d. To guide quantization, we compute the vector-wise salience $S^v_{i,b}$ for the b-th block in row i by summing the element-wise salience factors within its d entries (see Line 6). This metric subsequently drives weighted K-means clustering and the adaptive selection of salient blocks.

3.2 Sparse Compensation Matrix

To compensate for quantization errors, we introduce a sparse matrix consisting of three components:

1. **Outliers ($W_{outlier}$):** Based on the L_2 norm of each weight sub-vector, we designate those deviating significantly from the spherical distribution as outliers and preserve them.
2. **Salient Vectors ($W_{salient}$):** Weight sub-vectors with the highest salience factor S^v are maintained to protect critical model information.
3. **Residual Compensation (R):** We implement a salience-weighted ranking strategy on the quantization residuals $W^v - Q$. For the each sub-vector, we select the top fraction of d-dimensional residuals according to the metric $M_{i,b}$:

$$M_{i,b} = (S^v_{i,b})^{\frac{\alpha}{d}} \cdot \|W^v_{i,b} - Q_{i,b}\|_2^2 \tag{6}$$

where $S^v_{i,b}$ is the aggregate salience ,and $Q \in \mathbb{R}^{m\times(n/d)}$ denotes the matrix of quantized sub-vectors reconstructed via K-means decoding. The hyperparameter α acts as a coefficient that balances the contribution of salience against

the raw quantization residual magnitude. In our framework, we set $\alpha = 0.25$. A detailed ablation study on α is provided in Appendix H.

Table 1: Sparsity analysis of weight metrics. We observe that both the L_2 norm and our salience factors exhibit significant sparsity in sparse vector selection. The entries represent the percentage of the total value range covered by a given percentile. For instance, the 99.99% percentile of S covers only 28.5% of its maximum-to-minimum range, indicating a highly heavy-tailed distribution. These results are averaged across the `q_proj` weight matrices of all 32 layers in the LLaMA-2-7B model.

Metric	**90%**	**99%**	**99.9%**	**99.99%**
L_2 Norm	23.4%	34.7%	45.5%	56.8%
S^{global}	17.5%	21.2%	27.5%	31.5%
S^{local}	21.4%	27.8%	34.5%	45.8%
S	13.6%	15.7%	21.5%	28.5%

As illustrated in Table 1, the metrics defining $W_{outlier}$ and $W_{salient}$ exhibit pronounced sparsity, suggesting that preserving only approximately 1% of the sub-vectors in the compensation matrix is sufficient for effective performance restoration. Notably, while specific sparsity levels may be selected individually for each component matrix, the aggregate sparsity ratio r_c is significantly lower than their arithmetic sum. This efficiency stems from the substantial overlap among vectors selected across three different criteria, ensuring a highly compact representation.

Additionally, we examine the distribution patterns of $\bar{W}$ and S, as illustrated in Figure 3. We observe elements with large magnitudes tend to concentrate within the same row, where they frequently form contiguous, block-like clusters. This distributional characteristic provides a strong rationale for $W_{outlier}$ and $W_{salient}$ as vector-wise sparse matrices, effectively preserving the critical weights that are essential for maintaining model performance.

3.3 Enhanced K-Means

Salience-Weighted K-Means Our vector quantization framework adopts a *salience-weighted K-means* approach, where the salience factor S^v serves as the weighting factor for each weight sub-vector. To optimize centroid initialization and avoid suboptimal local minima, we implement an salience-weighted variant of the K-means++ algorithm (Arthur and Vassilvitskii, 2007). This ensures that weights with higher salience have a more significant influence on both the initial placement of centroids and their subsequent refinement. In practice, the clustering is performed on $W^v - W_{\text{outlier}} - W_{\text{salient}}$. Since some weight sub-vectors are already extracted and stored in W_{outlier} and W_{salient}, their corresponding positions become zero vectors. To ensure these vectors are perfectly reconstructed, we explicitly constrain the codebook to include a zero vector as a fixed centroid in Line 11. Detailed steps are provided in E.

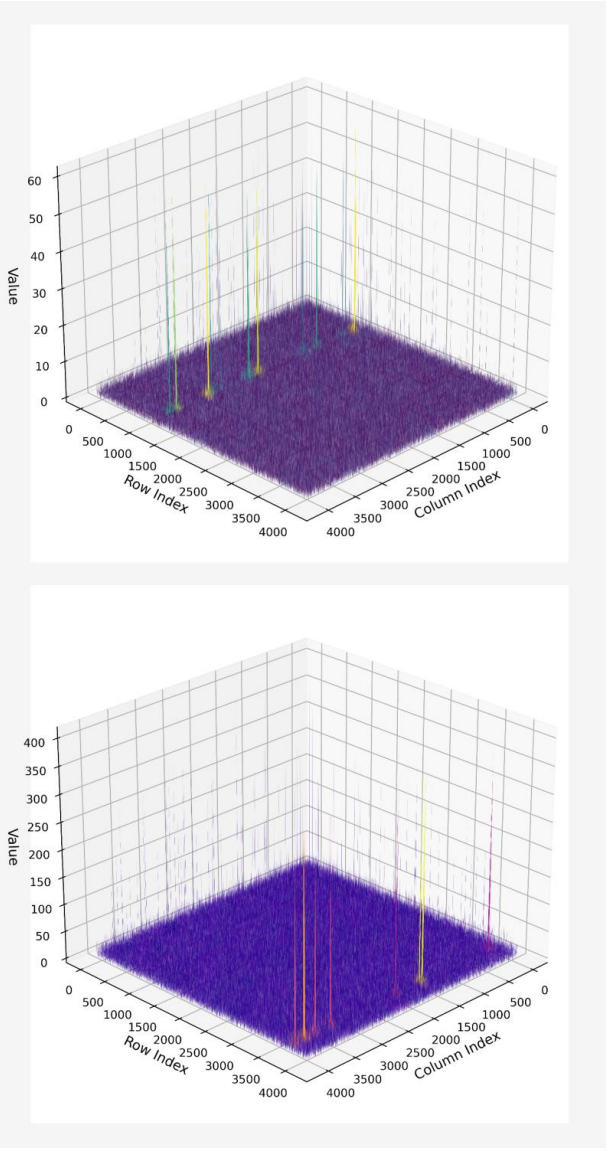

Figure 3: Visualization of the absolute values of $\bar{W}$(top) and S(bottom) for LLaMA-2-7B (`k_proj`, Layer 21).

Symmetry-Constrained K-Means As observed in Section 3.1, weight sub-vectors exhibit pronounced axial symmetry following normalization. Leveraging this property, we propose a

symmetry-constrained K-means algorithm to alleviate the memory overhead associated with large codebooks. Given a set of n weight sub-vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathbb{R}^d$, we first identify a subset of dimensions $D \subset \{1, \ldots, d\}$ with cardinality $|D| = \bar{d}$ that demonstrate the dimension with strongest symmetry. Specifically, for the i-th dimension, we compare the distribution of the original values with the distribution of their negated counterparts using the Kolmogorov-Smirnov(KS) test where a lower KS statistic indicates more pronounced symmetry. Further details are provided in Appendix D. For each sub-vector $\mathbf{v}_i$, we transform the values in dimensions D into their absolute values, yielding the transformed vectors $\{\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_n\} \subset \mathbb{R}^d$, while concurrently preserving the original signs as a bitmask $\mathbf{bitmask}_i \in \{+1, -1\}^{\bar{d}}$. By performing K-means clustering on these absolute-valued vectors $\{\tilde{\mathbf{v}}_i\}$, the required number of centroids is reduced from k to $k/2^{\bar{d}}$ without compromising the effective representation capacity. To implement this efficiently, each entry in the resulting compressed matrix $I \in \mathbb{R}^{m \times (n/d)}$ is partitioned: the first $\bar{d}$ bits are allocated to store the sign $\mathbf{bitmask}_i$, and the subsequent $\log_2(k/2^{\bar{d}})$ bits are used for the reduced codebook index $\mathbf{idx}_i$. Conceptually, our method can be viewed as imposing an explicit axial symmetry constraint on the codebook.

Codebook Quantization To further compress the memory footprint of codebook, we apply INT8 quantization to the codebook entries. Specifically, for a learned codebook $\mathcal{B} \in \mathbb{R}^{k \times d}$, we employ a symmetric per-row quantization scheme. Each centroid vector $\mathcal{B}_{i,:}$ (the i-th row of the codebook) is assigned an individual scale factor $s_i \in \mathbb{R}$, allowing for a more fine-grained representation of the distribution of each centroid. The quantization process is defined as $\hat{\mathcal{B}}_{i,:} = \text{round}(\frac{\mathcal{B}_{i,:}}{s_i})$ where $s_i = \frac{\max(|\mathcal{B}_{i,:}|)}{2^7 - 1}$. $\hat{\mathcal{B}} \in \mathbb{Z}^{k \times d}$ denotes the INT8 quantized codebook. During inference, the centroids are dequantized on-the-fly to their floating-point equivalents before being used for matrix multiplication. As demonstrated in Appendix G , quantizing the codebook $\mathbf{B}$ to 8 bits introduces negligible precision loss while further reducing the overall model size.

3.4 Fine-tuning

Inter-layer dependencies are crucial for restoring the performance of quantized models (Du et al., 2024; Liu et al., 2024b; Shao et al., 2023), particularly under extreme quantization configurations. To capture these interactions, we introduce an end-to-end fine-tuning stage following the enhanced K-means clustering step. To maintain high parameter efficiency during fine-tuning, we select two learnable components: the scaling vectors and the non-zero elements of the sparse compensation matrix. We perform fine-tuning via knowledge distillation, treating the full-precision model as the teacher and employing the Confidence-Aware KL divergence from BitDistiller (Du et al., 2024) as the objective function. In our 2.04 BPW configuration, these trainable parameters collectively account for only approximately 2–3% of the total model parameters while reducing perplexity on WikiText-2 and C4 by 0.7 and 0.63, respectively, achieving a superior balance between performance restoration and parameter efficiency. More details are provided in Appendix L.

3.5 Implementation and Hardware Optimization

To achieve efficient inference, we implement specialized LUT kernels for 2.04 and 3.20 BPW configurations. These kernels perform vector-wise dequantization by extracting sign bitmasks and codebook indices through bitwise shifts. For dimensions flagged by the symmetry constraint, the kernel conditionally restores the sign of retrieved entries based on the bitmasks. Following dequantization, all subsequent GEMV operations are executed in FP16 precision.

The sparse compensation matrix C exhibits a structured sparsity pattern in which d non-zero elements are arranged consecutively along each row. We propose the **Vector Sparse Row (VSR)** format, illustrated in Figure 4, to exploit this structure. Compared to the standard CSR (Pinar and Heath, 1999) format, which requires a column index for every individual non-zero element, VSR stores only a single index per sub-vector. This format reduces the metadata overhead for sparse matrix storage, thereby decreasing inference latency in memory-bound scenarios. Furthermore, VSR circumvents the limitations of NVIDIA cuSPARSE BSR (NVIDIA Corporation, 2024) kernels, which are primarily optimized for square blocks rather than the vector-wise sparsity patterns prevalent in quantized LLMs. We developed a custom CUDA kernel for VSR-based Sparse Matrix-Vector Multiplication (SpMV). Crucially, the contiguous arrangement of non-zero elements in VSR is ex-

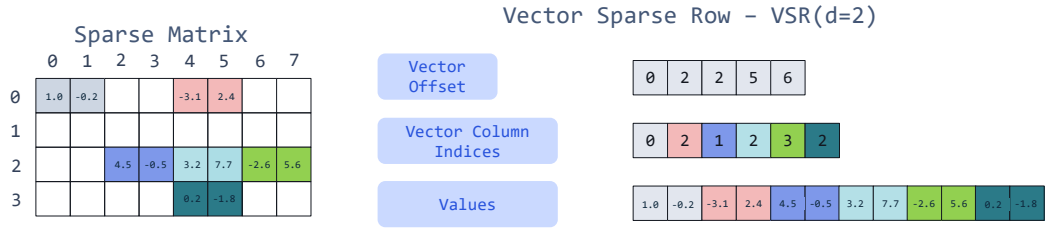

Figure 4: The example shows a 4×8 matrix represented in VSR format when $d = 2$.

plicitly leveraged to enable fully coalesced memory access within our custom kernel. This alignment allows the GPU memory subsystem to coalesce these requests into a minimal number of wide memory transactions, drastically reducing DRAM access overhead and L2 cache thrashing. To mitigate load imbalance caused by the uneven distribution of non-zero sub-vectors across rows, we assign an equal number of sub-vectors to each thread, following the workload partitioning strategy adopted in SqueezeLLM (Kim et al., 2024). To support the fine-tuning stage, we implemented a custom Pytorch autograd operator for the VSR format. During the backward pass, the kernel updates gradients exclusively for pre-selected non-zero sub-vectors, thereby maintaining a static sparsity pattern to ensure hardware compatibility.

4 Experiments

In this section, we evaluate the performance of our proposed method on modern LLMs. To facilitate a comprehensive comparison with established baselines, we focus primarily on the LLaMA-2 (Touvron et al., 2023) model family. Results for other LLMs are provided in Appendix J.

Table 2: Decoding throughput (tokens/s) comparison with a batch size of 1 on a single NVIDIA A100 GPU. BPW values are reported for the 7B model.

METHOD	BPW	L2-7B	L2-13B	L2-70B
FP16	16.00	103.7	55.2	OOM
SqueezeLLM	3.24	154.2	97.0	13.5
SCVQ	3.20	188.5	116.4	16.8
AQLM	2.02	165.9	101.3	17.2
VPTQ	2.02	156.3	105.5	18.4
SCVQ	2.04	235.4	135.6	22.7

4.1 Compression Quality

To evaluate the compression quality of the quantized models, we report perplexity on the WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2020) validation sets. Additionally, we evaluate zero-shot accuracy across several widely-used LLM benchmarks, including WinoGrande (Sakaguchi et al., 2020), PiQA (Tata and Patel, 2003), ARC-Easy, and ARC-Challenge (Clark et al., 2018).

We first report the results of SCVQ under the 2-bit configuration; please refer to Appendix B for detailed Bits-Per-Weight (BPW) calculations. We compare SCVQ against state-of-the-art baselines that also utilize 8-dimensional vector quantization and end-to-end optimization. As illustrated in Table 3, our method consistently outperforms other vector quantization approaches under this challenging 2-bit setting, demonstrating superior performance of the quantized model.

To demonstrate the effectiveness of our method even in the absence of fine-tuning, we evaluate its performance against various PTQ baselines that do not require additional gradient computations. The results for LLaMA-2-7B are illustrated in Table 15 in Appendix J. We prove flexible BPW scaling of SCVQ by adjusting several key hyperparameters in Table 11.

4.2 Inference Efficiency

To evaluate the practical computational efficiency of our quantization framework, we benchmark the decoding throughput of SCVQ on a single NVIDIA A100 GPU (80GB). We conduct inference experiments across the LLaMA-2 model family (7B, 13B, and 70B). To simulate real-world latency-sensitive scenarios, we fix the batch size to 1.

As illustrated in Table 2, SCVQ significantly outperforms both AQLM (Egiazarian et al., 2024) and SqueezeLLM (Kim et al., 2024) across comparable bitwidths. Notably, SCVQ achieves a throughput of 235.4 tokens/s for LLaMA-2-7B under 2-bit quantization, delivering a 2.3× speedup over the FP16 baseline. These performance gains are driven by two key optimizations: First, unlike SqueezeLLM's reliance on unstructured sparse matrices, SCVQ utilizes the structured VSR format which enables efficient coalesced memory access and eliminates the bank conflicts and branch divergences typical of irregular sparse indices, significantly enhancing SpMV kernel efficiency. Second, compared to AQLM, the acceleration stems from our symmetry-constrained K-means. By consolidating the code-

Table 3: Perplexity and zero-shot accuracy results for LLaMA-2 models under 2-bit quantization. All listed methods utilize 8-dimensional vector quantization and end-to-end fine-tuning. Evaluations are performed with a context length of 4096.

Model	Method	BPW	Wiki2 ↓	C4 ↓	WINO↑	PIQA↑	ARCE↑	ARCC↑
L2-7B	FP16	16.0	5.12	6.63	67.25	78.45	69.32	40.02
	AQLM	2.02	6.14	8.09	65.67	76.01	63.43	34.39
	QuIP#	2.02	6.19	8.16	64.96	75.41	64.96	35.15
	VPTQ	2.02	6.13	8.07	64.33	75.19	63.84	35.24
	SCVQ	2.04	**5.87**	**7.71**	**66.29**	**77.22**	**66.72**	**37.80**
L2-13B	FP16	16.0	4.57	6.05	69.61	78.73	73.27	45.56
	AQLM	1.97	5.33	7.19	68.67	76.82	69.99	40.36
	QuIP#	2.01	5.35	7.20	67.64	**77.26**	69.02	39.85
	VPTQ	2.02	5.32	7.15	66.85	**77.26**	**71.55**	40.02
	SCVQ	2.05	**5.03**	**6.74**	**68.90**	76.96	71.04	**42.18**
L2-70B	FP16	16.0	3.12	4.97	76.95	81.07	77.74	51.11
	AQLM	2.07	3.83	5.62	74.35	**80.90**	74.58	48.98
	QuIP#	2.01	3.91	5.71	74.66	79.54	77.06	47.61
	VPTQ	2.07	3.93	5.72	74.98	80.33	77.06	47.78
	SCVQ	2.04	**3.56**	**5.41**	**75.21**	80.33	**77.18**	**49.06**

Table 4: Ablation study of SCVQ on LLaMA-2-7B. We evaluate the contribution of each component by reporting changes in perplexity and zero-shot accuracy when a specific optimization is removed. Notably, we also report the impact of each component on the model's memory footprint, as reflected by the BPW. The results demonstrate that the full SCVQ achieves Pareto optimality in terms of both memory and performance.

Ablation Case	BPW	Wiki2 ↓	C4 ↓	Wino ↑	PiQA ↑	ARC-E ↑	ARC-C ↑
Full Pipeline (2.04 BPW)	2.04	5.78	7.71	66.29	77.22	66.72	37.80
w/o Symmetry Constraint	2.16	5.69	7.66	66.40	77.22	66.25	37.98
w/o Codebook Quantization	2.10	5.72	7.67	66.40	76.99	66.03	37.53
w/o Residual Matrix (R)	1.93	6.24	7.95	66.01	76.04	64.46	35.75
w/o Salient Matrix ($W_{salient}$)	1.89	6.31	8.07	66.01	76.12	64.37	35.30
Naive K-Means (Unweighted)	2.04	6.37	8.11	66.01	75.76	64.02	34.14
w/o Fine-tuning	2.04	6.57	8.34	65.84	75.04	63.81	31.19
w/o Outlier Matrix ($W_{outlier}$)	1.90	6.59	8.41	65.16	74.76	63.81	32.62
Full Pipeline (3.20 BPW)	3.20	5.27	6.81	67.14	78.10	68.34	39.95
w/o Residual Matrix (R)	3.07	5.31	6.89	67.14	77.04	68.21	39.50
w/o Salient Matrix ($W_{salient}$)	3.11	5.32	6.91	67.02	77.10	67.99	39.14
w/o Fine-tuning	3.20	5.35	6.93	67.02	77.04	68.12	39.14
w/o Outlier Matrix ($W_{outlier}$)	3.09	5.63	7.07	65.16	75.84	67.72	38.24
Naive K-Means (Unweighted)	3.20	5.87	7.24	65.04	75.78	67.42	38.16

book through axial symmetry, we reduce its memory footprint and alleviate cache pressure during dequantization lookups.

4.3 Ablation Analysis

We conduct an ablation study on the LLaMA-2-7B model to evaluate the contribution of each optimization component within our quantization pipeline. Table 4 presents the performance of our full proposed scheme alongside variants where a specific optimization is removed. Our analysis indicates that fine-tuning, salience-weighted K-means clustering, and the sparse outlier compensation are the most critical factors influencing model performance. Removing these components leads to the most significant degradation in both perplexity and zero-shot accuracy. We also investigate the impact of our codebook compression design and the VSR-based inference kernels on decoding throughput in Appendix K. More ablation analyses for SCVQ are in Appendices H, G and I.

5 Conclusion

We presented SCVQ, a novel post-training quantization approach that integrates sparse compensation with vector quantization for large language models. By combining salience-aware K-means clustering, symmetry-constrained codebook compression, and structured sparse compensation matrices, SCVQ achieves state-of-the-art performance at ultra-low bit-widths. Our custom VSR-based CUDA kernels enable efficient inference, delivering speedup on hardware while maintaining high model quality. Extensive experiments on the

LLaMA-2 family demonstrate that SCVQ consistently outperforms existing methods across perplexity and zero-shot benchmarks. This work highlights the potential of co-designing quantization algorithms with hardware-efficient sparse representations for practical LLM deployment.

Limitations

Despite promising results, SCVQ has limitations warranting further study. Our sparse sub-vector selection relies on a heuristic ranking metric, and the sensitivity hyperparameter α was set via linear search, underscoring the need for theoretical grounding. While our compensation framework is orthogonal to various VQ schemes, this work focuses solely on K-means clustering; future iterations could integrate lattice-based or other advanced VQ methods for higher fidelity and efficiency. Additionally, due to the curse of dimensionality, we currently restrict experiments to $d = 8$. Subsequent work will explore scaling high-dimensional VQ to LLMs through alternative quantization paradigms.

Acknowledgements

We gratefully acknowledge Denghao Li, Jiuyuan Lu, Yuwei Ren, and Yin Huang at Qualcomm AI Research for their support in providing essential computational resources and in fostering a collaborative research environment. We are especially thankful to Denghao Li for his insightful discussions and constructive feedback throughout this work.

References

David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035. Society for Industrial and Applied Mathematics.

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer, New York.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36:4396–4429.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training data. https://github.com/togethercomputer/RedPajama-Data. Accessed: 2026-01-02.

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

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *Preprint*, arXiv:2306.03078.

Dayou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. 2024. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation. *arXiv preprint arXiv:2402.10631*.

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

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

Georgi Gerganov and contributors. 2023. llama.cpp: Inference of llama model in c/c++. https://github.com/ggerganov/llama.cpp. Accessed: 2025-12-27.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pages 291–326. Chapman and Hall/CRC.

RobertM. Gray. 1984. Vector quantization. *IEEE Assp Magazine,IEEE Assp Magazine*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Wengang Guo, Kaiyan Lin, and Wei Ye. 2021. Deep embedded k-means clustering. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 686–694. IEEE.

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

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2024. Squeezellm: Dense-and-sparse quantization. In *International Conference on Machine Learning*, pages 23901–23923. PMLR.

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

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. 2024a. Vptq: Extreme low-bit vector post-training quantization for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8181–8196.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024b. Llm-qat: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 467–484.

J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv: Computation and Language,arXiv: Computation and Language*.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International conference on machine learning*, pages 7197–7206. PMLR.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*.

NVIDIA Corporation. 2024. cuSPARSE Library. https://developer.nvidia.com/cusparse. Accessed: 2024-05-20.

A. Pinar and M.T. Heath. 1999. Improving performance of sparse matrix-vector multiplication. In *SC '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, pages 30–30.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, page 8732–8740.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.

S. Tata and J.M. Patel. 2003. Piqa: an algebra for querying protein data sets. In *15th International Conference on Scientific and Statistical Database Management, 2003.*

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip #: Even better llm quantization with hadamard incoherence and lattice codebooks. In *International Conference on Machine Learning*, pages 48630–48656. PMLR.

Mart van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2025. Gptvq: The blessing of dimensionality for llm quantization. *Preprint*, arXiv:2402.15319.

A Theoretical Motivation for Bi-Normalization

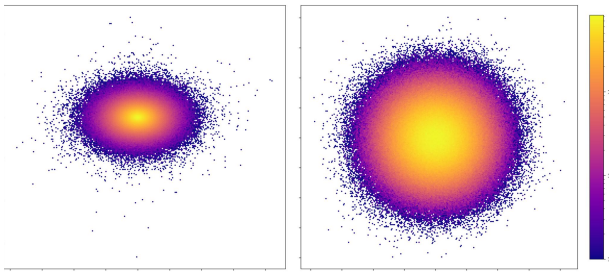

Figure 5: 2D PCA visualization of $d = 8$ sub-vector distributions for LLaMA-2-7B (`q_proj`, Layer 15). **Left:** without normalization; **Right:** after bi-normalization. The bi-normalization transforms the distribution from an anisotropic structure to a significantly more isotropic, spherical distribution.

K-means quantization performs optimally when the underlying data distribution exhibits spherical symmetry. This stems from its objective function, which minimizes the sum of squared Euclidean distances to cluster centroids. From a probabilistic perspective, this objective admits a well-known interpretation as the limiting case of Gaussian Mixture Models with isotropic, tied covariance matrices(Bishop, 2006; MacQueen, 1967). Under such conditions, the resulting Voronoi partitions form convex polyhedral regions that closely approximate spherical geometry, thereby minimizing quantization distortion.

In contrast, weight matrices in large language models typically exhibit highly anisotropic distributions, where probability mass concentrates along principal coordinate axes rather than radially (often manifesting as "ellipse-shaped"). This geometric mismatch between the data's intrinsic structure and the isotropic Euclidean metric employed by K-means induces distorted cluster boundaries and elevates reconstruction error. Consequently, preconditioning the weight distribution toward near-isotropy is essential for achieving high-fidelity vector quantization. A concrete example illustrating this phenomenon is provided in Figure 5.

To validate the practical impact of bi-normalization on LLM weights, we performed quantization on the q_proj weights from the 16th layer of LLaMA-2-7. We compared the SQNR of standard K-means against K-means preceded by bi-normalization across varying sub-vector dimensions and codebook sizes. As summarized in Table 5, bi-normalization yields consistent and significant SQNR improvements. We attribute these gains to two key mechanisms: (1) it enhances weight vector isotropy, aligning the distribution with the spherical geometry optimal for K-means; and (2) it promotes norm concentration, thereby compressing the dynamic range of sub-vectors and enabling more efficient codebook utilization.

Table 5: SQNR (dB) comparison of LLaMA-2-7B weights across different normalization schemes. **None** denotes standard K-means without normalization; **Row-norm** indicates row-wise scaling only; **Bi-norm** refers to the proposed bi-normalization.

Dim (d)	k	None	Row-norm	Bi-norm
2	4	3.47	4.12	**4.35**
	8	5.89	6.23	**6.89**
	16	8.42	8.44	**9.56**
	32	11.08	11.67	**12.29**
4	4	1.56	1.62	**1.88**
	8	2.73	3.08	**3.34**
	16	3.92	4.27	**4.63**
	32	5.17	5.33	**5.94**
8	4	0.71	0.78	**0.84**
	8	1.24	1.33	**1.48**
	16	1.89	2.04	**2.22**
	32	2.45	2.65	**2.85**
16	4	0.35	0.36	**0.40**
	8	0.58	0.62	**0.68**
	16	0.87	0.95	**1.02**
	32	1.20	1.28	**1.38**

B Bits per weight

To evaluate compression efficiency, we employ the *bits per weight* (BPW) metric (Gerganov and contributors, 2023), which quantifies the average bit consumption per parameter within our PTQ framework. The BPW is formulated as:

$$\text{BPW} = \underbrace{\frac{\log_2(k)}{d}}_{\text{Indices}} + \underbrace{\frac{k(d \cdot b_c + 16)}{g \cdot 2^{\bar{d}}}}_{\text{Codebook}} + \underbrace{16 \cdot r_c}_{\text{Sparse}} + \underbrace{\frac{32}{\sqrt{g}}}_{\text{Bi-norm}} \quad (7)$$

where k denotes the number of centroids, d represents the sub-vector dimension for quantization, and b_c is the bit-width of each codebook entry (e.g., $b_c = 8$ for INT8 quantized codebooks). The parameter g refers to the group size, and $\bar{d}$ denotes the number of dimensions subjected to symmetry constraints. Finally, r_c represents the aggregate sparsity ratio of the unified compensation matrix

$C = W_{\text{outlier}} + W_{\text{salient}} + R$. To simplify calculations, we do not account for the storage overhead associated with sparse matrix metadata in this formulation; a detailed analysis is provided in Appendix C.

In Eq. 7, the second term accounts for the storage overhead of both the quantized codebook and its associated per-row scale factors. Specifically, the scale factors are stored in FP16 format, contributing $\frac{16k}{g \cdot 2^d}$ bits per weight. The final term represents the storage cost of the bi-normalization scaling vectors. Assuming square sub-matrices of dimensions $\sqrt{g} \times \sqrt{g}$, the row and column scaling vectors together contribute $\frac{2 \times \sqrt{g} \times 16}{g} = \frac{32}{\sqrt{g}}$ bits per weight.

Table 6: Hyperparameter configurations and resulting BPW across LLaMA-2 model scales.

Hyperparams				**BPW / Sparsity** r_c		
d	k	b_c	$\bar{d}$	7B	13B	70B
8	2^{12}	8	1	2.04/2.2%	2.05/2.3%	2.04/2.2%
8	2^{12}	8	1	2.44/4.7%	2.42/4.6%	2.45/4.8%
8	2^{16}	8	4	2.70/2.2%	2.71/2.3%	2.70/2.2%
8	2^{16}	8	4	3.10/4.7%	3.08/4.6%	3.11/4.8%
8	2^{20}	8	8	3.20/2.2%	3.21/2.3%	3.20/2.2%
8	2^{20}	8	8	3.60/4.7%	3.58/4.6%	3.61/4.8%

Inspired by GPTVQ (van Baalen et al., 2025), we partition the weight matrices of linear layers into blocks, each assigned an independent codebook. Specifically, we adopt a block size of 1024×1024 (i.e., $g = 1024$). Unlike GPTVQ, which employs extremely small blocks to accommodate memory-constrained on-device LUT computation, our approach performs K-means clustering over a larger pool of weight sub-vectors, thereby more effectively capturing their underlying distribution patterns. Furthermore, in contrast to AQLM (Egiazarian et al., 2024), which assigns a fixed-size codebook regardless of matrix dimensions, our method adaptively allocates more codebooks to larger weight matrices. This size-proportional allocation ensures superior quantization reconstruction fidelity.

C VSR vs. CSR

Consider an original weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$. Let r denote the sparsity ratio, and d represent the grouping dimension employed by VSR. For a fair comparison, we assume that all non-zero values, row offsets, and column indices are uniformly stored using 16-bit precision (FP16). The total storage footprint (in bits) required by the CSR and VSR formats is given by:

$$\underbrace{16\,\lceil mnr \rceil}_{\text{Values}} + \underbrace{16(m+1)}_{\text{Offsets}} + \underbrace{16\,\lceil mnr \rceil}_{\text{Column Indices}} \tag{8}$$

$$\underbrace{16\,\lceil mnr \rceil}_{\text{Values}} + \underbrace{16(m+1)}_{\text{Offsets}} + \underbrace{\frac{16\,\lceil mnr \rceil}{d}}_{\text{Column Indices}} \tag{9}$$

As evident from Eqs. (8) and (9), VSR achieves a d-fold reduction in the storage cost of column indices compared to CSR, while maintaining identical overhead for values and row offsets. This compression stems from VSR's strategy of grouping d consecutive non-zero elements into a single index entry, thereby significantly improving memory efficiency without altering the underlying sparse structure.

Table 7: KS statistics across eight dimensions for linear layers in the 5-th transformer layer of LLaMA-2-7B. A lower score indicates stronger symmetry of the corresponding axis. Bold values represent the dimension with the highest symmetry in each layer.

Layer	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**
q_proj	.13	.21	.32	**.09**	.12	.17	.22	.25
k_proj	.23	.34	.22	.17	.15	**.11**	.12	.15
v_proj	.19	.26	**.06**	.16	.19	.07	.09	.14
o_proj	**.13**	.24	.31	.25	.16	.19	.24	.29
gate_proj	.11	.17	.17	.21	.12	.23	.31	**.09**
up_proj	**.04**	.08	.14	.19	.22	.31	.23	.26
down_proj	.28	.26	**.17**	.31	.25	.34	.19	.25

D Symmetry Evaluation via Kolmogorov-Smirnov Test

To implement the symmetry-constrained K-means described in Section 3.3, it is essential to identify the dimensions that exhibit the highest degree of axial symmetry. We employ a non-parametric approach based on the two-sample Kolmogorov-Smirnov (KS) test to quantify this symmetry across the weight space.

For a d-dimensional vector space, let V_j be a random variable representing the weight values along the j-th coordinate axis, where $j \in \{1, \ldots, d\}$. If the distribution along this dimension is perfectly symmetric about the origin, its cumulative distribution function (CDF) must satisfy $F_j(x) = P(V_j \leq x) = P(-V_j \leq x)$. In a discrete setting, this is equivalent to testing whether the sample set and its negation are drawn from the same underlying distribution.

Let $\mathcal{V}_j = \{v_{1,j}, v_{2,j}, \ldots, v_{n,j}\}$ be a set of n weight samples collected from the j-th dimension

of the sub-vectors $\{\mathbf{v}_i\}_{i=1}^n$. We define the Empirical Cumulative Distribution Function (ECDF) of the original samples as:

$$F_{n,j}(x) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}_{(-\infty,x]}(v_{i,j}) \tag{10}$$

where $\mathbb{I}(\cdot)$ is the indicator function. Similarly, let $\hat{F}_{n,j}(x)$ be the ECDF of the mirrored sample set $\mathcal{V}'_j = \{-v_{i,j} \mid v_{i,j} \in \mathcal{V}_j\}$:

$$\hat{F}_{n,j}(x) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}_{(-\infty,x]}(-v_{i,j}) \tag{11}$$

The KS statistic $D_{n,j}$ is defined as the supremum of the absolute differences between these two ECDFs:

$$D_{n,j} = \sup_x |F_{n,j}(x) - \hat{F}_{n,j}(x)| \tag{12}$$

A smaller $D_{n,j}$ value indicates that the distribution along the j-th dimension is more closely aligned with its reflection, implying stronger axial symmetry.

To select the $\bar{d}$ dimensions for the symmetry constraint, we rank all d dimensions by their respective KS statistics in ascending order:

$$D_{n,\pi(1)} \leq D_{n,\pi(2)} \leq \cdots \leq D_{n,\pi(d)} \tag{13}$$

where π is a permutation of the indices $\{1,\ldots,d\}$. The set of constrained dimensions D is then constructed from the top $\bar{d}$ indices:

$$D = \{\pi(1), \pi(2), \ldots, \pi(\bar{d})\} \tag{14}$$

This data-driven selection ensures that the mirror consolidation process targets the most intrinsically symmetric axes of the weight distribution, thereby minimizing the reconstruction error during symmetry-constrained quantization. Table 7 shows an example of our symmetry evaluation method when $d = 8$.

E Salience-Weighted K-Means Algorithm

Our quantization framework utilizes a weighted K-means algorithm to minimize the salience-weighted reconstruction error, ensuring that critical weights for model performance are reconstructed with higher fidelity. Given a set of n weight sub-vectors $\{\mathbf{v}_1,\ldots,\mathbf{v}_n\} \subset \mathbb{R}^d$ and their corresponding aggregate salience factors $\{S_1,\ldots,S_n\}$, we aim to find a codebook of k centroids $\mathcal{B} = \{\mathbf{c}_1,\ldots,\mathbf{c}_k\}$ that minimize the following objective:

$$J = \sum_{i=1}^{n} S_i \|\mathbf{v}_i - \mathbf{c}_{\mu(i)}\|^2 \tag{15}$$

where $\mu(i) \in \{1,\ldots,k\}$ is the index of the centroid nearest to $\mathbf{v}_i$.

E.1 Salience-Weighted K-Means++ Initialization

To avoid suboptimal local minima and ensure salient regions are well-represented, we modify the K-Means++ initialization. The first centroid $\mathbf{c}_1$ is sampled randomly. Subsequent centroids $\mathbf{c}_j$ are sampled from the remaining sub-vectors with a probability biased by both distance and salience:

$$P(\mathbf{c}_j = \mathbf{v}_i) = \frac{S_i \cdot D(\mathbf{v}_i)^2}{\sum_{m=1}^{n} S_m \cdot D(\mathbf{v}_m)^2} \tag{16}$$

where $D(\mathbf{w}_i)$ is the shortest Euclidean distance from $\mathbf{v}_i$ to the centroids already selected.

E.2 Iterative Refinement

The algorithm alternates between the following two steps until the centroids stabilize:

1. **Weighted Assignment**: Each sub-vector $\mathbf{v}_i$ is assigned to the nearest centroid $\mathbf{c}_j$ in the Euclidean space. Note that while the assignment is based on distance, the impact on the total loss J is scaled by S_i.

2. **Weighted Centroid Update**: Each centroid $\mathbf{c}_j$ is updated to be the salience-weighted mean of the sub-vectors assigned to it:

$$\mathbf{c}_j = \frac{\sum_{i\in\mathcal{A}_j} S_i\mathbf{v}_i}{\sum_{i\in\mathcal{A}_j} S_i} \tag{17}$$

where $\mathcal{A}_j$ is the set of indices of sub-vectors assigned to centroid $\mathbf{c}_j$.

F Experimental Configurations

All experiments were conducted on NVIDIA A100 GPUs, with the number of GPUs ranging from 1 to 8 depending on the model scale.

To compute the local salience factor S^{local}, we utilized the RedPajama-Data-1T (Computer, 2023) dataset with a batch size of 4 and a context length of 4096. The calculation was performed over 1536 batches, totaling 6144 samples. For the global

Table 8: Impact of codebook quantization precision (b_c) on LLaMA-2-7B. All experiments use $d = 8, k = 2^{12}, \bar{d} = 1$, and $r_c = 2.23\%$.

Scheme	b_c	BPW	Wiki2	C4
Symmetric	4	1.98	6.93	9.01
	8	2.04	6.57	8.34
	16	2.17	6.13	8.01
Asymmetric	4	2.01	6.89	8.87
	8	2.08	6.32	8.23
	16	2.20	6.03	7.98

salience factor S^{global}, we used a smaller calibration set of 128 samples from the C4 (Raffel et al., 2020) dataset with a context length of 512. This stage followed the default training configurations provided by the Hugging Face accelerate (Gugger et al., 2022) library, employing the AdamW optimizer with a learning rate of 5×10^{-5}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$.

During the fine-tuning phase, we performed a grid search to determine the optimal learning rates for the sparse compensation matrix and the scaling vectors, which were set to 1×10^{-3} and 5×10^{-3}, respectively. We similarly utilized the AdamW optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$) and trained on 1024 samples from the WikiText-2 (Merity et al., 2016) dataset with a context length of 1024. Furthermore, the coefficient for the Confidence-Aware KL Divergence (Du et al., 2024) was estimated over 10 steps.

G Ablation Study on Codebook Quantization

To evaluate the impact of codebook quantization on both the memory footprint and the performance of the model, we conducted an ablation study comparing different BPW for codebook entry storage (b_c): FP16 , INT8, and INT4. This analysis was performed on the Llama-2-7B model without the fine-tuning stage described in Section 3.4, allowing us to isolate the raw quantization effect.

As shown in Table 8, 8-bit quantization provides a superior trade-off between the bit-rate and perplexity. We also compared symmetric and asymmetric quantization schemes. While asymmetric quantization achieves slightly higher reconstruction accuracy, it incurs an additional BPW overhead due to the requirement of storing zero-points for each codebook entry. Consequently, we selected 8-bit symmetric quantization for our final implementation as discussed in Section 3.3.

Table 9: Impact of sparsity ratios on model performance. r_{outlier}, r_{salient}, and r_{residual} denote vector-wise sparsity ratios for individual components, while r_c is the aggregate ratio after merging.

Category	Size	r_c	r_o	r_s	r_r	Wiki2↓	C4↓
Baseline	7B	0.00	0.0	0.0	0.0	7.58	9.44
Single	7B	1.00	1.0	0.0	0.0	6.63	8.51
	7B	1.00	0.0	1.0	0.0	6.77	8.68
	7B	1.00	0.0	0.0	1.0	7.02	9.11
Joint	7B	0.03	.01	.01	.01	7.33	9.36
	7B	0.12	.05	.05	.05	7.23	9.18
	7B	0.23	.10	.10	.10	6.97	8.93
Full(1%)	7B	2.23	1.0	1.0	1.0	6.57	8.34
	13B	2.32	1.0	1.0	1.0	–	–
	70B	2.21	1.0	1.0	1.0	–	–
Full(2%)	7B	4.73	2.0	2.0	2.0	5.65	7.55
	13B	4.62	2.0	2.0	2.0	–	–
	70B	4.75	2.0	2.0	2.0	–	–

H Ablation Study on Hyperparameter α

We conducted an ablation study to investigate the sensitivity of the hyperparameter α in our residual compensation strategy. The experiments were performed using the LLaMA-2-7B model configured at approximately 2.08 BPW. We evaluated the model's perplexity on the WikiText-2 and C4 validation sets across various α values.

Table 10 exhibits a U-shaped trend peaking at $\alpha = 0.25$, which optimally balances salience and residual compensation:

- $\alpha > 0.25$: Over-prioritizes salience, under-compensating large quantization errors.
- $\alpha < 0.25$: Over-emphasizes residuals, neglecting structurally critical weights.

Table 10: Ablation of α on LLaMA-2-7B at approximately 2.08 BPW. $\alpha = 0.25$ consistently yields the lowest values.

Hyperparameter α	Wiki2	C4
1.00	7.24	10.33
0.75	6.27	8.34
0.50	5.82	7.92
0.25	**5.78**	**7.71**
0.20	5.84	8.09
0.10	6.14	8.13

Table 11: Perplexity and zero-shot accuracy (%) for LLaMA-2 models. All results use the SCVQ pipeline.

Model	BPW	PPL ↓		Zero-shot Acc ↑			
		Wiki2	C4	WINO	PIQA	ARC-E	ARC-C
L2-7B	2.04	5.78	7.71	66.29	77.22	66.72	37.80
	2.44	5.65	7.55	65.72	77.79	66.89	38.23
	2.70	5.52	7.17	66.52	77.90	67.99	38.52
	3.10	5.31	6.86	66.86	78.02	68.24	39.41
	3.20	5.27	6.81	67.14	78.10	68.34	39.95
	3.60	5.21	6.75	67.25	78.22	68.95	40.03
L2-13B	2.05	5.03	6.74	68.90	76.96	71.04	42.18
	2.42	4.88	6.51	69.07	77.93	71.78	43.34
	2.71	4.81	6.32	69.52	78.02	72.04	43.61
	3.08	4.73	6.25	69.52	78.62	72.91	44.24
	3.21	4.67	6.17	69.85	78.82	73.39	43.97
	3.58	4.65	6.12	69.97	78.96	73.61	44.41
L2-70B	2.04	3.56	5.41	75.21	80.33	77.18	49.06
	2.45	3.46	5.34	76.14	80.65	77.40	48.70
	2.70	3.32	5.28	76.37	80.90	77.53	49.24
	3.11	3.28	5.11	76.66	80.96	77.74	50.04
	3.20	3.23	5.09	76.71	81.19	77.61	50.22
	3.61	3.19	5.07	76.88	81.28	77.92	50.49

I Ablation Study on Sparse Compensation Ratios

To investigate the impact of various sparsity ratios and the specific contribution of each component within the sparse compensation matrix to model performance restoration, we conducted a series of ablation experiments. Using the baseline configuration of LLaMA-2-7B, we varied the sparsity levels for r_{outlier}, r_{salient}, and r_{residual}. All perplexity results in Table 9 were evaluated in a post-training setting without the fine-tuning stage to isolate the raw impact of the sparse components.

As shown in the table, the outlier matrix W_{outlier} provides the most significant gain among single components, followed by the salient weight matrix W_{salient}. Notably, when multiple components are integrated, the aggregate sparsity r_c is consistently lower than the sum of individual ratios. This confirms the existence of substantial overlap between different salience criteria, allowing for a more compact and efficient sparse representation.

J Additional Result

Beyond the LLaMA-2 family discussed in the main text, Table 12 summarizes the performance of various quantization methods on the LLaMA-3.1-8B and LLaMA-3.2-3B models. Specifically, for GPTVQ (van Baalen et al., 2025), we employ 4-dimensional vector quantization. For AQLM (Egiazarian et al., 2024) and QuIP# (Tseng et al., 2024), we incorporate fine-tuning and strictly adhere to the hyperparameter configurations reported in their respective works. Additionally, we validate our SCVQ method on the Qwen3 model, as illustrated in Table 16.

Table 12: Results on LLaMA-3.1-8B and LLaMA-3.2-3B. We report perplexity and zero-shot accuracy.

		BPW	PPL ↓		Zero-shot Acc ↑	
			Wiki2	C4	ARC-E	ARC-C
L3.1-8B	BF16	16.00	6.50	8.02	78.03	51.37
	SCVQ	2.05	7.64	9.09	74.94	48.21
	GPTVQ	2.13	9.41	11.13	71.52	46.08
	AQLM	2.29	7.82	9.20	74.10	47.61
	SCVQ	2.42	6.81	8.32	75.57	49.06
	GPTVQ	3.13	7.02	8.76	73.84	47.61
L3.2-3B	BF16	16.00	9.62	10.75	71.42	44.45
	SCVQ	2.05	13.35	13.44	67.08	38.57
	AQLM	2.02	14.31	15.56	66.10	38.91
	QuIP#	2.02	15.03	15.71	66.28	37.67
	QuIP#	3.02	10.48	11.19	68.19	41.38
	SCVQ	3.20	10.46	11.15	69.70	41.89

BPW Scaling. We achieve flexible BPW scaling by adjusting several key hyperparameters, including the number of centroids k in K-means clustering, the number of axes $\bar{d}$ for symmetry constraints, and the average sparsity r_c of the sparse compensation matrix. As demonstrated in Table 11, the performance of the quantized models exhibits a clear and consistent scaling behavior as the BPW increases. The specific hyperparameter configurations used for these experiments are detailed in Table 6.

K Ablation Study for Decoding Throughput

The ablation results are summarized in Table 13. We observe that employing a naive CSR-based SpMV kernel results in a noticeable throughput degradation compared to our proposed VSR kernel. Since weight-only quantized LLMs are primar-

Table 13: Ablation study of decoding throughput (tokens/s) on LLaMA-2-7B using a single NVIDIA A100 GPU (Batch Size 1).

METHOD	BPW	b_c	$\bar{d}$	TOK/S
FP16 Baseline	16.00	16	-	103.7
SCVQ (Ours)	**3.20**	8	8	**188.5**
SCVQ (naive CSR)	3.20	8	8	161.5
SCVQ (FP16 codebook)	3.45	16	8	124.8
SCVQ (less symmetry)	3.48	8	7	106.3
SCVQ (Ours)	**2.04**	8	1	**235.4**
SCVQ (naive CSR)	2.04	8	1	213.5
SCVQ (FP16 codebook)	2.10	16	1	185.7
SCVQ (less symmetry)	2.16	8	0	134.5

ily constrained by the "memory wall" (Kim et al., 2024), this performance gap stems from the fact that the CSR format requires significantly more storage for column indices. In contrast, VSR minimizes this metadata overhead, reducing the total volume of data transferred from DRAM and alleviating memory bandwidth pressure. Furthermore, utilizing an FP16 codebook or relaxing the axial symmetry constraint doubles the codebook size. This expansion increases loading latency. Specifically, in the case of reduced symmetry, the doubling of codebook entries causes the LUT-based dequantization process to become the primary computational bottleneck rather than SpMV process.

Table 14: Ablation of trainable modules on 2.04 BPW LLaMA-2-7B. PPL is reported on WikiText-2 and C4.(SV: Scaling Vectors, SM: Sparse Matrix, CB: Codebook.)

Trainable Components	**Wiki2** ↓	**C4** ↓
Baseline	6.57	8.34
CB	6.34	8.05
SV	6.39	8.12
SM	6.23	7.97
SV + SM	**5.87**	**7.71**
SV + CB	6.04	7.92
CB + SM	6.55	8.17
SV + SM + CB	6.27	8.03

L Ablation Study for Finetuning

We evaluated three candidate trainable components—scaling vectors, codebook centroids, and the non-zero elements of the sparse compensation matrix—along with their various combinations for fine-tuning the quantized model. As summarized in Table 14, our ablation study reveals that jointly optimizing the scaling vectors and the sparse compensation matrix yields the most significant gains, achieving the lowest perplexity on WikiText-2 and C4. Notably, we observed a manifest optimization conflict when simultaneously updating the codebook and the sparse matrix, which hindered performance recovery. Consequently, we exclude the codebook from the final fine-tuning stage to ensure optimal and robust results.

Table 15: Performance of LLaMA-2-7B without fine-tuning. We report perplexity and zero-shot accuracy (%).

			PPL ↓		Zero-shot Acc ↑			
Bit	**Method**	**BPW**	**Wiki2**	**C4**	**WINO**	**PIQA**	**ARC-E**	**ARC-C**
2	AQLM	2.02	6.59	8.54	65.67	74.76	63.68	32.76
	QuIP#	2.02	8.22	11.01	62.43	71.38	55.56	28.84
	SCVQ	2.04	**6.57**	**8.34**	**65.84**	**75.04**	**63.81**	**31.19**
	GPTQ	2.13	36.83	51.34	—	—	—	—
	AQLM	2.29	6.29	8.11	65.67	74.92	66.50	34.90
	SCVQ	2.44	**5.65**	**7.55**	**66.12**	**75.21**	**66.59**	**35.12**
3	SpQR	2.98	6.20	8.20	63.54	74.81	67.42	37.71
	QuIP#	3.02	5.60	7.34	64.89	76.84	67.03	37.09
	AQLM	3.04	5.46	7.08	66.93	76.88	68.06	38.40
	SCVQ	3.10	**5.38**	**7.02**	**67.02**	**76.99**	**68.12**	**39.14**
	GPTQ	3.13	8.06	10.61	59.19	71.49	58.46	31.06
	AWQ	3.13	7.93	10.23	58.81	69.98	57.97	29.94
	SCVQ	3.20	5.35	6.93	67.02	77.04	68.12	39.14
	SCVQ	3.60	5.27	6.87	67.14	77.18	68.20	39.14

Table 16: Results on Qwen3 series models. We report perplexity and zero-shot accuracy. Best results are in **bold**.

			PPL ↓		Zero-shot Acc ↑			
Model	**Method**	**BPW**	**Wiki2**	**C4**	**WINO**	**PIQA**	**ARC-E**	**ARC-C**
4B	BF16	16.00	13.72	16.63	65.82	75.02	80.51	50.68
	AQLM	2.02	14.94	19.07	63.04	72.91	73.74	43.43
	QuIP#	2.02	14.84	18.91	63.61	73.12	73.86	43.69
	VPTQ	2.02	14.87	18.86	63.44	73.01	74.11	44.80
	SCVQ	2.05	**14.21**	**18.34**	**65.03**	**73.72**	**74.41**	**45.14**
8B	BF16	16.00	9.71	13.32	68.02	76.44	83.50	55.55
	AQLM	2.02	10.72	14.44	66.78	74.42	80.60	50.00
	QuIP#	2.03	10.75	14.23	66.32	74.27	80.47	49.66
	VPTQ	2.07	10.67	14.21	67.06	74.54	80.64	50.42
	SCVQ	2.04	**10.35**	**14.12**	**67.57**	**75.02**	**81.02**	**52.04**
14B	BF16	16.00	8.64	12.03	72.95	80.03	84.30	59.04
	AQLM	1.97	9.23	13.07	71.87	78.07	82.11	57.08
	QuIP#	2.02	**9.12**	13.13	70.63	78.61	81.90	57.25
	VPTQ	2.03	9.22	13.11	**72.16**	78.40	82.07	57.59
	SCVQ	2.05	9.17	**12.87**	72.09	**78.94**	**82.74**	**58.70**
32B	BF16	16.00	7.61	10.87	73.57	80.90	84.39	57.76
	AQLM	2.06	8.34	11.65	71.87	80.33	81.02	56.14
	QuIP#	2.02	8.41	11.73	71.64	79.70	81.27	54.95
	VPTQ	2.07	8.36	11.64	71.92	79.54	81.44	56.74
	SCVQ	2.04	**8.15**	**11.26**	**72.16**	**80.41**	**82.24**	**56.82**