Distribution-aware Low-bitwidth Quantization for Large Language Models

Bao Tan Duy Huynh, Takashi Tsunakawa, Masafumi Nishida
Shizuoka University
3 Chome-5-1 Johoku, Chuo Ward, Hamamatsu, Shizuoka 432-8011, Japan
duybaohuynhtan@gmail.com, {tuna, nishida}@inf.shizuoka.ac.jp

Abstract

The increasing scale and complexity of large language models (LLMs) present significant computational and memory challenges, limiting their widespread deployment. Post-training quantization (PTQ) has emerged as a key technique for mitigating these challenges without costly retraining. However, compressing models to ultra-low bitwidths (e.g., 2-3 bits) while maintaining accuracy remains a major challenge. In this study, we present a comprehensive PTQ framework that addresses this problem by compressing LLM weights through three core innovations: (1) a calibration process guided by Kullback-Leibler divergence minimization to preserve the original weight distribution, (2) a learnable codebook optimization mechanism employing noise substitution for vector quantization to enable robust gradient estimation, and (3) a layer-grouping strategy based on statistical distribution similarity to improve parameter efficiency. Experimental evaluations on large-scale models show that the proposed framework achieves competitive performance compared with state-of-the-art quantization techniques. Importantly, these results are obtained without any post-quantization fine-tuning, highlighting the efficiency and practical applicability of our approach for deploying highly compressed LLMs.

Keywords: Post-training quantization, large language model compression, low-bitwidth precision

1. Introduction

The proliferation of large language models (LLMs) has advanced numerous fields; however, their massive architectures—often comprising billions of parameters—pose substantial operational challenges. Training and inference require extensive computational and memory resources, resulting in high financial costs and considerable environmental impact. Consequently, developing efficient compression methods that reduce these costs while maintaining model performance has become a critical research focus.

Among the proposed approaches, post-training quantization (PTQ) reduces memory usage by converting high-precision floating-point representations (e.g., FP32 or FP16) of model components—such as weights, activations, and key-value caches—into lower-precision integer formats (e.g., INT8 or INT4). PTQ is applied after the model has been fully trained, eliminating the need for the original training dataset or an expensive retraining phase.

In this study, we propose a PTQ method designed to compress LLMs to ultra-low bitwidths, that is 2-3-bit representations. Our method is built upon the principles of product quantization (Jégou et al., 2011), an effective technique for high-dimensional vector compression, and is adapted to optimize the quantization of LLM weights. Our approach incorporates three core methods aimed at improving compression efficiency while preserving model accuracy.

First, to determine codeword vectors for the codebooks, our method leverages a small calibration dataset. This process aims to minimize the Kullback-Leibler divergence (KLD) between the distributions of the original and quantized weights.

Second, we introduce a learnable codebook optimization procedure in which quantization error is simulated by injecting noise into the input tensors. This approach is adapted from the noise substitution for vector quantization method (Vali and Backstrom, 2022).

Third, rather than quantizing layers independently, we implement a layer-grouping mechanism based on statistical distribution similarity. Layers within the LLM architecture that exhibit similar weight distributions are grouped and quantized collectively.

A key principle of the proposed method is to eliminate the need for post-quantization fine-tuning. While fine-tuning can partially recover performance, it increases computational complexity, risks overfitting, and may degrade the model's generalization. Accordingly, our approach emphasizes preserving performance directly through an intrinsically effective quantization process.

2. Related Work

2.1. Post-training Quantization Techniques

Early PTQ methods often employed uniform quantization schemes (Krishnamoorthi, 2018; Nagel

Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), pages 10057–10070
11–16 May 2026.

et al., 2021), using linear mapping based on a scale factor and a zero-point. However, such approaches are limited in handling outliers—rare values with disproportionately large effects on LLM performance—often resulting in severe accuracy degradation.

To overcome this limitation, advanced PTQ techniques have been developed. LLM.int8() (Dettmers et al., 2022) introduced a mixed-precision decomposition method to preserve outliers. GPTQ (Frantar et al., 2022) leverages second-order approximations to quantize weights to low bitwidths (3-4 bits) with minimal error. SmoothQuant (Xiao et al., 2023) enables simultaneous quantization of weights and activations by smoothing activation distributions. QLoRA (Dettmers et al., 2023) combines low-rank adaptation (LoRA) (Hu et al., 2022) with a 4-bit quantized base model for efficient fine-tuning on memory-constrained devices. The Activation-aware Weight Quantization (AWQ) method (Lin et al., 2023) applies activation-aware per-channel scaling to protect salient weights from large quantization errors. These developments illustrate the ongoing efforts to optimize PTQ for large-scale LLMs.

2.2. Research on Low-bitwidth Quantization

Recent work has investigated ultra-low-bitwidth quantization (e.g., 2-bit) to maximize compression efficiency. Notable contributions include QuIP (Chee et al., 2023), which provided a theoretical framework for 2-bit quantization using Incoherence Processing and Adaptive Rounding. Building on this approach, QuIP# (Tseng et al., 2024a) incorporated a Randomized Hadamard Transform for efficient weight mixing together with Vector Quantization (VQ) using an E_8 lattice. Concurrently, AQLM (Egiazarian et al., 2024) achieved state-of-the-art performance in the 2-3 bit regime through three key components: Additive Quantization, Input-Adaptive Quantization, and Joint Optimization, enabling efficient deployment on both GPUs and CPUs. More recently, QTIP (Tseng et al., 2024b) addressed the exponential complexity of VQ in high dimensions by introducing Trellis Coded Quantization (TCQ). Using the Viterbi algorithm, QTIP supports quantization in very high-dimensional spaces (> 100) with computational complexity that scales linearly with sequence length.

Despite these advances, leading methods such as QuIP#, AQLM, and QTIP still rely on post-quantization fine-tuning to recover accuracy. Although effective at mitigating quantization errors, this stage increases algorithmic complexity and computational cost, while also introducing the risk of overfitting to the fine-tuning dataset and potentially degrading the base model's generalization capability. To address these limitations, this work proposes a method that eliminates the need for post-quantization fine-tuning, focusing instead on a robust intrinsic quantization process capable of preserving high accuracy without additional training.

2.3. Approximate Nearest-neighbor Search and the Role of Product Quantization

Similar to AQLM (Egiazarian et al., 2024) and inspired by approximate nearest-neighbor search, our method leverages the principles of product quantization (PQ) (Jégou et al., 2011) and its variants. PQ decomposes an input vector into sub-vectors, quantizing each independently with a dedicated codebook. By combining the indices of each sub-vector, PQ represents a large vector space with minimal storage, achieving high compression ratios.

However, conventional PQ may be suboptimal for unstructured vectors, such as LLM weights, and often struggles to handle outliers. To address these limitations, several extensions have been proposed. Additive Vector Quantization (AVQ) (Babenko and Lempitsky, 2014) represents a vector as a sum of codevectors without constraints, while Residual Vector Quantization (RVQ) (Chen et al., 2010) employs a hierarchical scheme where each stage quantizes the residual from the previous stage, minimizing cumulative quantization error.

3. Proposed Method

In this section, we present a detailed account of our proposed quantization method, which focuses on extending and refining the Residual PQ technique to address the challenges of compressing LLMs to low bitwidths. Figure 1 provides an overview of the proposed framework for LLM compression.

3.1. Quantization with an Extended Residual Product Quantization Architecture

An analysis of existing vector quantization methods indicates that techniques with minimal structural constraints, such as AVQ (Vali and Backstrom, 2022), can achieve high quantization accuracy but often incur substantial computational costs. These costs arise both during codebook training and, more critically, during encoding, due to reliance on complex combinatorial search algorithms, such as heuristic beam search, to identify the optimal representation for each input vector. Our research focuses on quantization methods with

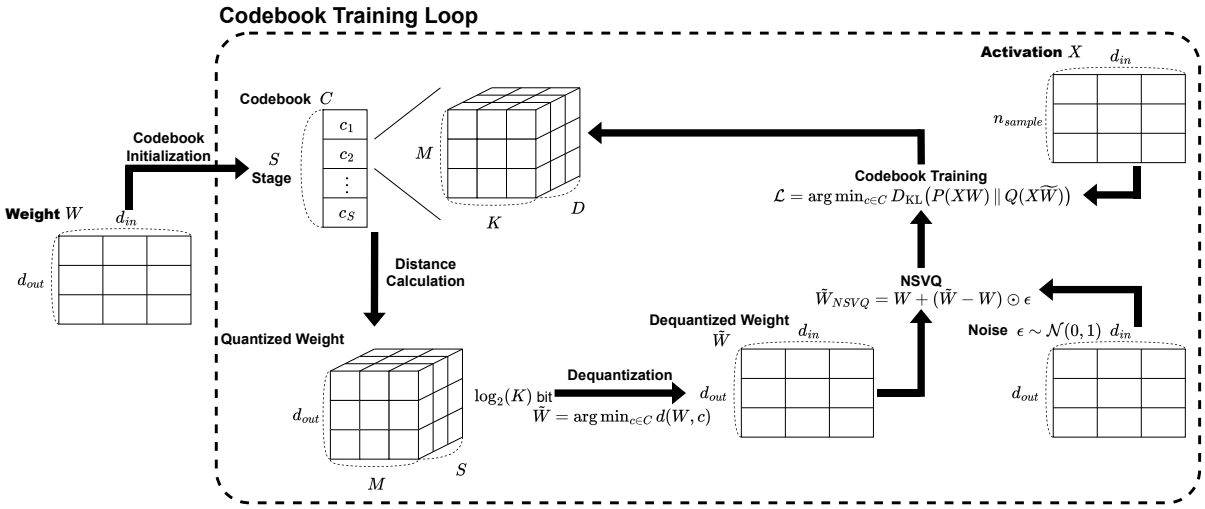

Figure 1: Overview of the proposed framework for LLM compression.

stronger structural constraints to achieve a superior quantization performance without substantially increasing computational complexity. This choice is motivated by the understanding that such constraints directly influence quantization performance and affect the computational complexity and search speed of the encoding process while being better suited to handle the unstructured nature of weight tensors in LLMs.

Based on this analysis, our research focuses on developing a method that builds upon and extends the hierarchical architecture of RVQ (Chen et al., 2010). A key advantage of this approach lies in its relatively low computational complexity stemming from the hierarchical organization of its codebooks. In RVQ, an input vector is quantized through multiple hierarchical stages. At each stage, the vector—or the residual from the previous stage—is represented by the nearest codevector from the current stage's codebook. The residual, defined as the difference between the input vector and the selected codevector, is then passed to the next stage for further quantization. This hierarchical process allows progressive refinement of the vector's quantized representation.

Within our proposed RVQ framework, we formalize this process for an input weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, through an iterative, multi-stage procedure.

We apply the PQ principle (Jégou et al., 2011) by partitioning the d_{in}-dimensional input feature space into M disjoint, non-overlapping subspaces, each with dimension $D = d_{\text{in}}/M$. This is equivalent to partitioning each row vector of the weight matrix W into M sub-vectors. The quantization process is then performed sequentially across S stages, following RVQ principles (Chen et al., 2010). The complete set of codebooks is structured within a four-dimensional tensor $C \in \mathbb{R}^{S \times M \times K \times D}$, where S is the total number of stages, M is the number of sub-codebooks, K is the number of codewords per sub-codebook, and D is the dimension of each codeword. The hierarchical architecture of RVQ is illustrated in Figure 2.

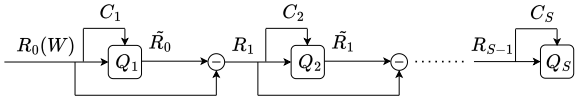

Figure 2: Hierarchical structure of Residual Vector Quantization.

The quantization process begins with the original weight matrix $R_0 = W$. At each stage $s \in \{1, \ldots, S\}$, the input residual matrix R_{s-1} is quantized to produce an approximation matrix $\tilde{R}_{s-1}$ using the quantizer $Q_s(R_{s-1}, C_s)$, where C_s is the codebook dedicated to stage s:

$$\begin{aligned} \tilde{R}_{s-1} &= Q_s(R_{s-1}; C_s) \\ &= \{\arg\min_{c \in C_s} d(r, c) \mid r \in R_{s-1}\} \end{aligned} \tag{1}$$

The quantizer Q_s first partitions each row vector of the input matrix into M sub-vectors and maps each sub-vector to its nearest codeword in the corresponding sub-codebook according to a specified distance metric $d(\cdot, \cdot)$. The residual matrix for the next stage R_s is then computed as:

$$R_s = R_{s-1} - Q_s(R_{s-1}, C_s) \tag{2}$$

establishing a recursive refinement process across stages.

After completing S stages, the final quantized weight matrix, denoted as $\tilde{W}$, is reconstructed by

summing all the approximation matrices generated at each stage:

$$\tilde{W} = \sum_{s=1}^{S} C_s(R_{s-1}) \tag{3}$$

The original weight matrix W is then replaced by a compressed representation comprising the codebooks and an index tensor. This index tensor has dimensions (S, d_{out}, M), where each element is an index referencing a specific codevector in the corresponding codebook. Each index requires $b = \log_2(K)$ bits of storage.

To address the nonuniform variance across input channels (i.e., the columns of matrix W), which can impair quantization performance, we integrate a per-channel scaling mechanism. Drawing on approaches from convolutional neural network compression (Krishnamoorthi, 2018), we introduce a learnable scaling matrix of size (S, d_{in}), enabling a distinct scaling factor for each input channel at every quantization stage. Before performing the nearest-neighbor search in the codebook, the columns of the residual matrix at each stage are normalized by these scaling factors. The factors are initialized based on the average magnitude (L^2 norm) of the corresponding channel in the stage's input matrix and are jointly optimized with the codebooks during quantizer training. The initial scaling factor for the j^{th} column at stage s is computed as:

$$scale_{s,j}^{(\text{init})} = \|(R_{s-1})_{:,j}\|_2 = \sqrt{\sum_{i=1}^{d_{\text{out}}} (R_{s-1})_{i,j}^2} \tag{4}$$

where $(R_{s-1})_{i,j}$ denotes the element in the i^{th} row and j^{th} column of R_{s-1}.

This approach ensures that channels with larger amplitudes are appropriately handled from the beginning of the optimization process.

3.2. Calibration Based on Kullback-Leibler Divergence Minimization

Our method builds on insights from prior work, such as AWQ (Lin et al., 2023). AWQ demonstrated that not all weights in an LLM contribute equally to performance; protecting a small subset of salient weights—often just 1% of the total—can substantially reduce quantization error. We extend this principle by proposing that, for complex quantization tasks, preserving the overall distributional shape of the weights is a more effective objective than focusing solely on individual values. By maintaining distributional similarity between the original and quantized weight sets, our approach minimizes overall error and safeguards values sensitive to precision loss, including outliers that are critical for model performance.

Thus far, we propose a calibration procedure to optimize quantization parameters, including learnable codebooks. This procedure uses a small calibration dataset and is guided by the objective of minimizing the KLD. The objective is to find the set of codebooks C that minimizes the KLD between the probability distribution of the layer with original weights, $P(W)$, and that of the layer with quantized weights, $Q(\tilde{W})$:

$$\arg\min_{c \in C} D_{\text{KL}}\big(P(XW;\theta) \,\|\, Q(X\widetilde{W};\theta)\big) \tag{5}$$

where X is the input data from the calibration set, θ represents the all pretrained model parameters (the backbone). Minimizing the KLD ensures that the quantization model captures the core statistical characteristics of the original weight distribution, thereby mitigating information distortion.

It is critical to note that during this optimization process, the parameters of the original model, θ, remain completely frozen. Although the optimization leverages backpropagation through the quantization mechanism (as will be detailed in Section 3.3), the optimizers and gradients are configured such that only the codebook C and other quantization parameters are updated. No gradient updates are applied to the model parameters θ. Therefore, our method is fundamentally a controlled minimization over an auxiliary parameter space, not a fine-tuning process that updates the entire model.

The KLD-based calibration procedure uses a small but representative dataset. Specifically, we employ 128 text segments of 2048 tokens each, randomly sampled from the RedPajama dataset (Weber et al., 2024) (see Appendix C.2 for details). A key aspect of our approach is the strict separation of data sources: the calibration process is entirely task-agnostic, ensuring that the model is never exposed to samples from evaluation or downstream task datasets. This preserves the integrity of the evaluation, allowing the post-quantization model's performance to be validly assessed in a zero-shot setting.

3.3. Learnable Codebook Optimization using Noise Substitution for Vector Quantization (NSVQ)

Conventional vector quantization maps each input vector x to vector $\tilde{x}$ in codebook C such that $\tilde{x}$ is the nearest vector to x according to a specified distance metric $d(\cdot,\cdot)$. This can be expressed mathematically as:

$$\tilde{x} = \arg\min_{c \in C} d(x, c) \tag{6}$$

where c is a codevector within codebook C. Machine learning algorithms, particularly those based on backpropagation and gradient descent, have demonstrated superior performance in solving complex optimization problems across diverse domains. Consequently, rather than relying solely on conventional clustering methods, such as k-means, for codebook construction, optimizing the codebook through learnable techniques can achieve higher quantization accuracy. A fundamental challenge is that the vector quantization function, as defined in (6), is discrete and non-differentiable at most points due to the `argmin` operator, preventing the direct application of gradient-based optimization methods.

To overcome this challenge, gradient estimation techniques can be applied to the quantization function, allowing gradients to propagate through the quantization nodes during backpropagation and update the codebook vectors. We posit that constructing and optimizing codebooks in a learnable manner—where codevectors can be flexibly fine-tuned—reduces the need for post-quantization full-model fine-tuning. This approach enables direct backpropagation onto individual codevectors, allowing precise adjustments to minimize quantization errors.

Several methods exist for estimating gradients of the quantization function. In this study, we employ the advanced NSVQ method (Vali and Backstrom, 2022), which replaces the quantization error, $e_q = \tilde{x} - x$, with the product of the original error and an artificial noise vector ϵ sampled from a standard normal distribution (zero mean, unit variance). This creates a continuous and differentiable approximation of the quantization function. By accounting for the statistical effects of quantization, NSVQ is hypothesized to provide more accurate gradient estimates for codebook vectors than the straight-through estimator, resulting in improved quantization performance and faster convergence during codebook optimization.

According to NSVQ, for an input vector x and its corresponding quantized vector $\tilde{x}$ from the codebook, the quantization error is simulated as follows:

$$\tilde{x}_{NSVQ} = x + (\tilde{x} - x) \odot \epsilon \tag{7}$$

where ϵ is a random noise vector sampled from a multivariate normal distribution with a zero mean and an identity covariance matrix and $\odot$ denotes the element-wise product. The function $\tilde{x}_{\text{NSVQ}}$ is differentiable with respect to the components of $\tilde{x}$ (and thus the codebook parameters), enabling codebook optimization using gradient-based methods.

3.4. Layer-grouping Strategy Based on Statistical Similarity

Analysis of LLM architectures shows that many linear layers, despite occupying different positions in the network, often perform similar functional roles and, notably, exhibit significant similarities in the statistical distribution of their weights. Concurrently, our analysis of RVQ in Section 3.1 indicates that increasing the number of stages, S, reduces quantization error. This suggests that a multi-stage quantization model has greater capacity to correct and mitigate cumulative errors at each stage.

Furthermore, prior work, such as Entropy-Based Weight Quantization (EWQ) (Behtash et al., 2025), has demonstrated that the statistical properties of weights constitute a meaningful and informative signal for guiding quantization. Specifically, EWQ utilized the entropy of the weight distribution as an indicator of a transformer layer/block's quantizability, enabling a mixed-precision quantization scheme by allocating more bits to higher-entropy (more critical) blocks.

This motivated our research to extend the use of weight distributions to optimize the quantization process. Instead of relying on a single scalar like entropy to rank layers, we propose a layer-grouping strategy in which a single shared codebook is used for a set of statistically similar linear layers. Sharing codebooks reduces the total number of parameters and allows data from multiple layers to train a more robust codebook. To mitigate potential performance loss from using a shared codebook instead of dedicated ones, we increase the number of stages, S, in the RVQ architecture. We hypothesize that the enhanced intrinsic error-correction capability of multi-stage RVQ will compensate for—and potentially surpass—the information loss caused by layer grouping, achieving an optimal balance between compression efficiency and model accuracy.

To systematically implement this strategy, we developed a three-step, LLM-based, data-driven process for identifying layer groups. First, for each linear layer with weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, we extract a probability distribution representation. The weight matrix W is flattened into a one-dimensional vector $x \in \mathbb{R}^{d_{\text{out}} d_{\text{in}}}$. The range of x is uniformly partitioned into B bins to construct a histogram, where h_k denotes the number of elements in the k^{th} bin. The histogram is then normalized to produce a discrete probability distribution:

$$p_k = \frac{h_k + \alpha}{\sum_{j=1}^{B} (h_j + \alpha)}, \quad k = 1, \ldots, B \tag{8}$$

where a small constant $(\alpha > 0)$ is added for numerical stability.

The second step is to quantify the similarity between these weight distributions. We use the

Jensen-Shannon Divergence (JSD), a symmetric and bounded measure of the difference between two probability distributions. This choice has distinct advantages over entropy (which does not capture pairwise shape differences) and Kullback-Leibler Divergence (which is asymmetric and unbounded, making it unsuitable for direct use in clustering algorithms). The JSD between distributions P and Q is defined based on KLD as follows:

$$D_{JS}(P\|Q) = \frac{1}{2}D_{KL}(P\|M) + \frac{1}{2}D_{KL}(Q\|M) \quad (9)$$

where $M = \frac{1}{2}(P + Q)$. Thereafter, we compute the pairwise JSD for all linear layers to construct a divergence matrix $D \in \mathbb{R}^{n\times n}$, where n is the total number of linear layers and $D_{ij} = D_{JS}(P_i, P_j)$.

The final step is to cluster the layers based on the divergence matrix. A challenge is that standard clustering algorithms, such as k-means, operate on Euclidean distances in a vector space. To address this, we first apply multidimensional scaling (MDS) (Kruskal, 1964a,b; Borg and Groenen, 2005) to the divergence matrix D. MDS is a dimensionality reduction technique that embeds the n layers into a low-dimensional Euclidean space $\mathbb{R}^d$ such that the Euclidean distances between the embedded points approximate the original JSD. The MDS output is a coordinate matrix $X \in \mathbb{R}^{n\times d}$, which serves as input to the k-means algorithm. K-means then partitions the n layers into k clusters, with each cluster representing layers of similar statistical distributions that share the same set of codebooks during quantization.

4. Experimental Results

4.1. Evaluation of LLM Compression Quality

Following the experimental setup outlined in Appendix C.2, we conducted a series of experiments to evaluate the effectiveness of the proposed method. These experiments assess both core language modeling performance and broader capabilities in general understanding and reasoning. Our approach, targeting an average bitwidth of approximately 2.5 bits, was benchmarked against leading state-of-the-art weight-only PTQ methods, QuIP# (Tseng et al., 2024a), QTIP (Tseng et al., 2024b) and AQLM (Egiazarian et al., 2024). For a comprehensive comparison, we report the performance of these methods under two conditions: with and without post-quantization fine-tuning.

In our evaluation, we investigated two distinct quantization configurations to analyze the sensitivity of different architectural components:

1. **Comprehensive Quantization:** A configuration where all linear layers in the model, including those in the multi-head self-attention and multi-layer perceptron blocks, are quantized.
2. **Selective Quantization:** A configuration where the quantization process is applied to all linear layers, except for two layers identified as having critical functional roles and high sensitivity to quantization error.

In the second configuration, two layers are maintained at their original FP16 precision: the attention output projection layer and the multilayer perceptron (MLP) down-projection layer. The attention output projection layer aggregates information from multiple attention heads into a shared representation space, while the MLP down-projection layer reduces the feature dimension after expansion by the MLP, preserving the transformer block's output size. Beyond these two functionally critical layers, an additional configuration is considered to further mitigate performance degradation during quantization.

The quantitative results are summarized in Tables 1 and 2. Analysis indicates that, when applying the selective quantization configuration (excluding the attention output projection and MLP down-projection layers), our method achieves lower perplexity values, demonstrating high competitiveness with the baseline techniques. This advantage becomes particularly pronounced in the scenario without post-quantization fine-tuning, which indicates the effectiveness of our proposed intrinsic optimization mechanisms. However, when the comprehensive quantization configuration is applied, our model exhibits a slight increase in perplexity with state-of-the-art PTQ algorithms. Nevertheless, even in this more demanding configuration, our method maintains competitive performance, especially under the no-fine-tuning condition, confirming the robustness of the proposed framework.

4.2. Effectiveness of Proposed Methods

To quantify the contribution of each component in our framework, we conducted a detailed ablation study. Experiments were performed on the Llama 2 7B model (Touvron et al., 2023) using the comprehensive quantization configuration (all linear layers) while maintaining an average bitwidth of 2.5 bits. Performance was measured via perplexity on the WikiText-2 dataset (Merity et al., 2017). Table 3 summarizes results for different combinations of the three key components: (1) KLD-based calibration, (2) NSVQ-based codebook optimization, and (3) the layer-grouping strategy.

Results indicate that applying only a basic RVQ method with a mean squared error (MSE) objective per layer leads to substantial performance

Model	Method	Config	Bitwidth	WikiText-2↓	C4↓
Llama 2 7B	Baseline	–	32	4.80	5.46
	Baseline	–	16	5.12	6.63
	QuIP#★	C	2.0	6.18	8.75
	QuIP#	C	2.0	8.22	11.00
	QTIP★	C	2.0	5.86	8.22
	AQLM★	C	2.3	5.92	7.86
	AQLM	C	2.3	6.29	8.11
	Proposed	S	2.5	**5.78**	**7.93**
	Proposed	C	2.5	**7.27**	**10.72**

Table 1: Comprehensive performance comparison of Llama 2 7B on evaluation benchmarks (Perplexity).

Model	Method	Config	Bitwidth	Wino Grande↑	PIQA↑	Hella Swag↑	ARC-E↑	ARC-C↑	MMLU↑
Llama 2 7B	Baseline	–	16	69.3	77.5	57.1	76.0	42.9	40.7
	QuIP#★	C	2.0	65.8	75.6	52.2	71.9	38.0	30.9
	QTIP★	C	2.0	67.3	76.2	53.7	73.4	39.8	36.6
	AQLM★	C	2.3	65.3	76.9	53.4	74.0	39.5	35.6
	Proposed	S	2.5	**67.3**	**77.3**	**54.7**	**74.5**	**41.1**	**35.5**
	Proposed	C	2.5	**66.4**	**74.8**	**49.3**	**70.6**	**35.3**	**30.7**

Table 2: Comprehensive performance comparison of Llama 2 7B on evaluation benchmarks (Zero-shot Accuracy).

Notes. ★ denotes result obtained after post-quantization fine-tuning. Arrows: ↓ = lower better; ↑ = higher better. Abbreviations: C = Comprehensive Quantization; S = Selective Quantization.

degradation. Although MSE minimization reduces magnitude-wise errors, it drives the codebooks toward mean values, failing to capture critical statistical features, particularly high-impact outliers. Moreover, due to the non-differentiable nature of the quantization function, gradient-based optimization is severely impeded without an effective derivative estimation mechanism, causing premature and suboptimal convergence.

A detailed analysis of the individual components shows that the layer-grouping strategy offers a modest accuracy improvement, but its effect is limited without a well-defined optimization objective for the codebooks. Similarly, using calibration data with the KLD objective enhances performance to some extent; however, optimization remains constrained by the non-differentiable nature of the quantization function. In contrast, incorporating the NSVQ gradient estimation technique—either alone or in combination with the other components—yields a substantial improvement in model quality. This highlights the critical role of NSVQ in providing a meaningful gradient signal that enables effective codebook optimization.

Overall, the results in Table 3 indicate that optimal performance is achieved only when all three components are combined. The KLD minimization objective with calibration data, together with the NSVQ gradient estimator, effectively guides codebook optimization and preserves the critical statistical properties of the weights. Once this optimization is properly directed, the layer-grouping strategy—leveraging the structural properties of LLMs and the hierarchical design of residual quantization—further enhances model accuracy. These findings confirm the synergistic effect of the proposed components, with each contributing a distinct yet complementary role in achieving maximal quantization performance.

4.3. Inference Latency

While the primary goal of this study is to optimize model accuracy at high compression ratios, a thorough evaluation of any compression algorithm must also consider inference latency. Recognizing that the multi-stage quantization architecture can introduce computational overhead during dequantization, we developed a custom GPU CUDA kernel to parallelize this process and accelerate inference for the quantized model.

To assess the impact of this optimization, we conducted a latency analysis by measuring the time required for autoregressive token generation during perplexity evaluation on the WikiText-2 dataset, fol-

Methods	KLD	NSVQ	LGS	WikiText-2↓
Baseline	–	–	–	52,406.05
LGS	–	–	◯	45,898.45
KLD	◯	–	–	35,673.72
KLD + LGS	◯	–	◯	19,047.07
NSVQ	–	◯	–	13,872.31
NSVQ + LGS	–	◯	◯	4,846.52
KLD + NSVQ	◯	◯	–	264.55
KLD + NSVQ + LGS	◯	◯	◯	**7.27**

Table 3: Ablation Study of Proposed Methods on Llama 2 7B (2.5-bit).

Notes. ◯ indicates the component is enabled; – indicates it is disabled. Arrows: ↓ = lower better. Abbreviations: KLD = Kullback-Leibler divergence-based calibration; NSVQ = Noise Substitution in Vector Quantization-based codebook optimization; LGS = Layer-grouping Strategy

lowing the experimental protocol described in Section 4.1. Table 4 compares the baseline FP16 and FP32 models with our quantized model, both with and without the optimized CUDA kernel. The quantized model was configured to an average bitwidth of 2.5 bits and employed the comprehensive quantization strategy.

The results highlight the exceptional efficiency of the custom CUDA kernel, achieving an inference speed roughly 64 times faster than the non-optimized implementation. Compared to the baseline models, the optimized quantized model attains a significant speedup of approximately 4.4× over the FP32 model. However, a trade-off is observed relative to the FP16 model, with the quantized model exhibiting about 1.5× higher latency.

Method	Bitwidth	Execution Time (min)
Baseline	16	0.93
Baseline	32	6.38
Proposed (w/o kernels)	2.5	91.45
Proposed (w/ kernels)	**2.5**	**1.43**

Table 4: Inference latency comparison on Llama 2 7B.

The inherent computational cost of the dequantization process can explain this discrepancy. Although the CUDA kernel efficiently parallelizes the codebook lookups and result aggregations, these operations still introduce additional computational work that is irrelevant in direct matrix-vector multiplications, which are highly optimized at the hardware level for formats such as FP16. Nevertheless, this result confirms that with the targeted hardware acceleration, the proposed quantization method is not only computationally feasible but can also achieve competitive inference speeds.

5. Limitations

Although the proposed method achieves competitive compression performance at ultra-low bitwidths, several limitations remain to be addressed in future work.

The first and most notable limitation lies in our quantization strategy. Specifically, to achieve superior quality compared to other state-of-the-art methods, our approach relies on a selective quantization scheme. In this scheme, two critical architectural components must be maintained at their original precision: the attention output projection layers, which are responsible for aggregating information from multiple attention heads into a common representation space, and the MLP down-projection layers, which reduce the dimensionality of the feature space following non-linear transformations. This dependency highlights the pronounced sensitivity of these specific functional layers to quantization error. Consequently, developing more sophisticated optimization techniques to enable full linear layer quantization without degrading model performance remains a significant technical challenge for future research.

Second, the generalizability of the current methodology is constrained by the experimental scope. The results and evaluations presented in this paper have been validated on a single architecture: the Llama 2 7-billion-parameter (Llama 2 7B) model. To establish the robustness and scalability of the proposed method, future studies must extend the experimental scope to larger variants within the Llama family (e.g., 13B, 70B), as well as to other advanced open-source LLM architectures such as Gemma, Qwen, and Mistral. Diversifying the evaluated models will facilitate a more comprehensive assessment of the proposed quantization framework's compatibility with the varying weight distribution characteristics inherent to different architectural paradigms.

Third, the current evaluation framework relies predominantly on standard quantitative metrics to assess accuracy and latency. While these metrics provide a solid baseline overview, they are in-

sufficient to comprehensively capture the behavior of the quantized model. To gain deeper insights, future work should incorporate multi-dimensional evaluation methodologies, including assessments of generation quality, complex logical reasoning capabilities, and human evaluation. Furthermore, given that the ultimate objective of LLM compression is deployment on resource-constrained hardware, the current lack of empirical performance profiling on actual edge devices represents a gap that must be addressed in subsequent research.

Finally, the experiments and evaluations in this study were conducted exclusively on English corpora and tasks. Given the vast lexical and semantic diversity of natural languages, whether the quantized model can maintain stable performance across different linguistic spaces remains an open question. Therefore, extending the evaluation framework to encompass multilingual tasks will be a necessary step to demonstrate the reliability and global applicability of the proposed compression method.

6. Conclusion and Future Work

This study has successfully proposed and validated a comprehensive framework for PTQ, specifically designed to compress the weights of LLMs to ultra-low precision levels ranging from 2 to 3 bits. This framework integrates three core techniques. First, guided by KLD minimization, a parameter calibration process is employed to preserve the statistical distribution characteristics of the original weight matrices. Second, a learnable codebook optimization mechanism is implemented using NSVQ, providing a robust solution for accurate gradient approximation and computation within a discrete space. Finally, to optimize the storage footprint and enhance codebook quality, we introduce a layer grouping strategy that leverages statistical similarity measures among weight distributions. Experimental evaluations confirm the efficacy of the proposed framework, demonstrating inference performance competitive with current state-of-the-art techniques. Notably, these results are achieved entirely without post-quantization fine-tuning, effectively eliminating the massive computational overhead required by many prior methods.

Despite these highly promising results, the current methodology exhibits certain limitations regarding the scope of quantization and the breadth of the experimental evaluation (as detailed in Section 5). Addressing these limitations, future research will focus on developing a more sophisticated and comprehensive optimization framework. A primary objective is to enable full quantization across all linear layers of the LLMs without inducing significant degradation in the original model's accuracy. Concurrently, establishing a multi-dimensional evaluation strategy—encompassing diverse LLM architectures, multilingual benchmarks, and on-device testing on edge devices—will be essential to validate the generalizability of the proposed method.

To address the challenge of full-model quantization, a promising approach is the further development of the current layer grouping strategy. This advancement requires not only the investigation of more precise data clustering algorithms but also the integration of an adaptive bit-width allocation scheme. By strategically assigning higher bit-widths to functionally sensitive layer clusters while applying more aggressive compression to groups with higher error tolerance, this method promises to establish an optimal global equilibrium. This optimized trade-off between the maximum compression ratio and accuracy preservation will unlock immense potential for deploying massive LLMs on severely resource-constrained hardware systems.

7. Bibliographical 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 '07, page 1027–1035, USA. Society for Industrial and Applied Mathematics.

Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE.

Alireza Behtash, Marijan Fofonjka, Ethan Baird, Tyler Mauer, Hossein Moghimifam, David Stout, and Joel Dennison. 2025. Universality of layer-level entropy-weighted quantization beyond model architecture and size.

Ingwer Borg and Patrick J. F. Groenen. 2005. *Modern multidimensional scaling: Theory and applications*, 2 edition. Springer Series in Statistics. Springer, New York, NY.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. 2023. QuIP: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by

residual vector quantization. *Sensors (Basel)*, 10(12):11259–11273.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme compression of large language models via additive quantization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training quantization for generative pre-trained transformers.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper.

J B Kruskal. 1964a. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27.

J B Kruskal. 1964b. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2023. AWQ: Activation-aware weight quantization for LLM compression and acceleration.

S Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–137.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024a. QuIP#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. 2024b. QTIP: Quantization with trellises and incoherence processing. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Mohammad Hassan Vali and Tom Backstrom. 2022. NSVQ: Noise substitution in vector quantization for machine learning. *IEEE Access*, 10:13598–13610.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

8. Language Resource References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press. PID https://doi.org/10.1609/AAAI.V34I05.6239.

Clark, Peter and Cowhey, Isaac and Etzioni, Oren and Khot, Tushar and Sabharwal, Ashish and Schoenick, Carissa and Tafjord, Oyvind. 2018. *Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*. PID https://doi.org/10.48550/arXiv.1803.05457.

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. PID https://doi.org/10.5281/zenodo.5371628.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. PID https://doi.org/10.48550/arXiv.2009.03300.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. PID https://doi.org/10.48550/arXiv.1609.07843.

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. PID https://doi.org/10.48550/arXiv.1910.10683.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106. PID https://doi.org/10.1145/3474381.

Touvron, Hugo and Martin, Louis and Stone, Kevin and Albert, Peter and Almahairi, Amjad and Babaei, Yasmine and Bashlykov, Nikolay and Batra, Soumya and Bhargava, Prajjwal and Bhosale, Shruti and Bikel, Dan and Blecher, Lukas and Ferrer, Cristian Canton and Chen, Moya and Cucurull, Guillem and Esiobu, David and Fernandes, Jude and Fu, Jeremy and Fu, Wenyin and Fuller, Brian and Gao, Cynthia and Goswami, Vedanuj and Goyal, Naman and Hartshorn, Anthony and Hosseini, Saghar and Hou, Rui and Inan, Hakan and Kardas, Marcin and Kerkez, Viktor and Khabsa, Madian and Kloumann, Isabel and Korenev, Artem and Koura, Punit Singh and Lachaux, Marie-Anne and Lavril, Thibaut and Lee, Jenya and Liskovich, Diana and Lu, Yinghai and Mao, Yuning and Martinet, Xavier and Mihaylov, Todor and Mishra, Pushkar and Molybog, Igor and Nie, Yixin and Poulton, Andrew and Reizenstein, Jeremy and Rungta, Rashi and Saladi, Kalyan and Schelten, Alan and Silva, Ruan and Smith, Eric Michael and Subramanian, Ranjan and Tan, Xiaoqing Ellen and Tang, Binh and Taylor, Ross and Williams, Adina and Kuan, Jian Xiang and Xu, Puxin and Yan, Zheng and Zarov, Iliyan and Zhang, Yuchen and Fan, Angela and Kambadur, Melanie and Narang, Sharan and Rodriguez, Aurelien and Stojnic, Robert and Edunov, Sergey and Scialom, Thomas. 2023. *Llama 2: Open foundation and fine-tuned chat models*. PID https://doi.org/10.48550/arXiv.2307.09288.

Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. RedPajama: an open dataset for training large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. PID https://doi.org/10.48550/arXiv.2411.12372.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics. PID https://doi.org/10.18653/v1/P19-1472.

A. Codebook Initialization Procedure

The efficacy of the learnable codebook optimization process is highly dependent on the quality of its initial parameters. A suboptimal starting point can lead to slow convergence or convergence to a poor local minimum. To establish a robust starting point and minimize sensitivity to random initialization, we employ a two-stage initialization procedure for the codebooks before commencing the gradient-based optimization process.

The first stage employs the k-means++ algorithm (Arthur and Vassilvitskii, 2007) to strategically select an initial set of centroids. Unlike random selection, k-means++ is a probabilistic seeding strategy designed to ensure that the initial centroids are well-distributed throughout the data space. The process begins by selecting the first centroid uniformly at random from the data points. Subsequently, each subsequent centroid is chosen sequentially from the remaining data points, with a selection probability for each point proportional to its squared distance to the nearest already-selected centroid. Specifically, the probability $P(x_j)$ of selecting a data point x_j as the next centroid is given by:

$$P(x_j) = \frac{D(x_j)^2}{\sum_i D(x_i)^2} \tag{10}$$

where $D(x_j)^2$ is the squared Euclidean distance from point x_j to the nearest centroid among those already chosen.

After obtaining the initial set of centroids from k-means++, the second stage consists of performing a small number of iterations of the standard k-means algorithm, also known as Lloyd's algorithm (Lloyd, 1982), to preliminarily refine the positions of these centroids. Each iteration of Lloyd's algorithm comprises two steps:

1. **Assignment Step:** Each data vector x_p in the training set is assigned to the cluster $S_i^{(t)}$ corresponding to the nearest centroid $m_i^{(t)}$, based on the squared Euclidean distance.

$$S_i^{(t)} = \Big\{ x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2, \quad \forall j = 1, \ldots, K \Big\} \tag{11}$$

2. **Update Step:** The position of each centroid is updated by computing the mean vector (centroid) of all data points assigned to its cluster in the previous step.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{12}$$

This comprehensive initialization procedure ensures that the codebooks begin the optimization process from a high-quality initial state, rather than from a random or zero-initialized configuration. By providing a meaningful starting point, this method accelerates the convergence of the subsequent gradient-based optimization phase and increases the likelihood of reaching a high-quality local minimum, thereby improving the final performance of the trained codebooks.

B. Estimating Average Bit-Width

A critical aspect of our quantization method is determining the average bits per parameter (b_{avg}) post-compression. This value accounts not only for the bits used to store the quantization indices but also for the storage overhead of auxiliary components, namely the codebooks and scaling factors. Based on the architecture described in 3.1, for an input weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, the total storage cost (B_{total}) comprises three main components:

1. **Codebook Storage Cost (B_{codebook}):** The codebooks C have dimensions (S, M, K, D) and are stored in half-precision (FP16), corresponding to 16 bits per element.

$$B_{\text{codebook}} = S \cdot M \cdot K \cdot D \cdot 16 \tag{13}$$

2. **Index Storage Cost (B_{index}):** The quantized weight matrix is represented by an index tensor of dimensions (S, d_{out}, M). Each index requires $\log_2(K)$ bits.

$$B_{\text{index}} = S \cdot d_{\text{out}} \cdot M \cdot \log_2(K) \tag{14}$$

3. **Scaling Factor Storage Cost (B_{scale}):** The per-channel scaling factors have dimensions (S, d_{in}) and are also stored in half-precision (FP16).

$$B_{\text{scale}} = S \cdot d_{\text{in}} \cdot 16 \tag{15}$$

Therefore, the average bits per parameter of the original weight matrix is calculated by dividing the total storage cost by the initial number of parameters ($N_{\text{params}} = d_{\text{out}} \cdot d_{\text{in}}$). The full formula is expressed as:

$$b_{\text{avg}} = \frac{B_{\text{total}}}{N_{\text{params}}} = \frac{B_{\text{codebook}} + B_{\text{index}} + B_{\text{scale}}}{d_{\text{out}} \cdot d_{\text{in}}} \tag{16}$$

C. Experimental Setup

This section details the computational infrastructure and experimental configuration underlying the proposed LLMs quantization framework. We first specify the hardware environment used to perform all

quantization and evaluation experiments. We then describe the linguistic resources and benchmark datasets employed for quantization, calibration and evaluation.

In addition, we provide a comprehensive account of the experimental configurations and hyperparameter settings adopted throughout the study. This includes a detailed specification of the layer grouping strategy and the codebook optimization procedure. These methodological details are reported to ensure reproducibility and to facilitate rigorous evaluation of the proposed quantization framework for LLMs.

C.1. Hardware

All experiments, including the quantization process and subsequent evaluations, were performed on a single hardware platform equipped with an NVIDIA RTX A6000 GPU. Detailed specifications of the platform are provided in Table 5.

Specification	Details
Graphics processor	GA102
Cores	10752
Texture mapping units (TMUs)	336
Render output units (ROPs)	112
Memory size	48 GB
Memory type	GDDR6
Bus width	384 bits

Table 5: Specifications of NVIDIA RTX A6000.

C.2. Language Resources

To ensure fair and reproducible comparisons with state-of-the-art methods, we use the Llama 2 7-billion-parameter model (Llama 2 7B) (Touvron et al., 2023) as our primary evaluation subject. The Llama family, released by Meta AI, is a widely used open-source LLM with publicly available weights, facilitating research, fine-tuning, and deployment.

For calibration (as described in Section 3.2), we use a subset of the RedPajama dataset (Weber et al., 2024). The diversity of this corpus ensures that the calibration process captures representative statistical characteristics of natural language.

To evaluate the post-quantization performance of the model, we employ a diverse set of datasets and tasks. First, core language modeling capabilities are assessed via perplexity on two standard datasets: WikiText-2 (Merity et al., 2017) and C4 (Colossal Clean Crawled Corpus) (Raffel et al., 2020). To evaluate general understanding and reasoning, we conduct zero-shot accuracy assessments across a range of downstream tasks, including common-sense reasoning (WinoGrande (Sakaguchi et al., 2021), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019)), scientific reasoning (ARC-Easy and ARC-Challenge (Clark et al., 2018)), and multi-domain knowledge (MMLU (Hendrycks et al., 2021)). All evaluations are performed using the standard Language Model Evaluation Harness (Gao et al.) toolkit to ensure consistency and comparability.

C.3. Layer Grouping Configuration

The layer grouping strategy is implemented through a two-stage hierarchical process. Prior to applying the data-driven clustering algorithm, we perform an initial coarse partitioning, classifying all of the model's linear layers into four macro-groups. The purpose of this preliminary classification is twofold: first, to isolate layer groups with critical functional roles and known high sensitivity to quantization error, such as the attention output projection and MLP down-projection layers; and second, to ensure uniformity of tensor shapes within each macro-group, thereby facilitating batch processing during quantization.

After partitioning the layers into these macro-groups, we apply the clustering procedure based on Jensen-Shannon Divergence and Multidimensional Scaling as described in Section 3.4. The k-means algorithm is applied independently within each macro-group, with a predefined number of sub-clusters for each group. For the Llama 2 7B model, the specific grouping configuration is presented in Table 6. This process results in a total of 88 fine-grained layer groups distributed across the four main macro-groups.

C.4. Codebook Optimization Hyperparameters

To achieve the target average bit-width of 2.5 bits for the Llama 2 7B model, we configured the RPQ architecture and the codebook optimization procedure with specific hyperparameters. The RPQ architecture was configured with $S = 5$ stages, codebooks containing $K = 16$ codewords per sub-codebook, and a codeword dimension of $D = 8$.

The optimization of the codebooks and learnable scaling factors was performed using the Adam optimizer (Kingma and Ba, 2015). The Adam hyperparameters were set to a learning rate of 1×10^{-5}, with momentum coefficients $\beta_1 = 0.90$ and $\beta_2 = 0.95$. The training was conducted for a maximum of 1000 epochs. To ensure effective convergence, an early stopping mechanism was implemented. This mechanism terminates the optimization if the quantization loss does not improve within a patience window of 50 epochs.

Layer Group	Clusters	Histogram Bins	MDS Dimensions
Attention Q/K/V projections	64	256	4096
Attention output projection	4	256	4096
MLP gate and up projections	16	256	4096
MLP down projection	4	256	4096

Table 6: Hyperparameter configuration for layer grouping in Llama 2 7B.

D. Quantization Time

An important aspect of our methodology to consider is the computational cost associated with the quantization process. The optimization of the learnable codebooks is a computationally intensive task, primarily due to its iterative nature and the necessity of maintaining high numerical precision. Specifically, the use of high-precision (float64) operations is required to ensure numerical stability and accurate gradient updates throughout the optimization. It is crucial to emphasize, however, that this is a one-time, offline cost incurred only during the quantized model preparation stage. Once the quantization is complete, this cost has no impact on the inference latency of the deployed model.

It should be noted that this quantization time is not a fixed constant but represents a tunable trade-off between computational cost and final compression quality. By adjusting the quantization architecture's hyperparameters, such as reducing the number of stages (S) or increasing the subspace dimension (D), the time required for optimization can be significantly reduced. However, such adjustments typically come at the cost of a degradation in the quantized model's accuracy, as they diminish the representational capacity of the quantizer.

To provide a concrete quantitative measure of this cost, the quantization of the entire Llama 2 7B model in this study, using the configuration detailed in Appendix 3.4 and C.4, required approximately 35 hours of computation on a single A6000 GPU (as described in Appendix C.1).