VPTQ: Extreme Low-bit Vector Post-Training Quantization for Large Language Models

Yifei Liu[‡,†,*] **Jicheng Wen**[†] **Yang Wang**[†,◇] **Shengyu Ye**[‡,†]

Li Lyna Zhang[†] **Ting Cao**[†] **Cheng Li**[‡] **Mao Yang**[†]

[‡]University of Science and Technology of China

[†]Microsoft

{v-liuyifei, jicheng.wen, Yang.Wang92, v-shengyuye, lzhani, ting.cao, maoyang}@microsoft.com, chengli7@ustc.edu.cn

Abstract

Scaling model size significantly challenges the deployment and inference of Large Language Models (LLMs). Due to the redundancy in LLM weights, recent research has focused on pushing weight-only quantization to extremely low-bit (even down to 2 bits). It reduces memory requirements, optimizes storage costs, and decreases memory bandwidth needs during inference. However, due to numerical representation limitations, traditional scalar-based weight quantization struggles to achieve such extreme low-bit. Recent research on Vector Quantization (VQ) for LLMs has demonstrated the potential for extremely low-bit model quantization by compressing vectors into indices using lookup tables.

In this paper, we introduce **Vector Post-Training Quantization (VPTQ)** for extremely low-bit quantization of LLMs. We use Second-Order Optimization to formulate the LLM VQ problem and guide our quantization algorithm design by solving the optimization. We further refine the weights using Channel-Independent Second-Order Optimization for a granular VQ. In addition, by decomposing the optimization problem, we propose a brief and effective codebook initialization algorithm. We also extend VPTQ to support residual and outlier quantization, which enhances model accuracy and further compresses the model. Our experimental results show that VPTQ reduces model quantization perplexity by 0.01-0.34 on LLaMA-2, 0.38-0.68 on Mistral-7B, 4.41-7.34 on LLaMA-3 over SOTA at 2-bit, with an average accuracy improvement of 0.79-1.5% on LLaMA-2, 1% on Mistral-7B, 11-22% on LLaMA-3 on QA tasks on average. We only utilize 10.4-18.6% of the quantization algorithm execution time, resulting in a 1.6-1.8× increase in inference throughput compared to SOTA. Our code is available at https://github.com/microsoft/VPTQ.

Table 1: LLM Quantization Algorithm Comparison. VPTQ balances all dimensions and achieves SOTA.

	VPTQ	AQLM	QuIP#	GPTVQ	GPTQ	AWQ
Effective Bitwidth	↓	↓	↓	↑	↑↑	↑↑
Accuracy @ Low-bit	↑	↑	↑	↓	↓↓	↓↓
Quantization Time Cost	↓	↑↑	↓	↓	↓	↓
Inference Throughput	↑	↑	↓	↑	↑	↑

1 Introduction

Large language models (LLMs) (Touvron et al., 2023; Meta, 2024) have shown excellent performance across various complex tasks as their sizes increase. However, the enormous weight of LLMs poses significant challenges for efficient inference and practical deployment. For instance, storing the LLaMA-2 70B model weights in FP16 format requires 140GB of memory, surpassing the capacity of high-end GPUs and necessitating multi-GPU deployment. This huge size significantly affects memory capacity and hard disk storage and requires substantial bandwidth for inference. Weight-only quantization is a mainstream model compression technique that effectively reduces the model's size by representing floating-point numbers with fewer bits.

In weight-only quantization of LLMs, a prominent method is Post-Training Quantization (PTQ). PTQ quantizes model weights directly without retraining the model. Typically, PTQ only involves converting model weights into lower-bit fixed-point numbers. Currently, the main approach in PTQ is scalar quantization, which converts each scalar weight in the model into a lower bit value. Recent work (Frantar et al., 2023; Lin et al., 2023; Xiao et al., 2023; Lee et al., 2024; Chee et al., 2023) has achieved near-original model accuracy with 3-4 bit quantization. Table 1 summarizes the char-

[*]Contribution during internship at Microsoft Research

[◇]Corresponding author

This paper is the result of an open-source research project, and the majority work of the project is accomplished in April 2024.

acteristics of typical scalar quantization methods (GPTQ, AWQ) in LLM. However, due to the limitations of numerical representation, traditional scalar-based weight quantization struggles to achieve extremely low-bit levels. For instance, with 2-bit quantization, we can only use four numerical values to represent model weights, which severely limits the range of weight representation. Although BitNet (Wang et al., 2023; Ma et al., 2024) has enabled quantization-aware training that can quantize weights to below 2 bits during the model's pre-training phase, this approach requires substantial GPU cluster resources to maintain reasonable accuracy.

Recent studies (van Baalen et al., 2024; Tseng et al., 2024; Egiazarian et al., 2024) have explored an efficient method of weight-only quantization known as Vector Quantization (VQ).

VQ is a data compression technique that maps high-dimensional vectors to a set of predefined lower-dimensional vectors stored in codebooks (lookup tables). During encoding, each data point is represented by the index of a corresponding vector in the codebook, and during decoding, the original data is approximated using these indices. This method substantially reduces the storage requirements for data while allowing for the quick reconstruction of original vectors through simple index references. VQ achieves more effective data compression than scalar quantization by leveraging correlations and redundancies across different data dimensions. By detecting and leveraging interdependence, VQ can encode complex multidimensional data with fewer bits, thus achieving higher compression ratios and reduced bit width.

While Vector Quantization (VQ) shows promise in extreme low-bit weight compression for Large Language Models (LLMs), it faces several significant challenges. Table 1 compares the strengths and weaknesses of various VQ algorithms in multiple dimensions.

The first challenge is ensuring the accuracy after extreme low-bit VQ quantization. Unlike scalar quantization, the quantization granularity of VQ algorithms is vector-based. The quantization may introduce additional accumulation errors due to the simultaneous quantization of multiple numbers. For example, GPTVQ (van Baalen et al., 2024) uses the Second-Order Optimization method to implement PTQ. However, GPTVQ accumulates quantization errors within vector quantization, leading to an inevitable increase in quantization errors as the vector length increases. This prevents the use of longer vectors and, consequently, limits the compression ratio.

The second challenge lies in efficiently executing VQ quantization on LLMs. VQ can compress vectors in the weight matrix into indices, but these indices are discrete, non-differentiable integers. This introduces difficulties in implementing VQ quantization methods through model training. For instance, AQLM (Egiazarian et al., 2024) employs beam search and backpropagation to quantize and update centroids in lookup tables. VQ necessitates additional gradient estimation, slowing the convergence of model quantization training and requiring intensive training efforts to achieve better accuracy.

The third challenge arises as the dequantization overhead in VQ model inference. To reduce quantization errors, complex data preprocessing methods may be used to process weights. QuIP# (Tseng et al., 2024) introduces incoherence processing using the randomized Hadamard transform for the weight matrix before VQ. These preprocessing steps can reduce quantization errors and improve model accuracy. However, postprocessing must be performed in real time during model inference, which can severely impact throughput in inference.

VPTQ seeks to bypass the limitations of current VQ by offering a lightweight and efficient approach exclusively for extreme low-bit weight quantization.

In this paper, we present **Vector Post-Training Quantization (VPTQ)**, a novel approach for extremely low-bit quantization of LLMs.

1. VPTQ achieves SOTA accuracy results on extremely low-bit LLMs. We formulate the quantization problem as an optimization problem and employ Second-Order Optimization to guide our quantization algorithm design. By Channel-Independent Second-Order Optimization, VPTQ reduces model quantization perplexity by 0.01-0.34, 4.41-7.34, 0.38-0.5 on LLaMA-2/3/Mistral-7B, respectively, over SOTA at 2-bit, with an accuracy improvement of 0.79-1.5%,11-22%,1%, on LLaMA-2/3/Mistral-7B in QA tasks on average.
2. VPTQ can transform LLMs into extremely low-bit models with a minor quantization algorithm overhead. Under the guidance of the optimization problem, we transform the quan-

tization algorithm into a heuristic algorithm to solve the optimization problem. We also analyze and propose a brief and effective codebook initialization algorithm to reduce the extra overhead of centroid training and updates. Experiments show that VPTQ only requires 10.4-18.6% of the quantization algorithm execution time compared to existing SOTA results.

3. VPTQ has low dequantization overhead. VPTQ algorithm quantizes all the weights in every Linear Operator in the model into an index matrix and codebooks. During model inference, we only need to dequantize the weight matrix by reading centroids from the codebook according to the index before executing the operator. The models quantized by VPTQ result in 1.6-1.8× improvement in inference throughput compared to SOTA.

2 Background and Motivation

2.1 Post Training Quantization in LLM

Post-Training Quantization (PTQ) (LeCun et al., 1989; Hassibi et al., 1993; Hassibi and Stork, 1992; Frantar et al., 2023; Singh and Alistarh, 2020) aims to decrease model weight size by simplifying the numerical representation and seeking to maintain the model's accuracy without retraining the model. We can formulate PTQ as the following optimization problem:

$$\begin{aligned} &\arg\min \quad \mathbb{E}[\mathcal{L}(\mathbf{X}, \mathbf{W} + \Delta\mathbf{W}) - \mathcal{L}(\mathbf{X}, \mathbf{W})] \\ &\approx \Delta\mathbf{W}^T \cdot g(\mathbf{W}) + \frac{1}{2}\Delta\mathbf{W}^T \cdot H(\mathbf{W}) \cdot \Delta\mathbf{W} \end{aligned}$$

where $\mathbf{W}$ is the original model weights, $\hat{\mathbf{W}}$ is quantized weights, and $\Delta\mathbf{W} = \hat{\mathbf{W}} - \mathbf{W}$ represents the weight quantization error. The loss of the model task is $\mathcal{L}$. The optimization object is to minimize the impact of model quantization on the model task, which means minimizing the expected deviation of the loss function.

PTQ typically employs a concise and accurate method for analyzing the above optimization problem: Second-Order Optimization. Following a Taylor series expansion, this method breaks down the optimization goal into first-order, second-order, and higher-order terms. $g(\mathbf{W})$ and $H(\mathbf{W})$ represent the gradient and Hessian of task loss $\mathcal{L}$, respectively. It often assumes that the model has already reached local optimum before model quantization, which means that the first-order term is nearly zero.

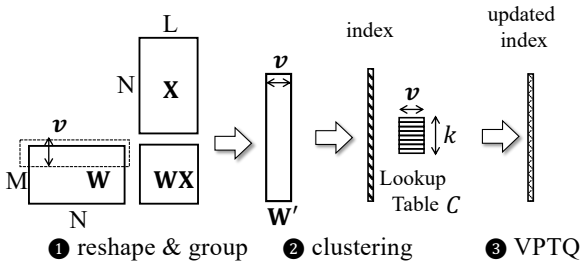

Figure 1: Vector Quantization in Weight Quantization

Higher-order terms exert a minor effect on the optimization goal, and we typically disregard interactions among weights between different layers. Consequently, we can simplify the optimization problem by focusing on optimizing the second-order term and then define the following optimization problem:

$$\begin{aligned} \arg\min_{\Delta\mathbf{W}} \quad & \Delta\mathbf{W}^T \cdot H(\mathbf{W}) \cdot \Delta\mathbf{W}, \\ \text{s.t.} \quad & \Delta\mathbf{W} = \mathbf{0} \end{aligned} \tag{1}$$

The objective of the optimization problem is to minimize the second-order error in model quantization, subject to the constraint that the change in model weights is as minimized as possible, i.e., $\Delta\mathbf{W} = \mathbf{0}$.

2.2 Vector Quantization in Neural Networks

VQ is a key method for efficient lossy data compression (Gersho, 1979). Its objective is to reduce the distortion by mapping high-dimensional original data to a lower-dimensional space represented by a lookup table (Eq. 2). VQ maps original vectors ($\mathbf{W}'$) from the vector space to a finite set of vectors, which is commonly referred to as a codebook (lookup table, $\mathcal{C}$). Each vector in the original space approximates the closest vector (centroid $\mathcal{C}_i$) in the codebook.

$$\arg\min_{i \in k} \|\boldsymbol{v} - \mathcal{C}_i\|^2, \forall \boldsymbol{v} \in \mathbf{W}' \tag{2}$$

VQ indicates the nearest centroid $\mathcal{C}_i$ that minimizes the Euclidean distance between the input vector $\boldsymbol{v}$ in the lookup table. The optimization problem aims to find the index i that results in the smallest distance between $\boldsymbol{v}$. Thus, each input vector is represented by the most similar centroids, thus minimizing total distortion.

Recent research has explored the use of VQ for model weight quantization (Chen et al., 2020; Cho et al., 2022; Stock et al., 2020, 2021). These studies

attempt to compress the embedding layer, the convolution layer, and the classification layer of neural networks using VQ. Figure 1 illustrates an example of applying VQ to compress model weights on a weight matrix. For a weight matrix $\mathbf{W}$ with dimensions $M \times N$, we reshape $\mathbf{W}$ into vectors of length $\boldsymbol{v}$ as $\mathbf{W}'$ (step ❶). The number of reshaped vectors should be $\frac{M \times N}{\boldsymbol{v}}$. Next, we employ k-means or other clustering algorithms to build a codebook (step ❷). The constructed codebook contains k centroid vectors, each with $\boldsymbol{v}$ dimensions. Applying the VQ algorithm directly often does not yield an acceptable accuracy. Typically, PTQ algorithms adjust the model index and centroid to enhance the accuracy of the quantized model (step ❸).

During model inference, each operator in the model first dequantizes the original weight matrix from the lookup table (codebook) by index and centroid. Unlike scalar quantization, VQ keeps the index and centroid in quantized weight. The equivalent compression ratio of VQ can be formulated as: total original model bits/(codebook bits + index bits). The equivalent quantization bitwidth is as: original bit width/compression ratio. For example, a 4096×4096 FP16 weight matrix with vectors of length $v = 8$ and 256 centroids, the compression ratio is $(16 \times 4096 \times 4096)/(8 \times 256 \times 16 + log_2(256) \times 4096 \times 4096/8) = 15.97$. The equivalent bitwidth is 1.002 bit.

2.3 Vector Quantization in LLMs

While VQ has been applied to weight quantization, the following significant challenges persist when quantizing LLM. We summarize the benefits and weaknesses of recent research (Egiazarian et al., 2024; Tseng et al., 2024; van Baalen et al., 2024) techniques in Table 1.

The number of parameters in LLMs is enormous, which requires quantizing the model using lightweight methods to avoid excessive resource consumption. AQLM (Egiazarian et al., 2024) utilizes gradient descent to train each layer of the VQ-quantized model and simultaneously trains across multiple layers using calibration data. It achieves effective compression through additive quantization and joint optimization of the codebook, which can achieve high accuracy. However, due to AQLM's use of backpropagation for model training, significant GPU hours and memory are required to achieve better accuracy, especially when dealing with LLMs with massive parameters.

Algorithm 1 VPTQ Algorithm

Input: $\mathbf{W} \leftarrow \mathbb{R}^{M \times N}$ ▷ Input weight matrix
Input: $\mathbf{H} \leftarrow \mathbb{R}^{N \times N}$ ▷ Hessian matrix
Output: $\hat{\mathbf{W}} \leftarrow \mathbb{R}^{M \times N}$ ▷ Quantized weight matrix
$\mathbf{E} \leftarrow \mathbb{R}^{M \times N}$ ▷ Initialize quantization errors
for $s = 0, B, 2B, \ldots$ **do** ▷ Column blocks
  for n $= s, s+1, \ldots, s+B-1$ **do**
  ▷ Quantize a single column n, fundamentally different from AQLM
    for $m = 0, V, 2V, \ldots, M$ **do**
  ▷ Parallel (Residual) Vector Quantization by function $Q(\boldsymbol{v})$ to vectors in the column n
      $\hat{\mathbf{W}}_{m:m+V,n} \leftarrow Q_V(\mathbf{W}_{m:m+V,n})$
    end for
    $\mathbf{E}_{:,n} \leftarrow (\mathbf{W}_{:,n} - \mathbf{W}'_{:,n})/(\mathbf{H}^{-1}_{n,n})$
      ▷ Update quantization error
    $\mathbf{W}_{:,n:s+B} \leftarrow \mathbf{W}_{:,n:s+B} - \mathbf{E}_{:,n}\mathbf{H}^{-1}_{n,n:s+B}$
      ▷ Merge quantization error to weights
  end for
  $\mathbf{W}_{:,s+B:} \leftarrow \mathbf{W}_{:,s+B:} - \mathbf{E}_{:,s:s+B}\mathbf{H}^{-1}_{s:s+B,s+B:}$
    ▷ Update all remaining weights
end for

GPTVQ (van Baalen et al., 2024) utilizes the Second-Order Optimization method to implement PTQ. However, GPTVQ accumulates quantization errors within vector quantization, leading to an inevitable increase in quantization errors as the vector length increases. It prevents the use of longer vectors and consequently limits the compression ratio.

QuIP# (Tseng et al., 2024) introduces an incoherence processing using the randomized Hadamard transform for the weight matrix before VQ. The processed weight matrix approximates a sub-Gaussian distribution, allowing for compression with a tiny codebook. However, incoherence processing requires a significant amount of computation, despite QuIP# being able to compress LLM to extremely low-bit with a low accuracy drop. It requires significantly more computation for inference compared to the original LLM, resulting in low inference throughput.

3 Vector Post-Training Quantization

3.1 VPTQ Algorithm

VPTQ leverages Second-Order Optimization and solves the optimization problem Eq.1 to achieve extreme low-bit quantization. Assume that a weight matrix is $\mathbf{W} \in \mathbb{R}^{M \times N}$, and a Hessian matrix collected from the current layer is $\mathbf{H} \in \mathbb{R}^{N \times N}$. We denote the q-th column of the weight matrix as $\hat{\mathbf{W}}_{:,q}$. The quantized column $\hat{\mathbf{W}}_{:,q}$ can be represented as the transpose of concatenated centroid vectors

$$\hat{\mathbf{W}}_{:,q} = (\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_{M/v})^T.$$

When the weight matrix of the model is large, we can first split the weight matrix into multiple groups. Each group has its own independent codebook. This method allows us to flexibly divide the weight matrix into several submatrices ($\hat{\mathbf{W}}_{:,q:q+(M/\text{group num})}$) equal to the group number. For clarity, we describe only one group in the following algorithm description.

Unlike GPTVQ, we quantize each column of the matrix independently, which we refer to as **Channel-Independent Second-Order Optimization**. It greatly simplifies the complexity of VQ in Second-Order Optimization. GPTVQ, on the other hand, quantizes $\boldsymbol{v}$ columns of the matrix ($\hat{\mathbf{W}}_{M,v}$) at once, leading to larger errors and more complex transformations for problem optimization.

We use the Lagrange Method to transform the optimization problem 1 into an unconstrained optimization problem. The Lagrangian function $L(\Delta\mathbf{W})$, and λ is the Lagrangian multiplier:

$$L(\Delta\mathbf{W}) = \Delta\mathbf{W}^T H(\mathbf{W})\Delta\mathbf{W} + \lambda\Delta\mathbf{W}$$

The dual function $g(\lambda)$ can be represented as:

$$g(\lambda) = -\mathbf{H}_{qq}^{-1}\lambda\lambda^T - \lambda(\hat{\mathbf{W}}_{:,q} - \mathbf{W}_{:,q})$$

Differentiating $g(\lambda)$ with respect to λ and setting it to 0,

$$g'(\lambda) = -\mathbf{H}_{qq}^{-1}\lambda - (\hat{\mathbf{W}}_{:,q} - \mathbf{W}_{:,q})^T = 0$$

we can find that when $\lambda^T = -\frac{(\hat{\mathbf{W}}_{:,q} - \mathbf{W}_{:,q})}{\mathbf{H}_{qq}^{-1}}$, the problem reaches an optimal solution.

By substituting λ^T into the optimization problem, we find that to minimize the error introduced by quantization, we need to minimize the impact on the Lagrangian function. Therefore, we can transform the quantization problem into minimizing:

$$\Delta L(\Delta\hat{\mathbf{W}}) = \frac{\sum \|\boldsymbol{v} - \mathcal{C}\|^2}{2\mathbf{H}_{qq}^{-1}}$$

We find that when quantizing a column vector each time, we only need to consider minimizing $\sum \|\boldsymbol{v} - \mathcal{C}\|^2$, which is to find the closest centroid in Euclidean Distance. It precisely aligns with the optimization of VQ. Moreover, since VPTQ quantizes the weight matrix column by column, $\mathbf{H}_{qq}^{-1}$ is constant when quantizing each column, so we do not need to consider Hessian when finding the centroid.

After quantizing a column of the weight matrix, we need to update the current quantization error to the unquantized part through:

$$\Delta\mathbf{W} = \frac{(\hat{\mathbf{W}}_{:,q} - \mathbf{W}_{:,q})}{\mathbf{H}_{qq}^{-1}}\mathbf{H}_{q,:}$$

It will transform current quantization errors to the following unquantized columns. Since GPTVQ quantizes $\boldsymbol{v}$ columns at the same time, quantization error can only spread to other unquantized columns when all $\boldsymbol{v}$ columns have been quantized. It will lead to more errors accumulating in the quantization, resulting in a decrease in model accuracy. We can have similar conclusions from Table 2. Algorithm 1 provides a detailed description of the steps to solve the optimization problem and quantize the weights according to the above analysis.

Distinguish VPTQ from GPTQ and GPTVQ: Compared with GPTQ, VPTQ employs vector representations in the quantization, which choose the vector closest to the original matrix to represent the original data. As VQ can use a larger codebook to store the quantized data, it covers a wider range of numerical distributions compared to the scalar quantization of GPTQ, thereby achieving better accuracy. Table 2 reveals that VPTQ significantly outperforms GPTQ under extremely low bit quantization.

Moreover, since GPTVQ quantizes multiple columns simultaneously, the propagation of quantization errors to unquantized columns is more challenging. Furthermore, the quantization errors in GPTVQ accumulate as the vector length increases, hindering GPTVQ from using longer vector lengths for weight compression (limited to only 1-4 bits). It significantly reduces the compression ratio of VQ. On the other hand, VPTQ is capable of compressing weights using longer vectors (> 8 bits) and representing data with a larger codebook. Table 2 shows the better accuracy achieved by VPTQ than GPTVQ.

3.2 Optimization in VPTQ

3.2.1 Hessian-Weighted Centroid Initialization

VTPQ algorithm requires the initialization of centroids in the codebooks prior to quantization. Properly initializing centroids can reduce quantization errors and improve model accuracy. A straightforward method is to perform K-means clustering on the weight matrix as centroids (Eq.2). However, it

does not consider the optimization object in Eq.1, leading to a significant accuracy drop (van Baalen et al., 2024; Egiazarian et al., 2024).

We can transform the optimization object by leveraging the cyclic property of matrix traces and the Hadamard product. We refine the optimization objective as:

$$\Delta\mathbf{W}^T\Delta\mathbf{W}\odot\mathbf{H}=\sum_{i=0}^{n-1}h_{i,i}\|\Delta\mathbf{W}_{:,i}\|^2 + \sum_{i=0}^{n-1}\sum_{j=0,j\neq i}^{n-1}h_{i,j}(\Delta\mathbf{W}_{:,i}\Delta\mathbf{W}_{:,j})$$

Due to the Hessian matrix being predominantly diagonal (Dong et al., 2020), it guides us to split the proxy error into two terms. The first term represents the dominant diagonal elements of the initial error matrix, which significantly impact the quantization error. The second term is the interaction of a single value in weight quantization with others.

Because the Hessian matrix is predominantly diagonal, we can prioritize optimizing the first term through centroid initialization. We can view the first term as a Weighted K-means Clustering problem (Cordeiro de Amorim and Mirkin, 2012; Kerdprasop et al., 2005; Liu et al., 2017). Since this problem is well-studied, we can directly solve it to achieve efficient and accurate centroid initialization.

3.2.2 Residual Vector Quantization

We enable Residual Vector Quantization (RVQ) (Barnes et al., 1996; Wei et al., 2014) in VPTQ. RVQ improves vector quantization (VQ) by breaking down the compression of a weight matrix into two (or more) stages. Each stage further compresses the residual error $v_{\text{res}} = v - Q(v)$ from the previous quantization stage:

$$Q(\boldsymbol{v}_{\text{res}})=\arg\min_i\|(\boldsymbol{v}-Q(\boldsymbol{v}))-\mathcal{C}_i^{\text{res}}\|^2$$

Unlike GPTVQ, VPTQ enables RVQ, which quantizes VQ quantization error using a separate lookup table for better representation and quantization. By partitioning the encoding into multiple stages and reducing quantization error, RVQ not only achieves superior compression efficiency but also ensures a balance between quantization error, the size of lookup tables, and the memory requirements for indices. During the decoding phase, VPTQ simply reads the centroids from these multiple lookup tables and combines them to reconstruct the original weight matrix.

Algorithm 2 End to End Quantization Algorithm

Require: original model, vector length v, centroid number k, hessian matrices $\mathbf{H}$
Ensure: quantized model
  for each layer l **do** ▷ Fully parallelized each layer on GPUs
    for each Linear operator **do**
      if outlier is enabled **then**
        Initialize outlier centroids $\mathcal{C}_{\text{outlier}}$
        $\mathbf{W}'_{\text{outlier}} \leftarrow \text{VPTQ}(\mathbf{W}_{\text{outlier}}, \mathcal{C}_{\text{outlier}})$
      end if
      Initialize centroids $\mathcal{C}$
      $w' \leftarrow \text{VPTQ}(\mathbf{W}, \mathcal{C})$
      if residual is enabled **then**
        Initialize residual centroids $\mathcal{C}_{\text{res}}$
        $\mathbf{W}'' \leftarrow \text{VPTQ}(\mathbf{W}-\mathbf{W}', \mathcal{C}_{\text{res}})$
      end if
    end for
    if finetune layer is enabled **then**
      Finetune layer l
    end if
  end for

3.2.3 Outlier Elimination

Recent studies on quantization in LLM have consistently observed a significant presence of outliers in activation (Xiao et al., 2023; Lin et al., 2023; Lee et al., 2024). Outliers, while small portions (~1% of the matrix), heavily affect the quantization error and simulate model accuracy. Outliers typically result in large values in the diagonal elements of the Hessian matrix. During centroid initialization in Sec.3.2.1, VPTQ already considers these Hessian diagonals as weights in K-means, allowing VPTQ to better quantize the error introduced by outliers.

$$Q(\boldsymbol{v}_{\text{outlier}})=\arg\min_i\|\boldsymbol{v}_{\text{outlier}}-\mathcal{C}_i^{\text{outlier}}\|^2$$

Furthermore, VPTQ flexibly partitions the weight matrix and uses a separate outlier lookup table to quantify matrix tiles most affected by outliers. It allows us to effectively trade off model accuracy and quantization overhead.

4 End to end Quantization Algorithm

In this section, we will detail the end-to-end model quantization algorithm (Algorithm 2). The algorithm takes the original model, vector length v, centroid number k, and Hessian matrices $\mathbf{H}$ as inputs. It starts by iterating over each layer l of the model. As each layer's quantization only relates to the current layer and the Hessian matrix, we can fully parallelize the quantization of each layer on GPUs.

In each layer, we first quantize the weight of each Linear Operator (matrix multiplication of input and weight). If we enable the outlier option, the algorithm first selects outlier columns

following Section 3.2 and initializes the outlier centroids $\mathcal{C}_{\text{outlier}}$. Then, VPTQ is applied to the outlier weights $\mathbf{W}_{\text{outlier}}$ using the outlier centroids, generating the quantized weights $\mathbf{W}'_{\text{outlier}}$. Next, the algorithm initializes the centroids $\mathcal{C}$ for the remaining columns and applies VPTQ to the weights $\mathbf{W}$ using these centroids to produce the quantized weights $\mathbf{W}'$. Lastly, if residual quantization is enabled, the algorithm initializes the residual centroids $\mathcal{C}_{\text{res}}$. It applies VPTQ to the residual error between the original weights and the quantized weights ($\mathbf{W} - \mathbf{W}'$), using the residual centroids. The quantized weight is updated as $\mathbf{W}''$.

After processing all the operators, the algorithm will fine-tune the layer l if we enable layer fine-tuning. The loss function is the Mean Squared Error (MSE) between the original and quantized computations. In layer-wise fine-tuning, we only update the normalization operator (e.g. RMSNorm) and centroid. These parameters only comprise a small fraction of the entire layer, and we can complete the fine-tuning quickly with limited memory. After each layer completes quantization and fine-tuning, we can further fine-tune the entire model as other PTQ methods used (Tseng et al., 2024; Chee et al., 2023; Egiazarian et al., 2024). Once the algorithm processes all layers, it outputs the quantized model. The end-to-end VPTQ algorithm quantizes all the weights in every Linear Operator in the model into an index and a codebook ($\mathcal{C}$). During model inference, we only need to dequantize the weight matrix by reading centroids from the codebook according to the index before executing the operator.

5 Experiments and Evaluations

5.1 Settings

Algorithm Baseline We focus on weight-only quantization. The detailed quantization parameters (such as vector length and codebook numbers) and fine-tuning parameters of our VPTQ are shown in Appendix B . Following (Frantar et al., 2023), our calibration data consists of 128 random segments of the C4 dataset (Raffel et al., 2020).

Models and Datasets We benchmark accuracy on LLaMA-2 (Touvron et al., 2023), LLaMA-3 families (Meta, 2024), and Mistral (Jiang et al., 2023). Following previous work (Frantar et al., 2023), we report perplexity on language modeling tasks (WikiText-2 (Merity et al., 2016), C4 (Raffel et al., 2020)). We also employ lm-eval-harness (Gao et al., 2021) to perform zero-shot evaluations on common sense QA benchmarks (PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC (Clark et al., 2018)). Detailed configuration is in Appendix A.

Baselines For LLaMA-2 and Mistral models, we compare VPTQ against GPTQ, GPTVQ, DB-LLM, QuIP#, and AQLM. To account for the different overheads resulting from varying codebook constructions, we provide results with comparable bit widths to facilitate a fair comparison. For LLaMA-3 models, we use the results of (Huang et al., 2024). However, due to alignment issues with the C4 dataset, we only show results for WikiText and QA tasks. Because LLaMA-3 models are new and running quantization ourselves is costly, we do not have results for QuIP# and AQLM.

5.2 Accuracy Evaluation

Results on LLaMA-2 model: We compare VPTQ with QuIP#, AQLM, GPTVQ, DB-LLM, and GPTQ on the LLaMA-2 model. First, we discuss the results of 2-bit quantization. As shown in Table 2, GPTQ, as a scalar quantization method, performs poorly with unusable accuracy. While DB-LLM and GPTVQ perform better, they still experience significant performance drops, with WikiText-2 perplexity increasing by 2. The significant accuracy drop in GPTVQ, despite being a vector quantization algorithm, is due to two factors: the use of shorter vector lengths, which introduces higher quantization loss, and the choice to update weights every v columns, which leads to cumulative errors. Therefore, we primarily focus on comparing VPTQ with the state-of-the-art QuIP# and AQLM which both choose longer vector lengths.

Table 2 includes the average scores for the five QA tasks mentioned in Section 5.1. VPTQ outperforms QuIP# and AQLM on 7B and 13B models. For the 7B model, VPTQ achieves a further reduction in WikiText-2 perplexity by 0.5 and 0.3 compared to the previous best results at 2-2.02 bits and 2.26-2.29 bits, respectively. In QA tasks, the VPTQ 2.26-bit model surpasses the AQLM 2.29-bit model with an average accuracy increase of 1%. For the 13B model, the VPTQ 2.02-bit model shows a slight improvement over QuIP#, and the 2.18-bit model outperforms AQLM in QA accuracy by 1.5%. On the LLaMA-2-70B model, we achieve similar perplexity (< 0.02) and comparable QA results($< 0.4\%$). The results for 3- and 4-bit quan-

Table 2: LLaMA-2 2bit Quantization Results. The "N/A" in the table stands for "not available," with further explanation provided in the Appendix A.1. [1] We use the naive Torch and Triton kernels, without optimizations like CUDA graphs, FlashAttention, or Torch compile. The inference performance for QuIP# and AQLM do not represent their performance with all optimizations enabled. QuIP# and AQLM can achieve high performance when all optimizations are enabled.

(a) 7B results

Method	bit	W2↓	C4↓	AvgQA↑	tok/s↑	mem(GB)↓	cost(h)↓
FP16	16	5.12	6.63	62.2	38.32	27.22	N/A
GPTQ	2.125	50.75	36.76	39.16	19.59	4.42	0.2
GPTVQ	2.25	6.71	9.9	56.14	N/A	N/A	1.5
DB-LLM	2.01	7.23	9.62	55.1	N/A	N/A	N/A
QuIP#[1]	2	6.19	8.16	**58.2**	4.4	2.25	N/A
AQLM [1]	2.02	6.64	8.56	56.5	19.4	**2.16**	N/A
AQLM [1]	2.29	6.29	8.11	58.6	19.6	2.4	11.07
VPTQ	2.02	**6.13**	**8.07**	**58.2**	**39.9**	2.28	2
	2.26	**5.95**	**7.87**	**59.4**	**35.7**	2.48	2.2

(b) 13B results

Method	bit	W2↓	C4↓	AvgQA↑	tok/s↑	mem(GB)↓	cost(h)↓
FP16	16	4.57	6.05	65.4	30.03	63.63	N/A
GPTQ	2.125	43.84	23.07	43.72	11.56	7.92	0.3
GPTVQ	2.25	5.72	8.43	61.56	N/A	N/A	3.7
DB-LLM	2.01	6.19	8.38	59.4	N/A	N/A	N/A
QuIP#[1]	2	5.35	7.2	62.0	3.5	**3.94**	N/A
AQLM [1]	1.97	5.65	7.51	60.6	N/A	N/A	N/A
AQLM [1]	2.18	5.41	7.2	61.6	16.5	4.14	22.7
VPTQ	2.02	**5.32**	**7.15**	**62.4**	**26.9**	4.03	3.2
	2.18	**5.28**	**7.04**	**63.1**	**18.5**	4.31	4

(c) 70B results

Method	bit	W2↓	C4↓	AvgQA↑	tok/s↑	mem(GB)↓	cost(h)↓
FP16	16	3.12	4.97	70.2	multi-gpu	N/A	N/A
GPTQ	2.125	NaN	NaN	59.18	2.38	37.63	2.83
GPTVQ	2.25	4.25	6.9	68.5	N/A	N/A	12
DB-LLM	2.01	4.64	6.77	65.8	N/A	N/A	N/A
QuIP#[1]	2	**3.91**	**5.71**	**69.0**	1.9	**18.36**	25
AQLM [1]	2.07	3.94	5.72	68.8	6.9	18.81	183
VPTQ	2.07	3.93	5.72	68.6	**9.7**	19.54	19
VPTQ	2.11	3.92	**5.71**	68.7	**9.7**	20.01	19

Table 3: LLaMA-3 and Mistra-7b 2,3,4-bit Quantization Results. The table shows LLaMA-3 Wikitext2 perplexity (context length 2048) and average zero-shot QA Accuracy, Mistral-7B Wikitext2, C4 perplexity (context length 8192) and average zero-shot QA accuracy. Detailed score for each task see Table 6 and Table 7.

	LLaMA-3 8B			LLaMA-3 70B				Mistral 7B			
	bit	W2↓	AvgQA↑	bit	W2↓	AvgQA↑		bit	W2↓	C4↓	AvgQA↑
FP16	16	6.14	68.6	16	2.9	75.3	FP16	16.0	4.77	5.71	68.6
QuIP	4	6.5	67.1	4	3.4	74.5	QuIP#	4.01	4.85	5.79	**68.7**
GPTQ	4	6.5	67.3	4	3.3	**74.9**	AQLM	4.02	4.85	5.79	68.0
VPTQ	4.03	**6.42**	**68.1**	4.05	**3.15**	74.7	GPTQ	4.125	4.83	5.74	68.4
QuIP	3	7.5	63.7	3	4.7	72.6	VPTQ	4.03	**4.81**	**5.72**	68.2
GPTQ	3	8.2	61.7	3	5.2	70.6	AQLM	3.0	5.07	5.97	**67.3**
VPTQ	3.03	**6.97**	**66.7**	3.01	**3.81**	**73.7**	VPTQ	3.03	**4.96**	**5.84**	**67.3**
QuIP	2	85.1	36.8	2	13	48.7	QuIP#	2.01	6.02	6.84	62.2
DB-LLM	2	13.6	51.7	N/A	N/A	N/A	AQLM	2.01	6.32	6.93	62.2
GPTQ	2	2.10E+02	36.2	2	11.9	45.4	GPTQ	2.125	1535	164	44.5
VPTQ	2.08	**9.29**	**60.2**	2.02	**5.6**	**70.9**	GPTVQ	2.25	8.99	18.6	57.7
VPTQ	2.24	**9.19**	**62.7**	2.07	**5.66**	**70.7**	VPTQ	2.04	**5.64**	**6.43**	**63.2**

tization shown in Table 5 are without end-to-end fine-tuning but are also comparable to AQLM and QuIP# which include end-to-end fine-tuning. The ablation study of quantization parameters is in Ap-

pendix C.

Results on LLaMA-3 and Mistral model: Table 3 presents VPTQ results on the LLaMA-3 model and Mistral-7b model. In all 2-, 3-, and 4-bit quantizations of LLaMA-3 models, we significantly outperform GPTQ, DB-LLM, and QuIP, whose accuracy drops to unusable levels. VPTQ ensures an accuracy drop of $< 8\%$ for the 8B model and $< 5\%$ for the 70B model. On the Mistral-7B model, our 2-bit performance surpasses both QuIP# and AQLM by 0.8% in QA accuracy. In 3-bit quantization, our perplexity is lower. At 4-bit, results are comparable overall. More detailed results are in Table 7. As bit width increases, the advantage of vector quantization diminishes, with GPTQ showing a similar WikiText-2 perplexity at 4-bit.

Inference throughput and quantization cost: In Table 2, the 'tok/s' column indicates the number of tokens generated per second during the decode phase of inference. VPTQ achieves a 2-9$\times$ speedup compared to QuIP# because QuIP# uses Hadamard Transform during decoding, which introduces $O(n^2)$ multiplications and additions, significantly slowing the inference throughput. Compared to AQLM, VPTQ uses a smaller codebook, resulting in a lower decoding overhead. Therefore, our inference throughput for the 7B and 13B models is 1.6-1.8$\times$ faster than AQLM. As the model size increases, our codebook size becomes comparable to theirs, leading to similar inference throughputs for the 70B model. The 'mem(GB)' column represents the GPU memory usage at runtime. The 'cost(h)' column represents the hours required for model quantization on $4\times$ 80GB A100 GPUs. We achieves comparable or even better results than AQLM in only 10.4-18.6% of quantization algorithm execution time.

6 Conclusion

In this paper, we propose Vector Post-Training Quantization (VPTQ), a novel approach to achieving extremely low-bit quantization of LLMs by Vector Quantization. Through the application of Second-Order Optimization, we have formulated the LLM Vector Quantization problem and directed the design of our quantization algorithm. By further refining the weights via Channel-Independent Second-Order Optimization, we have enabled a more granular VQ.

VPTQ also includes a brief and effective codebook initialization algorithm, which is achieved by decomposing the optimization problem. We have extended VPTQ to support residual and outlier quantization, which not only improves model accuracy but also further compresses the model size.

Our experimental results demonstrate the effectiveness and efficiency of VPTQ. The perplexity of quantized model is reduced by 0.01-0.34 on LLaMA-2, 0.38-0.68 on Mistral-7B, 4.41-7.34 on LLaMA-3 over SOTA at 2-bit, with an average accuracy improvement of 0.79-1.5% on LLaMA-2, 1% on Mistral-7B, 11-22% on LLaMA-3 on QA tasks. Furthermore, we achieved these results only using 10.4-18.6% of the execution time of the quantization algorithm, leading to a 1.6-1.8$\times$ increase in inference throughput compared to SOTA. These results underscore the potential of VPTQ as an efficient and powerful solution for the deployment and inference of LLMs, particularly in resource-constrained settings.

7 Limitations

Related research on PTQ (Egiazarian et al., 2024; Tseng et al., 2024; van Baalen et al., 2024) have adopted end-to-end model fine-tuning after the PTQ phase. Compared to other related works, VPTQ can better quantize the model in the PTQ, and it simplifies and reduces the cost and overhead of model fine-tuning.

Due to GPU resource constraints, we cannot fine-tune larger models (70B) for longer iterations and more tokens. It limits our experimental results, which can only achieve similar results to baselines in 70B models. It restricts the demonstration of VPTQ's advantages and potential on large models in this paper. We will strive for more GPU resources to fine-tune the VPTQ model for longer periods and with more tokens in the future, allowing for a fair comparison.

Additionally, since LLaMA-3 models are the latest released models, there is a lack of baselines from related works. It is difficult for us to fully demonstrate our performance improvements. We will continue to add more baselines in the future to highlight the advantages of VPTQ.

In this paper, we only use AI tools for grammar checking and code completion.

Acknowledgement

We thank James Hensman for his crucial insights into the error analysis related to Vector Quanti-

zation (VQ), and his comments on LLMs evaluation are invaluable to this research. We also thank QuIP# and AQLM for inspiring our paper and the authors for their guidance on implementation.

References

C.F. Barnes, S.A. Rizvi, and N.M. Nasrabadi. 1996. Advances in residual vector quantization: a review. *IEEE Transactions on Image Processing*, 5(2):226–262.

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

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees.

Ting Chen, Lala Li, and Yizhou Sun. 2020. Differentiable product quantization for end-to-end embedding compression. In *International Conference on Machine Learning*, pages 1617–1626. PMLR.

Minsik Cho, Keivan Alizadeh-Vahid, Saurabh Adya, and Mohammad Rastegari. 2022. DKM: differentiable k-means clustering layer for neural network compression. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Peter 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 preprint arXiv:1803.05457*.

Renato Cordeiro de Amorim and Boris Mirkin. 2012. Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering. *Pattern Recognition*, 45(3):1061–1075.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. HAWQ-V2: hessian aware trace-weighted quantization of neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*.

A. Gersho. 1979. Asymptotically optimal block quantization. *IEEE Transactions on Information Theory*, 25(4):373–380.

Babak Hassibi and David G. Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 164–171. Morgan Kaufmann.

Babak Hassibi, David G. Stork, and Gregory J. Wolff. 1993. Optimal brain surgeon and general network pruning. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, pages 293–299. IEEE.

Wei Huang, Xudong Ma, Haotong Qin, Xingyu Zheng, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. 2024. How good are low-bit quantized llama3 models? an empirical study.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Kittisak Kerdprasop, Nittaya Kerdprasop, and Pairote Sattayatham. 2005. Weighted k-means for density-biased clustering. In *International conference on data warehousing and knowledge discovery*, pages 488–497. Springer.

Yann LeCun, John S. Denker, and Sara A. Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. OWQ: outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 13355–13364. AAAI Press.

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

Hongfu Liu, Junjie Wu, Tongliang Liu, Dacheng Tao, and Yun Fu. 2017. Spectral ensemble clustering via weighted k-means: Theoretical and practical evidence. *IEEE Transactions on Knowledge and Data Engineering*, 29(5):1129–1143.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024. The era of 1-bit llms: All large language models are in 1.58 bits. *CoRR*, abs/2402.17764.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

AI Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. *Meta AI*.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

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

Sidak Pal Singh and Dan Alistarh. 2020. Woodfisher: Efficient second-order approximation for neural network compression. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. 2021. Training with quantization noise for extreme model compression. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. 2020. And the bit goes down: Revisiting the quantization of neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 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.

Mart van Baalen, Andrey Kuzmin, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. Gptvq: The blessing of dimensionality for llm quantization.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. Bitnet: Scaling 1-bit transformers for large language models. *CoRR*, abs/2310.11453.

Benchang Wei, Tao Guan, and Junqing Yu. 2014. Projected residual vector quantization for ann search. *IEEE MultiMedia*, 21(3):41–51.

Guangxuan Xiao, Ji Lin, Mickaël 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.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

A Appendix: All Experiments Results

A.1 Supplementary Explanation for Main Results Table 2

Table 2 shows our main results. Here we provide an explanation for the 'N/A' entries relative to other works.

DB-LLM Since they did not open source their code, we use the AvgQA results from their paper. However, this number does not align with our FP16 results.

GPTQ We reproduce the 2-bit results using the official GPTQ repository. As GPTQ quantizes each layer in sequential order, the 'cost(h)' represents the time taken to quantize on a single A100 GPU.

GPTVQ They do not release their 2-bit quantized model. We reproduce Llama-2, LLama-3 7B and 13B, Mistral 7b 2-bit results using their released GPTVQ code, which only supports single-GPU execution. Therefore, the quantization cost reflects the execution time for quantization on a single A100 GPU. Due to the lack of specific logic for loading their quantizers in the released code, we were unable to measure the throughput and runtime memory.

AQLM Their 1.97-bit LLaMA-2 13b model has not been open-sourced, so we are unable to measure its inference throughput and runtime memory.

QuIP# Due to recent changes in the libraries they rely on, the quantization cost is not measured. The quantization time for the 70B model is estimated based on their original paper.

A.2 All Experimental Results

In this section, we present all our experimental results, including the perplexity of the quantized model on different context lengths in two datasets, Wikitext2 and C4, and the accuracy on five Commonsense QA tasks (abbreviated as AE for Arc_easy, AC for Arc_challenge, HE for Hellaswag, QA for PIQA, and WI for Winogrande). Table 4 displays all results of LLaMA-2 at 2-bit quantization. Table 5 presents results of LLaMA-2 at 3 and 4 bits quantization. Table 6 displays all results of Llama3 at 2, 3, and 4-bit quantization. Table 7 shows all results of Mistral 7b at 2, 3, and 4-bit quantization.

B Quantitative Analysis of Quantization Parameter Settings

Quantization configuration The quantization parameters of all VPTQ 2bit models are shown in Table 8.

Layer-wise fine-tuning parameters Layer-wise finetuning trains centroids and layer norm using the input and output of each layer when entering 128 samples of C4 training sets into the full precision model. We train each layer for 100 iterations. Table 9 shows the learning rate and batch size used for each model.

C Ablation Study

Table 10 shows results from LLaMA-2 13b on Wikitext2 and C4 (sequence length = 4096) under different quantization parameters. The impact of techniques such as vector length, channel-independent optimization, residual vector quantization, outlier elimination, layer-wise fine-tuning, and end-to-end fine-tuning on quantization results will be discussed.

C.1 Parameter Description

When performing N% outlier elimination, N% of outliers will be quantized using a codebook with a vector length of v_0 and k_0 centroids. For the remaining (100-N)% parameters, the vector length is v_1. k_1 represents the number of centroids in the first codebook, while k_2 represents the number of centroids in the second codebook for residual vector quantization. $k_2 = -1$ indicates no residual vector quantization.

C.2 Vector Length and Residual Vector Quantization

Compression Ratio Calculation The average bitwidth per element of the index matrix obtained through vector quantization is:

$$\text{Average index bitwidth} = \frac{\log_2(k_1)}{v_1} + \frac{\log_2(k_2)}{v_1}$$

The compression ratio is calculated by:

$$\text{Compression ratio} = \frac{\text{Total original model bits}}{\text{Codebook bits} + \text{Index bits}}$$

For an original linear weight matrix with M parameters,

$$\text{Codebook bits} = (v_0 \times k_0 + v_1 \times (k_1 + k_2)) \times 16$$

$$\text{Index bits} = M \times N\% \times \log_2\left(\frac{k_0}{v_0}\right) + M \times (100 - N)\% \times \left[\frac{\log_2(k_1)}{v_1} + \frac{\log_2(k_2)}{v_1}\right]$$

The total bitwidth in the table is calculated per transformer block, which for LLaMA-2 includes 4 attention linear and 3 FFN linear layers.

Impact of Vector Length First, we discuss the impact of vector length on accuracy. In Table 10 rows #2, #3, #4, and #6 show results for $v_1 = 2, 4, 6, 8$, keeping the average index bit at 2 (i.e., $log_2(k_1/v_1) = 2$). As v_1 increases, the perplexity on Wikitext2 and C4 decreases, but the codebook size also increases exponentially. For $v_1 = 8$ and $k_1 = 65536$, the codebook overhead introduces an additional 0.19 bits. Then, we evaluate the model inference throughput in Table 11. Since we employ weight-only quantization, the main additional overhead of quantized model inference comes from the lookup table for model weights. Table 11 shows models with 2 bits on various throughputs. As the vector length increases (from 2 to 6), the granularity of memory access for reading the lookup table in dequantization increases, which allows memory access to match the GPU's cache line (128 bytes @ L1). This reduces memory access transactions and decreases cache misses. As the vector length further increases (from

Table 4: LLaMA-2 2bit Quantization Results, [1] We use the naive Torch and Triton kernels for inference performance evaluation, without optimizations like CUDA graphs, FlashAttention, or Torch compile. The inference performance for QuIP# and AQLM do not represent their performance with all optimizations enabled. QuIP# and AQLM can achieve high performance when all optimizations are enabled.

7B	bit	W2	C4	AC	AE	HE	QA	WI	tok/s	mem(GB)	cost(h)
FP16	16	5.12	6.63	39.93	69.28	56.69	78.35	66.93	38.32	27.22	N/A
GPTQ	2.125	50.75	36.76	20.9	34.9	30.5	57.2	52.3	19.59	4.42	0.2
GPTVQ	2.25	6.71	9.9	31.2	**66.3**	46.4	72.4	64.4	N/A	N/A	1.5
DB-LLM	2.01	7.23	9.62	33.53	45.2	61.98	73.18	61.72	N/A	N/A	N/A
QuIP#[1]	2	6.19	8.16	34.6	64.6	51.91	75.1	**64.9**	4.4	2.25	N/A
AQLM[1]	2.02	6.64	8.56	33.28	61.87	49.49	73.56	64.17	19.4	**2.16**	N/A
	2.29	6.29	8.11	34.9	66.5	50.88	74.92	65.67	19.6	2.4	11.07
VPTQ	2.02	**6.13**	**8.07**	**35.24**	63.8	52.08	**75.19**	64.33	**39.9**	2.28	2
	2.26	**5.95**	**7.87**	**36.43**	64.9	**52.87**	**76.17**	**66.46**	**35.7**	2.48	2.2
13b	bit	W2	C4	AC	AE	HE	QA	WI	tok/s	mem(GB)	cost(h)
FP16	16	4.57	6.05	45.56	73.23	59.71	78.73	69.69	30.03	63.63	N/A
GPTQ	2.125	43.84	23.07	23.3	43.3	36	61.3	54.7	11.56	7.92	0.3
GPTVQ	2.25	5.72	8.43	38.7	**73.6**	51.6	75.4	**68.5**	N/A	N/A	3.7
DB-LLM	2.01	6.19	8.38	38.14	51.64	68.04	75.14	64.09	N/A	N/A	N/A
QuIP#[1]	2	5.35	7.2	39.5	69.3	56.01	**77.3**	67.7	3.5	**3.94**	N/A
AQLM[1]	1.97	5.65	7.51	37.8	69.78	53.74	76.22	65.43	N/A	N/A	N/A
	2.18	5.41	7.2	39.42	69.15	54.68	76.22	68.43	16.5	4.14	22.7
VPTQ	2.02	**5.32**	**7.15**	**40.02**	71.55	**56.18**	77.26	66.85	**26.9**	4.03	3.2
	2.18	**5.28**	**7.04**	**40.96**	71.8	**56.89**	**77.48**	68.43	**18.5**	4.31	4
70b	bit	W2	C4	AC	AE	HE	QA	WI	tok/s	mem(GB)	cost(h)
FP16	16	3.12	4.97	51.11	77.74	63.97	81.12	77.11	multi-gpu	N/A	N/A
GPTQ	2.125	NaN	NaN	35.8	67	51.8	74.6	66.7	2.38	37.63	2.83
GPTVQ	2.25	4.25	6.9	49.4	**80.47**	58.26	79.4	75.2	N/A	N/A	12
DB-LLM	2.01	4.64	6.77	44.45	55.93	76.16	79.27	73.32	N/A	N/A	N/A
QuIP#[1]	2	**3.91**	**5.71**	**48.7**	77.3	**62.49**	80.3	75.9	1.9	**18.36**	25
AQLM[1]	2.07	3.94	5.72	47.93	77.68	61.79	**80.43**	**75.93**	6.9	18.81	183
VPTQ	2.07	3.93	5.72	47.7	77.1	**62.98**	80.3	74.98	**9.7**	19.54	19
	2.11	3.92	**5.71**	48.29	77.77	62.51	79.82	75.14	**9.7**	20.01	19

8 to 12) along with the size and levels of the codebook, the codebook size further increases, which results in the codebook not fitting in the L1 cache, thereby reducing the model's inference speed. Additionally, we find that a reasonable setting (e.g., $v = 6$, $k = 4096$) can achieve throughput similar to the original model for the quantized model, demonstrating the efficiency of the VPTQ design.

Residual Vector Quantization Without any fine-tuning, rows #4 and #7 show similar perplexities for $v_1 = 6, k_1 = 4096$ and $v_1 = 12, k_1 = k_2 = 4096$, with the latter even higher. However, after layer-wise fine-tuning, comparing rows #11 and #13, residual vector quantization (RVQ) reduces the perplexity by 0.3 compared to vector quantization (VQ) due to the increased number of finetunable centroids, showing significant improvement.

C.3 Channel-Independent Optimization

Row #4 with channel-independent optimization shows a perplexity decrease of 1 compared to row #5 without it, indicating that channel-independent second-order optimization effectively mitigates quantization error accumulation.

C.4 Outlier Elimination

Rows #4, #8, #9, and #10 represent the results for eliminating 0%, 1%, 2%, and 5% outliers, respectively. We used a codebook with $v_0 = 4$ and $k_0 = 4096$ to quantize N% of outliers, achieving an effective average index bit of 3 bits, while other parameters were 2 bits. Higher N% means more parameters are quantized with 3 bits, leading to a larger total bitwidth and lower perplexity.

C.5 Fine-tuning

Rows #4, #11, and #12 show results without any fine-tuning, with layer-wise fine-tuning, and with end-to-end fine-tuning, respectively. Adding fine-tuning reduced the perplexity on Wikitext2 from 6.29 to 6.07 and further to 5.32.

C.6 Group Number

Rows #14, #15, #16, and #17 show the quantization results when 99% of parameters are divided into 1, 2, 4, and 8 groups, respectively. Each group has its own independent codebook. When divided into 1,

Table 5: LLaMA-2 3, 4-bit Quantization Results. The table shows Witext2, C4 perplexity (context length 2048 and 4096) and zeroshot QA Accuracy.

7B	bit	W2(2k)	C4(2k)	W2(4k)	C4(4k)	AC	AE	HE	QA	WI
GPTQ	4	—	—	5.49	7.2	36.8	66.2	55.4	76.6	68.2
GPTVQ	4.125	5.68	7.25	5.27	6.88	**42.83**	**75.17**	**56.41**	77.37	**69.61**
QuIP#	4	**5.56**	**7.07**	**5.19**	**6.75**	40.5	69.1	—	**78.4**	67.6
AQLM	4.04	—	—	5.21	**6.75**	41.0	70.2	56.0	78.2	67.3
VPTQ	4.01	5.64	7.13	5.26	6.8	39.7	69.0	56.0	78.1	67.1
GPTQ	3	—	—	8.06	10.61	31.1	58.5	45.2	71.5	59.2
GPTVQ	3.125	5.83	7.51	5.44	7.24	**39.93**	**74.07**	54.21	76.17	**69.06**
QuIP#	3	**5.79**	**7.32**	**5.41**	**7.04**	39.2	68.4	—	**77.3**	66.5
AQLM	3.04	—	—	5.46	7.08	38.4	68.1	54.1	76.9	66.9
VPTQ	3.02	5.82	7.33	5.43	**7.04**	39.3	69.1	**54.9**	**77.3**	68.0
13B	bit	W2(2k)	C4(2k)	W2(4k)	C4(4k)	AC	AE	HE	QA	WI
GPTQ	4	—	—	4.78	6.34	42.49	70.45	58.67	77.75	**70.01**
GPTVQ	4.125	5.68	7.25	5.27	6.88	42.83	**75.17**	56.41	77.37	69.61
QuIP#	4	**4.95**	**6.54**	**4.63**	**6.13**	**45.50**	73.90	—	**78.90**	69.90
AQLM	3.94	—	—	4.65	6.14	44.80	73.32	59.27	78.35	69.85
VPTQ	4.02	4.96	**6.54**	4.64	**6.13**	44.37	73.19	**59.37**	77.75	69.77
GPTQ	3	—	—	5.85	7.86	38.48	65.66	53.47	76.50	63.93
GPTVQ	3.125	5.11	6.83	4.8	6.47	**44.45**	**77.23**	58.18	77.8	**71.98**
QuIP#	3	**5.1**	6.72	**4.78**	6.35	44.00	72.50	—	**78.40**	69.10
AQLM	3.03	—	—	4.82	6.37	42.58	70.88	58.30	77.26	68.43
VPTQ	3.03	5.12	**6.7**	4.79	**6.32**	42.32	73.99	**58.42**	77.64	68.67
70B	bit	W2(2k)	C4(2k)	W2(4k)	C4(4k)	AC	AE	HE	QA	WI
GPTQ	4	—	—	3.35	5.15	49.15	76.81	63.47	81.23	75.61
GPTVQ	4.125	5.32	—	—	—	—	—	—	—	—
QuIP#	4	**3.38**	**5.56**	**3.18**	**5.02**	50.6	78.1	—	81.4	**77.1**
AQLM	4.14	—	—	3.19	5.03	**50.68**	77.31	63.69	**81.5**	76.48
VPTQ	4.01	3.39	5.57	3.19	**5.02**	49.57	**78.16**	**63.71**	81.18	76.4
GPTQ	3	—	—	4.4	6.26	44.11	72.73	60	78.4	71.82
GPTVQ	3.125	5.51	—	—	—	—	—	—	—	—
QuIP#	3	3.56	**5.67**	3.35	**5.15**	**50.9**	**77.7**	—	**81.4**	76.4
AQLM	3.01	—	—	3.36	5.17	50	77.61	63.23	81.28	77.19
VPTQ	3.01	**3.55**	**5.67**	**3.34**	**5.15**	48.89	77.06	**63.52**	80.9	**77.51**

Table 6: LLaMA-3 Wikitext2 perplexity (context length 2048) and zeroshot QA Accuracy.

	LLaMA-3 8B							LLaMA-3 70B						
	bit	W2↓	AC↑	AE↑	HE↑	QA↑	WI↑	bit	W2↓	AC↑	AE↑	HE↑	QA↑	WI↑
FP16	16	6.14	50.3	80.1	60.2	79.6	73.1	16	2.9	60.1	87.0	66.3	82.4	80.8
QuIP	4	6.5	47.4	78.2	58.6	78.2	73.2	4	3.4	58.7	86.0	65.7	82.5	79.7
GPTQ	4	6.5	47.7	78.8	59.0	78.4	72.6	4	3.3	58.4	**86.3**	66.1	**82.9**	**80.7**
VPTQ	4.03	**6.42**	**49.1**	**78.8**	**59.3**	**78.7**	**74.8**	4.05	**3.15**	**59.0**	86.1	**66.2**	82.4	79.8
QuIP	3	7.5	41.0	72.9	55.4	76.8	72.5	3	4.7	54.9	83.3	63.9	**82.3**	78.4
GPTQ	3	8.2	37.7	70.5	54.3	74.9	71.1	3	5.2	52.1	79.6	63.5	80.6	77.1
VPTQ	3.03	**6.97**	**45.8**	**77.5**	**58.4**	**78.2**	**73.4**	3.01	**3.81**	**57.3**	**84.7**	**65.5**	81.7	**79.2**
QuIP	2	85.1	21.3	29.0	29.2	52.9	51.7	2	13	26.5	48.9	40.9	65.3	61.7
DB-LLM	2	13.6	28.2	59.1	42.1	68.9	60.4	N/A	N/A	N/A	N/A	N/A	N/A	N/A
GPTQ	2	2.10E+02	19.9	28.8	27.7	53.9	50.5	2	11.9	24.6	38.9	41.0	62.7	59.9
VPTQ	2.08	**9.29**	**36.9**	**71.0**	**52.2**	**75.1**	**65.9**	2.02	**5.6**	**52.5**	**81.8**	**61.7**	**80.4**	**77.9**
VPTQ	2.24	**9.19**	**42.6**	**73.2**	**53.1**	**75.4**	**69.1**	2.07	**5.66**	**54.2**	**83.6**	**61.8**	**80.1**	**74.0**

Table 7: Mistral-7B-v0.1 Wikitext2, C4 perplexity (context length 2048 and 8192) and zeroshot QA Accuracy

	Mistral 7b								
	bit	W2(2k)	W2(8k)	C4(8k)	AC	AE	HE	QA	WI
FP16	16	5.25	4.77	5.71	48.89	78.87	61.12	80.3	73.88
GPTVQ	4.125	5.38	4.87	6.13	**50**	**80.43**	60.36	79.65	73.4
QuIP#	4	—	4.85	5.79	49.4	78.96	60.62	**80.41**	73.95
AQLM	4.02	—	4.85	5.79	48.21	77.86	60.27	79.71	73.8
GPTQ	4.125	**5.36**	4.83	5.74	49.57	79.5	60.38	79.54	72.85
VPTQ	4.03	**5.36**	**4.81**	**5.72**	48.12	77.82	**60.61**	80.14	**74.19**
GPTVQ	3.125	6.42	6.8	13.28	40.78	75.67	54.18	77.42	67.4
AQLM	3.04	—	5.07	5.97	46.67	77.61	59.31	**80.14**	**72.69**
GPTQ	3.125	6.02	5.88	6.86	**47.35**	77.86	58.84	79.82	71.74
VPTQ	3.03	**5.53**	**4.96**	**5.84**	46.67	**77.95**	**59.91**	79.49	72.45
QuIP#	2	—	6.02	6.84	39.76	72.14	52.95	76.71	**69.3**
AQLM	2.01	—	6.32	6.93	40.44	**73.65**	52.13	76.01	68.75
GPTVQ	2.25	8.2	8.99	18.6	37.37	71	45.43	70.18	64.33
GPTQ	2.125	280	1535	164	24.49	44.91	36.56	63.33	52.96
VPTQ	2.04	**6.32**	**5.64**	**6.43**	**41.13**	72.22	**56.1**	**77.91**	68.67

Table 8: Parameters for 2-bit Quantization of Llama and Mistral Models. v represents the vector length, k denotes the codebook size, $k1$ and $k2$ correspond to the two codebooks, and $group\quad num$ indicates the number of groups into which PQ (Product Quantization) is divided.

	bit	Outlier			Other			
		N%	v	k	v	k1	k2	group num
LLaMA2-7b	2.02	0	-	-	6	4096	-	1
	2.26	1	4	8192	12	4096	4096	4
LLaMA2-13b	2.02	0	-	-	6	4096	-	1
	2.18	2	4	8192	12	4096	4096	4
LLaMA2-70b	2.07	1	4	8192	12	4096	4096	4
	2.11	1	4	8192	12	4096	4096	8
LLaMA3-8b	2.08	1	4	4096	12	4096	4096	1
	2.24	1	4	8192	6	4096	-	16
LLaMA3-70b	2.02	0	-	-	12	4096	4096	1
	2.07	1	4	4096	6	4096	-	16

Table 9: Layer-wise finetuning parameters on 8xH100

model	finetune lr	batchsize
LLaMA-2-7B	1×10^{-4}	32
LLaMA-2-13B	1×10^{-4}	32
LLaMA-2-70B	1×10^{-5}	16
LLaMA-3-8B	1×10^{-5}	16
LLaMA-3-70B	5×10^{-6}	8
Mistral-7B	5×10^{-6}	16

2, and 4 groups, the perplexity on Wikitext2 does not change much, likely because the distribution of the remaining parameters (after removing 1% outliers) is relatively uniform. This is likely because the distributions of different groups overlap after grouping, so the benefit of increasing the group number is not significant.

C.7 Higher Bitwidth

Rows #18 and #19 represent the results for 3-bit and 4-bit quantization, respectively. Compared to the FP16 results in row #1, 4-bit vector quantization incurs almost no loss.

D Inference Evaluation

D.1 Throughput Measurement Process

We follow the throughput measurement method used in AQLM (Egiazarian et al., 2024). During the prompt phase, we provide 1 token and then have the model generate 256 tokens, calculating the generation time for each output token to determine the throughput in tokens per second (tok/s).

Table 10: Ablation Study on Different Quantization Techniques for LLaMA-2 13B

	bit	channel independent	Finetune		outlier			other					
			layer wise	e2e	N%	v0	k0	v1	k1	k2	group num	W2(↓)	C4(↓)
#1	FP16	-	-	-	-	-	-	-	-	-	-	4.57	6.05
#2	2	Yes	No	No	0	-	-	2	16	-1	1	14800	13337
#3	2.01	Yes	No	No	0	-	-	4	256	-1	1	7.21	9.78
#4	2.02	Yes	No	No	0	-	-	6	4096	-1	1	6.29	8.29
#5	2.02	No	No	No	0	-	-	6	4096	-1	1	7.25	9.8
#6	2.19	Yes	No	No	0	-	-	8	65536	-1	1	5.8	7.68
#7	2.04	Yes	No	No	0	-	-	12	4096	4096	1	6.32	8.29
#8	2.03	Yes	No	No	1	4	4096	6	4096	-1	1	6.16	8.08
#9	2.04	Yes	No	No	2	4	4096	6	4096	-1	1	6.08	8.12
#10	2.07	Yes	No	No	5	4	4096	6	4096	-1	1	6.02	7.96
#11	2.02	Yes	Yes	No	0	-	-	6	4096	-1	1	6.07	7.64
#12	2.02	Yes	Yes	Yes	0	-	-	6	4096	-1	1	5.32	7.15
#13	2.04	Yes	Yes	No	0	-	-	12	4096	4096	1	5.71	7.52
#14	2.06	Yes	Yes	No	1	4	4096	12	4096	4096	1	5.63	7.45
#15	2.09	Yes	Yes	No	1	4	4096	12	4096	4096	2	5.63	7.41
#16	2.17	Yes	Yes	No	1	4	4096	12	4096	4096	4	5.63	7.38
#17	2.3	Yes	Yes	No	1	4	4096	12	4096	4096	8	5.55	7.38
#18	3.01	Yes	Yes	No	0	-	-	4	4096	-1	1	4.82	6.37
#19	4.02	Yes	Yes	No	0	-	-	6	4096	4096	1	4.64	6.13

Table 11: Ablation of Vector Length on Inference Throughput and Peak Memory Usage

	v1	k1	k2	group num	tok/s	mem(GB)
FP16	-	-	-	-	30.03	63.63
2	2	16	-1	1	18.85	4.17
2.01	4	256	-1	1	17.06	4
2.02	6	4096	-1	1	32.09	4.02
2.19	8	65536	-1	1	30.64	4.46
2.04	12	4096	4096	1	21.34	4.06

D.2 Our Dequantization Implementation

Our dequantization implementation is divided into two phases. In the first phase, which handles prompts with relatively long sequences, we restore the quantized weights (index and centroid, etc.) to FP16 and then call ‘torch.matmul‘. In the second phase, during decoding, we fuse the dequantization and GEMV operations into QGemv, eliminating the repetitive reading and writing of FP16 weights.