

GCPT: Gradient-aware Clustering Method for Efficient Post-Training Quantization in Large Neural Networks

Chuyi Dai¹, Chen Ye¹, Zeyu Li², Jun Tao¹, Wei Zhang², Grace Li Zhang³, Xin Li⁴

¹State Key Laboratory of Integrated Chips and Systems, College of Integrated Circuits & Micro-Nano Electronics School of Integrated Circuits, Fudan University, Shanghai, China

²Electronics and Computer Engineering Department, The Hong Kong University of Science and Technology, Hong Kong, China

³Hardware for AI Group, TU Darmstadt, Darmstadt, Germany

⁴Digital Innovation Research Center, Duke Kunshan University, Kunshan, China

Abstract—Large-scale neural network models have achieved outstanding performance across diverse tasks, but often come with expensive computational costs. In this paper, we propose a gradient-aware clustering method for post-training quantization (GCPT) in order to effectively reduce the computational overhead. Our key idea is to cluster the weights of linear layers based on their gradient-aware contributions to the overall loss function. Afterwards, all weights are replaced by a small set of cluster centroids to minimize the variation of the loss function due to quantization. To further accelerate inference, the inputs associated with those weights in the same cluster are first aggregated and then the sum is multiplied with the shared centroid, thereby reducing the number of scalar multiplications. Experiments on three large-scale models demonstrate that the proposed GCPT method achieves up to 93.8% computational cost reduction, while preserving memory usage and inference accuracy, compared to other state-of-the-art methods.

Keywords—Gradient, clustering, post-training quantization, neural networks

I. INTRODUCTION

In recent years, large-scale neural networks (NNs) such as ChatGPT [1] and LLaMA [2] have been widely applied to a broad range of tasks across language, vision, and multimodal domains. However, their massive computational and storage demands restrict their deployment on resource-limited edge devices. To reduce these overheads, numerous compression techniques have been explored, including pruning [3], quantization [4], low-rank decomposition [5], etc. Among them, quantization has become the most widely used method due to its significant reduction in storage and computational costs, while having minimal impact on model performance [6].

Quantization can be divided into quantization-aware training (QAT) (e.g., LLM-QAT [7]), and post-training quantization (PTQ) [8]. QAT integrates quantization into training to preserve accuracy, but increases retraining time and GPU memory consumption. On the contrary, PTQ does not require retraining, making it more attractive for large-scale model compression.

PTQ can be performed using either general or clustering-based quantization methods. General quantization typically maps model parameters or activations from high precision (e.g., FP32) to lower precision (e.g., INT8 or lower) using a

predefined linear mapping. This process often results in uniformly distributed quantized values, potentially leading to larger errors for data with non-uniform distributions. To reduce errors, mixed-precision methods (e.g., SpQR [9] and Oltron [10]) quantize most weights to a fixed bit-width while retaining a small subset of outlier weights in higher precision. This approach enhances accuracy but increases the complexity of the inference process. Another type of approaches to improving accuracy, such as GPTQ [11] and APTQ [12], leverages Hessian information to minimize reconstruction errors at the module level (e.g., linear layers or self-attention blocks). However, when multiple modules are quantized, the quantization errors will accumulate and propagate nonlinearly across layers, amplifying their effects on the final model output and ultimately degrading overall performance. Moreover, to maintain accuracy, GPTQ and APTQ still rely on high-precision matrix multiplication in inference. Thus, low-bit weights need to be dequantized back to floating-point values, increasing computational and storage overhead.

Clustering-based quantization, on the other hand, adopts a data-driven approach to derive optimal quantized values. Its fundamental idea is to use a clustering algorithm (e.g., K-means [13], GOBO [14]) to group the original values (e.g., the weights of a neural network layer) into K clusters. Each original value is then replaced with the centroid of the cluster it belongs to. Different from conventional quantization methods, the quantized values (i.e., cluster centroids) are often non-uniformly distributed, allowing this method to better preserve the original data distribution and, as a result, often achieve higher model accuracy. However, existing clustering-based quantization methods (e.g., K-means) rely only on numerical similarity of the weights when forming clusters. They overlook the practical contribution of each weight to the loss function. As a result, their performance may degrade under ultra-low-bit quantization, as they fail to preserve critical task-specific information [11].

In this paper, we propose Gradient-Aware Clustering for Post-Training Quantization (GCPT), a method for efficient quantization of large-scale NNs. GCPT utilizes loss gradients to guide weight clustering, minimizing accuracy degradation during quantization. The method proceeds in two alternating steps: (1) group all weights into clusters based on their gradient-weighted contributions to the overall loss, and (2) update the cluster centroids via minimizing the quantization-induced loss error. This iterative process continues until convergence. After quantization, all floating-point weights are replaced with a

This research is supported partly by National Key R&D Program of China 2023YFB4404400, 2023YFB4404402, partly by National Natural Science Foundation of China (NSFC) research projects 62434003.

Jun Tao and Xin Li are the corresponding authors.

compact set of cluster centroids and their corresponding indices. To further accelerate inference, we introduce the Accumulation-Before-Multiplication (ABM) scheme. This approach aggregates the inputs corresponding to weights within the same cluster prior to multiplication, allowing the shared cluster centroid to be applied only once to the accumulated sum. By doing so, ABM effectively reduces the required number of scalar multiplications, enhancing computational efficiency. Experiments on three large-scale models demonstrate that GCPT reduces computational cost by up to 93.8% compared with state-of-the-art quantization methods, while maintaining negligible accuracy loss and similar low memory overhead.

The remainder of this paper is organized as follows. In Section II, we describe the proposed GCPT approach. Some important implementation details are further discussed in Section III. Next, in Section IV, we demonstrate the efficacy of our proposed approach using several large scale models. Finally, we conclude the paper in Section V.

II. PROPOSED METHOD

In this section, we propose the weight clustering algorithm guided by the loss gradients. Next, we introduce an efficient ABM inference scheme with low computational overhead.

A. Gradient-aware Clustering Algorithm

Given a pre-trained neural network model, the weight matrix \mathbf{W} of a linear layer is typically represented as a two-dimensional array with dimensions $N_o \times N_i$, where N_o is the number of outputs $\mathbf{y} = [y_0, y_1, \dots, y_{N_o-1}]^T$ and N_i is the number of inputs $\mathbf{x} = [x_0, x_1, \dots, x_{N_i-1}]^T$. This matrix \mathbf{W} can be flattened into a one-dimensional weight vector $\mathbf{w} = [w_0, w_1, \dots, w_{N-1}]^T$, where $N = N_o \times N_i$. When a small perturbation Δw_i is applied to each weight w_i ($i \in \{0, 1, \dots, N-1\}$), the perturbed loss function $L(w_i + \Delta w_i)$ can be approximated using the first-order Taylor expansion around w_i :

$$L(w_i + \Delta w_i) \approx L(w_i) + \frac{\partial L(w_i)}{\partial w_i} \cdot \Delta w_i = g_i \cdot \Delta w_i, \quad (1)$$

where $g_i = \partial L(w_i) / \partial w_i$ denotes the first-order partial derivative of the loss with respect to w_i , i.e., the gradient of the loss. Therefore, when a weight w_i varies Δw_i , the variation of the loss (and the model accuracy) can be approximated by the product of the relative gradient and perturbation Δw_i :

$$\Delta L(w_i) = L(w_i + \Delta w_i) - L(w_i) \approx g_i \cdot \Delta w_i. \quad (2)$$

The gradient g_i naturally serves as a sensitivity measure of the weight perturbation Δw_i on the loss.

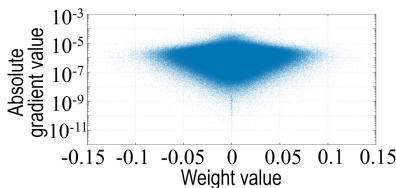


Fig. 1. The scatter plot shows the relationship between weights and their corresponding absolute gradient values for the Q projection matrix in the 10-th Transformer block of LLaMA-7B. The gradients are calculated from 128 segments, each containing 2048 tokens, randomly sampled from the C4 dataset.

As we know, the conventional clustering-based quantization (e.g., K-means [13], GOBO [14]) groups weights with similar values into one cluster and, next, replaces them with the cluster centroid. This minimizes weight perturbations, aiming to mitigate accuracy loss. However, even with small weight

perturbations, the relative gradients can still be very large, leading to significant loss variations based on (2). For instance, Fig. 1 shows a scatter plot of weights versus their absolute gradient values for the Q projection matrix in the 10-th transformer block of LLaMA-7B. The gradients are calculated using a calibration dataset. These gradients of the weights range from 10^{-12} to 10^{-3} . According to (2), replacing the weights close to zero with gradients around 10^{-3} may lead to much larger loss variations after quantization compared to other weights, even those with large values.

To address the gradient sensitivity issue, we explicitly integrate loss gradients into the clustering process. We begin with a pre-trained model and focus on a single layer's weights. Using a calibration dataset, we calculate the gradient for each weight via backpropagation, yielding the gradient set $\mathbf{g} = [g_0, g_1, \dots, g_{N-1}]^T$. Our objective is to partition the weights $\{w_i; i = 0, 1, \dots, N-1\}$ into K clusters, aiming to minimize the loss variation incurred by replacing each weight with the centroid of its corresponding cluster. To achieve this goal, first, we initialize a set of K cluster centroids, $\mathbf{w}_c = [w_c^{(0)}, w_c^{(1)}, \dots, w_c^{(K-1)}]^T$. Next, we conduct alternating optimization through two steps: (i) assigning weights to K clusters (referred to as *weight assignment*) and (ii) updating the cluster centroids (referred to as *centroid update*). This iterative process continues until convergence. Finally, we replace each weight in a cluster with the corresponding cluster centroid.

At the weight assignment step, for each weight w_i , we calculate its *gradient-scaled distance* to every cluster centroid $w_c^{(k)}$ ($k \in \{0, 1, \dots, K-1\}$):

$$e_i^{(k)} = \left[g_i \cdot (w_c^{(k)} - w_i) \right]^2. \quad (3)$$

According to (2) and (3), the gradient-scaled distance $e_i^{(k)}$ measures the change in the loss function when w_i is replaced by $w_c^{(k)}$. In other words, $e_i^{(k)}$ quantifies the cost (refer to the loss variation or the quantization error) of assigning w_i to the k -th cluster. Next, we assign w_i to the cluster corresponding to the smallest $e_i^{(k)}$ in the set $E_i = \{e_i^{(0)}, e_i^{(1)}, \dots, e_i^{(K-1)}\}$. This will minimize the impact on the loss function compared to assigning w_i to other clusters.

After assigning the weights \mathbf{w} to K clusters, the total cost of replacing all weights in the k -th cluster with the cluster centroid $w_c^{(k)}$ can be measured by the sum of the gradient-scaled distance between each weights in this cluster and $w_c^{(k)}$:

$$E^{(k)} = \sum_{i \in D^{(k)}} e_i^{(k)} = \sum_{i \in D^{(k)}} \left[g_i \cdot (w_c^{(k)} - w_i) \right]^2, \quad (4)$$

where $D^{(k)}$ represents the index set of all weights belonging to the k -th cluster. When this k -th cluster (and $D^{(k)}$) is fixed, minimizing $E^{(k)}$ gives the optimal $w_c^{(k)}$ that minimizes the loss variation. This $w_c^{(k)}$ is then assigned as the new k -th cluster centroid. Note that since $E^{(k)}$ is a quadratic function of $w_c^{(k)}$, the optimal value of $w_c^{(k)}$ can be analytically calculated as:

$$w_c^{(k)} = \arg \min E^{(k)} = \frac{\sum_{i \in D^{(k)}} g_i^2 \cdot w_i}{\sum_{i \in D^{(k)}} g_i^2}. \quad (5)$$

After updating the cluster centroids, we need to reassign weights. We repeat the weight assignment and centroid update steps until convergence occurs when the weight assignments remain unchanged. This iterative process results in optimized K clusters (and their corresponding weight index sets $\{D^{(0)}, D^{(1)}, \dots, D^{(K-1)}\}$).

$\dots, D^{(K-1)}\}$ along with their centroids $\mathbf{w}_c = [w_c^{(0)} w_c^{(1)} \dots w_c^{(K-1)}]^T$. These outputs minimize the total cost

$$E = \sum_{k=0}^{K-1} E^{(k)} \quad (6)$$

when all weights are replaced by their cluster centroid. We can quantize the model by applying this method to all linear layers.

Using the results from the gradient-aware clustering algorithm, we can substantially decrease the memory requirements for the linear layers in large-scale models. In general, each element in the weight vector \mathbf{w} and the inputs \mathbf{x} is typically stored as a floating-point number (e.g., FP16 or FP32), requiring N_{FP} bits each. Therefore, the total storage for \mathbf{w} is $N_o \times N_i \times N_{FP}$ bits. After clustering, the weights are grouped into K clusters. To save the clustering information, we establish a cluster index vector \mathbf{i}_c based on weight index sets $\{D^{(0)}, D^{(1)}, \dots, D^{(K-1)}\}$. Each element in \mathbf{i}_c represents the cluster index for the corresponding weight in \mathbf{w} , taking integer values from 0 to $K-1$. Suppose that storing each index requires N_Z bits, where N_Z is the smallest integer greater than $\log_2 K$. The total storage needed for \mathbf{i}_c is $N_o \times N_i \times N_Z$. Additionally, we also need to store K cluster centroids, which will be used to replace the weights in \mathbf{W} during inference. The memory required to store these centroids is $K \times N_{FP}$. Therefore, instead of using $N_o \times N_i \times N_{FP}$ bits to store the original weights \mathbf{w} , the total memory requirement after quantization reduces to $N_o \times N_i \times N_Z + K \times N_{FP}$, while N_{FP} is typically much larger than N_Z , and N_o and N_i are much larger than K . Note that the vector \mathbf{w} can be reshaped into a weight matrix \mathbf{W} with the size $N_o \times N_i$. Similarly, the vector \mathbf{i}_c can be reshaped into a $N_o \times N_i$ cluster index matrix \mathbf{I}_c . Each element in \mathbf{I}_c corresponds to the cluster index of the matching weight in \mathbf{W} .

B. Accumulation-Before-Multiplication Inference Scheme

As we know, during the inference stage, to compute each output, we multiply each element in the corresponding row of the weight matrix \mathbf{W} by the respective element of the input \mathbf{x} . This requires N_i floating-point operations for scalar multiplication (referred to as m-FLOPs). Next, we sum these products to obtain one output, which takes $N_i - 1$ floating-point operations for scalar addition (referred to as a-FLOPs). In total, to directly infer all outputs $\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$, we have to do $N_o \times N_i$ m-FLOPs and $N_o \times (N_i - 1)$ a-FLOPs.

Note that multiplication is more complex than addition, since it requires extra processing and incurs higher hardware and time costs. To reduce the number of multiplications and, thereby, decreasing hardware resource demands and computational cost of inference, we propose an ABM scheme informed by the clustering results. By using this method, we first calculate the sum of inputs with weights belonging to the same cluster and, next, multiply this sum by the centroid of that specific cluster.

Let us take Fig. 2 as an example. In this case, both input and output consist of 4 elements, i.e., $N_o = N_i = 4$. The weight matrix \mathbf{W} and the cluster index matrix \mathbf{I}_c both have dimensions of 4×4 . Let $\{w_{n,m}; n, m = 0, 1, 2, 3\}$ denote the weights in \mathbf{W} and $\{x_n; n = 0, 1, 2, 3\}$ denotes the elements in \mathbf{x} . Suppose that all weights in \mathbf{W} are assigned into $K = 2$ clusters. In \mathbf{I}_c , elements valued at 0 indicate that the corresponding weights in \mathbf{W} belong to the 0-th cluster. During the inference, these weights will be replaced by the cluster centroid $w_c^{(0)}$. Similarly, elements valued at 1 signify that the corresponding weights belong to the 1-st cluster, while these weights will be replaced by $w_c^{(1)}$. For instance, according

to the first row of \mathbf{I}_c (i.e., $[0 \ 1 \ 1 \ 0]$), the weights $w_{0,0}$ and $w_{0,3}$ from \mathbf{W} are assigned to the 0-th cluster, while the weights $w_{0,1}$ and $w_{0,2}$ belong to the 1-st cluster. Thus, we can calculate the 1-st output y_0 as follows:

$$\begin{aligned} y_0 &= w_{0,0} \cdot x_0 + w_{0,1} \cdot x_1 + w_{0,2} \cdot x_2 + w_{0,3} \cdot x_3 \\ &\approx w_c^{(0)} \cdot (x_0 + x_3) + w_c^{(1)} \cdot (x_1 + x_2) \\ &= w_c^{(0)} \cdot s^{(0)} + w_c^{(1)} \cdot s^{(1)} \end{aligned} \quad (7)$$

where $s^{(0)} = x_0 + x_3$ and $s^{(1)} = x_1 + x_2$ is referred to as *inner sums*. Hence, first, we calculate $K = 2$ inner sums $s^{(0)}$ and $s^{(1)}$. Next, we multiply inner sums by their corresponding cluster centroids respectively and, finally, sum the results. Namely, we perform the accumulation of certain inputs before calculating the multiplication of the input with the weights. This ABM process requires $K = 2$ m-FLOPs and $N_i - 1 = 3$ a-FLOPs. On the contrary, the direct calculation involves computing all products $\{w_{0,n} \cdot x_n; n = 0, 1, 2, 3\}$ before summing them, requiring $N_i = 4$ m-FLOPs and $N_i - 1 = 3$ a-FLOPs.

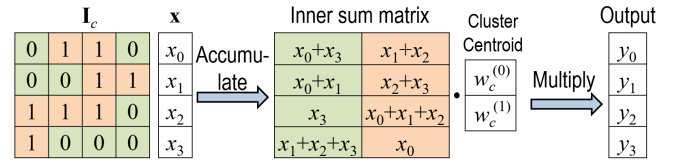


Fig. 2. We take a linear layer with a weight matrix of dimensions 4×4 as an example to explain the ABM process. The inputs are $\mathbf{x} = [x_0 \ x_1 \ x_2 \ x_3]^T$ and the output is $\mathbf{y} = [y_0 \ y_1 \ y_2 \ y_3]^T$. All weights in this linear layer are assigned to $K = 2$ clusters (distinguished by green and orange respectively), with the cluster centroids denoted as $w_c^{(0)}$ and $w_c^{(1)}$.

The above process can be directly applied to any-sized linear layer during inference. Suppose all weights \mathbf{W} of a linear layer with N_i inputs (i.e., $\mathbf{x} = [x_0 \ x_1 \ \dots \ x_{N_i-1}]^T$) and N_o outputs (i.e., $\mathbf{y} = [y_0 \ y_1 \ \dots \ y_{N_o-1}]^T$) are divided into K clusters. We construct a cluster index matrix \mathbf{I}_c based on these clustering results. The output y_n ($n \in \{0, 1, \dots, N_o - 1\}$) can be calculated by,

$$\begin{aligned} y_n &= w_{n,0} x_0 + w_{n,1} x_1 + \dots + w_{n,N_i-1} x_{N_i-1} \\ &\approx w_c^{(0)} \cdot \sum_{i \in D_n^{(0)}} x_i + w_c^{(1)} \cdot \sum_{i \in D_n^{(1)}} x_i + \dots + w_c^{(K-1)} \cdot \sum_{i \in D_n^{(K-1)}} x_i, \quad (8) \\ &= w_c^{(0)} \cdot s_n^{(0)} + w_c^{(1)} \cdot s_n^{(1)} + \dots + w_c^{(K-1)} \cdot s_n^{(K-1)} \end{aligned}$$

where $w_{n,j}$ ($j \in \{0, 1, \dots, N_i - 1\}$) represents the element in the n -th row and j -th column of the \mathbf{W} . $w_c^{(k)}$ denotes the centroid of the k -th cluster. $D_n^{(k)}$ is the index set of elements in the n -th row of \mathbf{I}_c that are equal to k , i.e., the index set of all weights in the n -th row of \mathbf{W} that belong to the k -th cluster. In addition,

$$s_n^{(k)} = \sum_{i \in D_n^{(k)}} x_i, \quad (9)$$

$k \in \{0, 1, \dots, K - 1\}$ represents the corresponding k -th inner sum. Therefore, to calculate y_n by using ABM, we first compute the inner sum for each cluster based on the n -th row vector of \mathbf{I}_c . Next, we sum the products of each inner sum with its corresponding cluster centroid. Thus, according to (8), we need to perform K m-FLOPs and $N_i - 1$ a-FLOPs.

By independently repeating this ABM process N_o times, we can infer all output values. Consequently, to compute all outputs, we need to do $N_o \times K$ m-FLOPs and $N_o \times (N_i - 1)$ a-FLOPs. Since K is typically much smaller than the number of inputs N_i , compared to directly calculate the matrix-vector multiplication

$\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$, our proposed method significantly reduces m-FLOPs from $N_o \times N_i$ to $N_o \times K$, while a-FLOPs remains unchanged.

III. IMPLEMENTATION DETAILS

A. Initialization of the Cluster Centroids

To improve clustering efficiency and accelerate convergence, we propose a modified K-means++ algorithm [15] to initialize cluster centroids by leveraging the gradient of the loss function.

We begin with the weights \mathbf{w} of a linear layer in a pre-trained model. First, we randomly select one weight from the set \mathbf{w} as the initial centroid $w_{c,init}^{(0)}$. At this point, the set of selected centroids is represented as $\mathcal{W}_{init} = \{w_{c,init}^{(0)}\}$.

Second, for each unselected weight w_n , we calculate the gradient-scaled distance from w_n to each selected centroid in the set \mathcal{W}_{init} . We find the minimum distance among these:

$$e_n^m = \min_{w_{c,init}^{(k)} \in \mathcal{W}_{init}} \left[\mathbf{g}_n \cdot (w_{c,init}^{(k)} - w_n) \right]^2, \quad (10)$$

Next, we compute the probability of each weight w_n being chosen as the next cluster centroid:

$$P(w_n) = \frac{e_n^{\min}}{\sum_m e_m^{\min}}. \quad (11)$$

After that, we randomly select the next cluster centroid $w_{c,init}^{(k+1)}$ based on this probability and add it to the set \mathcal{W}_{init} .

By iteratively calculating the minimum gradient-scaled distances and performing probability-based sampling, we sequentially select K cluster centroids.

Similar to the conventional K-means++ method, the proposed initialization approach uses a probabilistic process to select initial cluster centroids. According to (11), it prioritizes weights that are far away from the selected centroids, based on gradient-scaled distance, to be chosen as new centroids. The relatively uniform distribution of the initial centroids enhances numerical diversity and enables the algorithm to quickly converge.

B. Determination of the Number of Clusters

The number of clusters K plays a critical role in balancing model accuracy and compression efficiency. Small K forces many weights to share a few centroids, causing coarse quantization and substantial approximation errors, while larger K enriches codebook representational capacity, enabling finer approximations and higher accuracy. However, this improvement is accompanied by the growth in both storage and computational demands. The storage cost rises correspondingly, driven by two key factors. First, storing K centroids requires $K \times N_{FP}$ bits, meaning that the memory space needed for centroid storage grows linearly with K . Second, the index matrix \mathbf{I}_c , which records cluster assignments within the range $[0, K - 1]$, must be encoded with a bit-width N_Z (i.e., the smallest integer greater than $\log_2 K$) that also increases with K . Furthermore, under the ABM inference scheme, in the ABM inference scheme, the number of required multiplications (i.e., $N_o \times K$) grows linearly with K , thereby exacerbating the computational burden.

We apply the elbow method to identify the optimal number of clusters. First, we set a maximum cluster number, K_{max} . For K ranging from 2 to K_{max} , we implement GCPT on a validation dataset and, next, evaluate the accuracy of the quantized model. Afterwards, we plot a curve with K values on the x-axis and their model accuracy on the y-axis. Typically, this curve shows a rapid increase in accuracy for small K values, followed by a

gradual plateau as K rises. The K value at this elbow is considered the optimal number of clusters, as additional clusters yield only marginal improvements relative to their storage and computational costs.

C. Summary

Algorithm 1 describes gradient-aware clustering for a single linear layer. Given weights \mathbf{w} , their corresponding gradients \mathbf{g} , and the number of clusters K , each weight is assigned to the cluster whose centroid yields the smallest gradient-weighted distance to it from Step 5 to 8. Next, centroids are updated to minimize the sum of gradient-weighted distances within clusters in Step 12. This process repeats until the change in total cost E between iterations is below a predefined threshold, resulting in final converged clustering.

In the weight assignment step, each weight is allocated to the cluster that minimizes its gradient-scaled distance $e_i^{(k)}$. Therefore, according to (3), (4), and (6), the sum of all $\{e_i^{(k)}; i \in D^{(k)}, k = 0, 1, \dots, K - 1\}$, denoted as E , does not increase after this step. In the centroid update step, according to (5), each centroid is optimized to minimize $E^{(k)}$. Since E is the sum of all $E^{(k)}$, this step also ensures that E does not increase. Consequently, E monotonically decreases with each iteration. Moreover, since $E \geq 0$, the iteration process typically converges to a fixed point.

Algorithm 1: Gradient-aware Clustering for a Linear Layer

1. Input the weight vector \mathbf{w} of a linear layer, the corresponding gradient vector \mathbf{g} , the number of clusters K and a user-defined tolerance ε .
 2. Apply the proposed gradient-aware initialization method to identify a set of K cluster centroids $\{w_{c,init}^{(0)}, w_{c,init}^{(1)}, \dots, w_{c,init}^{(K-1)}\}$. Let $w_c^{(k)} = w_{c,init}^{(k)}$ ($k \in \{0, 1, \dots, K - 1\}$).
 3. Initialize $t = 0$ and the total cost $E_t = 0$.
 4. While (1)
 5. For $i = 0, 1, \dots, N - 1$
 6. Calculate the gradient-scaled distance $e_i^{(k)}$ from w_i to each centroid $w_c^{(k)}$ ($k \in \{0, 1, \dots, K - 1\}$) according to (3).
 7. Assign w_i to the cluster corresponding to the smallest $e_i^{(k)}$ in the set $E_t = \{e_i^{(0)}, e_i^{(1)}, \dots, e_i^{(K-1)}\}$.
 8. End For
 9. Calculate each sum of the gradient-scaled distance $E^{(k)}$ ($k \in \{0, 1, \dots, K - 1\}$) according to (4).
 10. Calculate the total cost E_{t+1} according to (6).
 11. If $|E_t - E_{t+1}| < \varepsilon$, break. Otherwise, update $t = t + 1$.
 12. Update the cluster centroid $w_c^{(k)}$ according to (5).
 13. End While
 14. Construct the cluster index vector \mathbf{i}_c according to the clustering results.
-

Algorithm 2 outlines the GCPT-based quantization process for a whole model. In Step 2, a calibration dataset is used to compute loss gradients for all model weights. From Step 3 to 6, for each candidate K , we apply Algorithm 1 to cluster the weights of each linear layer and, next, evaluate the quantized model using ABM scheme on a validation dataset. In Step 7, the inference errors corresponding to different cluster numbers are analyzed, and the optimal quantization configuration is selected using the elbow method. Note that by using this method, the number of weight clusters for each linear layer is the same.

Algorithm 2: Gradient-aware Clustering Method for Post-Training Quantization

1. Input a pre-trained model, a calibration dataset D_C , a validation dataset D_V , and a user-defined maximum cluster number K_{max} .
 2. Use D_C to calculate loss gradients for each weight in the linear layers via backpropagation.
 3. For $K = 2, 3, \dots, K_{max}$
 4. For each linear layer in the model, apply Algorithm 1 to obtain the cluster centroids \mathbf{w}_c and the cluster index vector \mathbf{i}_c .
 5. Apply the ABM scheme to evaluate the inference accuracy P_K on the validation dataset D_V with a quantized model having K weight clusters.
 6. End For
 7. Identify the elbow point on the curve plotting inference accuracy (i.e., $\{P_K; K = 2, 3, \dots, K_{max}\}$) against the number of clusters (i.e., K). The quantized model corresponding to this point is regarded as the optimal model.
-

IV. EXPERIMENT RESULTS

In this section, we demonstrate the efficacy of the proposed GCPT method using three large neural network models: LLaMA-7B and LLaMA-13B [2] with popular Transformer-based architectures, and ResMLP-S24/16 [16] with a multilayer perceptron (MLP)-based architecture for image classification.

For LLaMA-7B and LLaMA-13B, the loss gradient for each weight in each linear layer are calculated with 128 segments of 2048 tokens, randomly sampled from the C4 dataset [17]. Note that computing these gradients requires only a single backward pass on the calibration set, with an actual cost roughly equivalent to several forward passes. We evaluate the perplexity and zero-shot accuracy of GCPT against several quantization algorithms for large language models, including K-means [13], LLM-QAT [7], GPTQ [11], APTQ [12], and the original models.

For ResMLP-S24/16, we use 1280 randomly selected images from the ImageNet-1K training set [18] to compute loss gradients. We compare the Top-1 and Top-5 accuracy of GCPT with several quantization algorithms originally developed for general NNs, including PTQ [8], K-means [13], HAWQ-V3 [19], and the original model. Additionally, we analyze the inference costs of different methods.

A. LLaMa Models

For both LLaMA-7B and LLaMA-13B, we utilize the WikiText-2 validation dataset [20] to identify the optimal number of clusters. Fig. 3(a) shows the perplexity of LLaMA-7B as K varies. Perplexity decreases significantly as K increases from small values, with a noticeable elbow at $K = 16$. Beyond this point, the reduction in perplexity becomes minimal. A similar trend is observed for LLaMA-13B as shown in Fig. 3(b), revealing a similar elbow at $K = 16$.

We evaluate the perplexity of the LLaMA-7B model using GCPT with $K = 16$ and, furthermore, comparing it against several quantization methods on the WikiText-2 test dataset [20], as summarized in TABLE I. “Avg bit” represents the average bit-width of quantized model parameters, and “Size” refers to the model storage size. “# of mF” and “# of aF” denote the counts of scalar floating-point multiplication and addition operations (i.e., the number of m-FLOPs and a-FLOPs)

respectively, whereas “# of mB” and “# of aB” correspond to the bit operations count for multiplication and addition (also referred to as m-BOPs and a-BOPs) respectively. “PPL”, short for perplexity, is a metric that reflects uncertainty in next-token prediction and is commonly used to assess LLM performance. “Baseline” means the original LLaMA-7B model using FP16, serving as the baseline. After quantization, all models use 4-bit weights on average and occupy around 3.7GB.

According to TABLE I, the proposed GCPT method attains a perplexity of 6.33, making it the closest to the baseline (with a perplexity of 5.68), and outperforming other quantization methods. Note that K-means method exhibits high perplexity, as it clusters weights based solely on magnitude, which can introduce large errors for weights with high gradients and increase the loss. In contrast, GCPT assigns weights based on gradient-scaled distance and updates centroids to minimize loss variation, thereby maintaining low perplexity.

TABLE I also lists the computational costs of inference for different methods. In LLaMA-7B, linear layers dominate the workload [2]. This model consists of 32 Transformer blocks, each with 7 linear layers: 4 linear projections of size $d \times d$ from self-attention, 2 projections of size $d \times f$, and 1 projection of size $f \times d$ from the feed-forward network, where $d = 4096$ and $f = 11008$. Hence, for the baseline, total m-FLOPs per block is $4d^2 + 3df \approx 202\text{M}$. Across 32 blocks, this totals 5.9G per token, while a-FLOPs are comparable. GPTQ, APTQ and LLM-QAT, retain the full GEMM structure, resulting in the same costs. In contrast, the clustering methods (i.e., K-means and GCPT) group all linear weights into $K = 16$ clusters. Using the ABM scheme, the per-block multiplication reduces to $4dK + 2fK + dK \approx 0.68\text{M}$ and, hence, the total number drops to $0.68\text{M} \times 32 \approx 21.8\text{M}$. Namely, compared to the existing method, we achieve a $297\times$ reduction in m-FLOPs, while a-FLOPs remain unchanged at 6.48G.

To further quantify hardware efficiency, we analyze the number of BOPs, including m-BOPs and a-BOPs. As we know, a multiplication between a weight of bit-width b_w and an input of bit-width b_i incurs $b_w \times b_i$ m-BOPs, while an addition requires b_i a-BOPs when b_i larger than b_w [21]. For all methods, the input is in FP16 format, meaning $b_i = 16$. Thus, total a-BOPs for all methods is the same, i.e., “# of aF” $\times 16 \approx 104\text{G}$.

Since by using GPTQ and APTQ, the weights have to be dequantized to FP16 for inference (i.e., $b_w = 16$), their m-BOPs is “# of mF” $\times 16 \times 16 \approx 1659\text{G}$, the same as that of the baseline. LLM-QAT quantizes the model weights to 4 bits (i.e., $b_w = 4$) and, hence, results in 415G m-BOPs. For K-means and GCPT, multiplications are performed between the cluster centroids and the inner sum, both in FP16 format. This results in a total of “# of mF” $\times 16 \times 16 \approx 5.58\text{G}$ for m-BOPs. Although GCPT provides a slight perplexity improvement over APTQ, it reduces the total BOPs (i.e., the sum of m-BOPs and a-BOPs) by about 93.8%.

We further evaluate the zero-shot performance of LLaMA-7B/13B quantized by different methods on 5 benchmarks: PIQA [22], HellaSwag (i.e., “Hella.”) [23], ARC-Easy (i.e., “ARC-E”) [24], ARC-Challenge (i.e., “ARC-C”) [24], and WinoGrande (i.e., “Wino”) [25], as summarized in TABLE II. The “Avg acc” refers to the average accuracy across these 5 benchmarks.

According to TABLE II, the proposed GCPT achieves an average accuracy of 68.00% on LLaMA-7B, closely rivaling APTQ at 68.08% and just 0.6% below the baseline of 68.56%. On LLaMA-13B, GCPT achieves an average accuracy of

70.74%, outperforming all other methods and falling only 0.20% short of the baseline at 70.94%. These results indicate that GCPT maintains semantic understanding with minimal degradation. While its average accuracy on LLaMA-7B closely matches that of APTQ, GCPT benefits from a significantly lower computational cost according to the aforementioned discussion.

B. ResMLP-S24/16

For ResMLP-S24/16, we construct the validation and testing sets based on ImageNet-1K [18]. One-tenth of the images from each category form the validation set and the remaining images form the test set. We evaluate Top-1 and Top-5 accuracy as K varies by using the validation set. As shown in Fig. 3(c), both Top-1 and Top-5 accuracies improve significantly at $K = 8$, with diminishing gains after $K = 16$. By $K = 32$, further gains are negligible. Therefore, the optimal number of clusters for ResMLP-S24/16 is also $K = 16$.

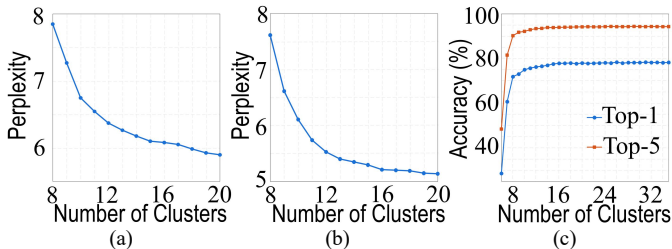


Fig. 3. The perplexity scores for LLaMA-7B and LLaMA-13B on the WikiText-2 validation dataset are presented in (a) and (b). These scores were obtained using the proposed clustering method with different K values. Meanwhile, (c) provides the Top-1 and Top-5 accuracy on the validation set constructed from ImageNet-1K for ResMLP-S24/16.

TABLE III summarizes the Top-1 and Top-5 accuracy (represented as “Top-1/5”) of ResMLP-S24/16, evaluated on the test set consisting of nine-tenths of the data from each class in ImageNet-1K. “Baseline” represents the original ResMLP-S24/16 model, serving as the baseline. By default, the parameters of ResMLP-S24/16 are stored in FP32 format. To provide a comprehensive accuracy comparison, we evaluate all methods using models quantized to average bit widths of 3, 4 and 5, corresponding to $K = 8, 16$, and 32 , respectively.

According to TABLE III, the GCPT achieves a Top-1 accuracy of 78.47% and Top-5 accuracy of 94.20% with average 4-bit parameters ($K = 16$), closely matching the baseline of 79.19% and 94.48%. With 5-bit parameters ($K = 32$), accuracy almost reaches the baseline. Even with 3-bit parameters ($K = 8$), GCPT maintains a Top-1 accuracy of 69.70%, outperforming other methods. Namely, GCPT delivers superior performance with only a slight increase in storage cost due to the cluster centroids.

In addition, as we know, ResMLP-S24/16 has 24 ResMLP blocks, each with 3 linear layers: one of size 196×196 applied across 384 features, and two of size 1536×384 and 384×1536 applied across 196 tokens. Therefore, for the baseline, m-FLOPs per block is 246M. Across 24 blocks, this totals 5.9G, while a-FLOPs are similar. The PTQ and HAWQ-V3, maintain the full GEMM structure, resulting in the same costs. In contrast, by using ABM for K-means and GCPT, the total number of m-FLOPs reduces to $24 \times (196 \times K \times 384 + 1536 \times K \times 196 + 384 \times K \times 196) \approx 10.8 \times K$ M, while a-FLOPs remain unchanged.

Furthermore, the input is in FP32 format for all methods, meaning $b_i = 32$. Hence, for the baseline, total m-BOPs is 6042G, and a-BOPs for all methods is the same 189G. For PTQ and HAWQ-V3, m-BOPs are their “# of mF” $\times 3/4/5 \times 32$. For K-

means and GCPT, multiplications are performed between FP32 centroids and FP32 inner sums, so m-BOPs is “# of mF” $\times 32 \times 32$. Compared to other quantization methods, GCPT reduces total BOPs by 63.3%, 61.2%, and 52.1% under 3/4/5-bit quantization, respectively.

Note that under a fixed number of clusters K , ABM reduces per layer multiplications from $N_o \times N_i$ to $N_o \times K$. Therefore, the reduction ratio is positively correlated with the input dimension N_i of each linear layer. As a result, layers with larger N_i benefit more significantly from clustering. This explains why GCPT leads to a much higher proportion of overall computation cost savings in LLaMA models compared to ResMLP, where most linear layers are relatively smaller. In addition to efficiency, GCPT consistently maintains accuracy across both architectures, highlighting its scalability in both performance and precision.

TABLE I. PERPLEXITIES OF 4-BIT QUANTIZED MODELS OF LLaMA-7B ON WIKITEXT-2.

| Method | Avg bit | Size(GB) | # of mF | # of mB | # of aF | # of aB | PPL |
|-------------|---------|----------|--------------|--------------|---------|---------|-------------|
| Baseline | 16 | 13.4 | 6.48G | 1659G | 6.48G | 104G | 5.68 |
| K-means | 4 | 3.7 | 21.8M | 5.58G | 6.48G | 104G | 23894 |
| LLM-QAT | 4 | 3.7 | 6.48G | 415G | 6.48G | 104G | 10.90 |
| GPTQ | 4 | 3.7 | 6.48G | 1659G | 6.48G | 104G | 8.14 |
| APTQ | 4 | 3.7 | 6.48G | 1659G | 6.48G | 104G | 6.45 |
| GCPT | 4 | 3.7 | 21.8M | 5.58G | 6.48G | 104G | 6.04 |

TABLE II. ZERO-SHOT ACCURACY OF LLaMA-7B AND LLaMA-13B ON 5 BENCHMARKS.

| Model | Method | Avg bit | PIQA | Hella. | Arc-E | Arc-C | Wino. | Avg acc (%) |
|-----------|-------------|---------|-------------|-------------|-------------|-------------|-------------|--------------|
| LlaMa-7B | Baseline | 16 | 79.2 | 76.2 | 72.8 | 44.7 | 69.9 | 68.56 |
| | LLM-QAT | 4 | 78.3 | 74.0 | 70.0 | 41.7 | 69.0 | 66.60 |
| | GPTQ | 4 | 76.0 | 69.4 | 66.9 | 43.0 | 66.7 | 64.40 |
| | APTQ | 4 | 78.6 | 75.7 | 72.4 | 44.4 | 69.3 | 68.08 |
| | GCPT | 4 | 78.8 | 76.0 | 71.8 | 44.4 | 69.5 | 68.10 |
| LlaMa-13B | Baseline | 16 | 80.3 | 79.0 | 74.8 | 47.9 | 72.7 | 70.94 |
| | LLM-QAT | 4 | 79.4 | 77.7 | 72.8 | 47.3 | 71.5 | 69.74 |
| | GPTQ | 4 | 79.8 | 77.7 | 73.2 | 45.9 | 72.6 | 69.84 |
| | APTQ | 4 | 79.9 | 78.8 | 73.9 | 47.0 | 72.1 | 70.34 |
| | GCPT | 4 | 79.6 | 79.3 | 73.9 | 47.5 | 72.4 | 70.54 |

TABLE III. TOP-1 AND TOP-5 ACCURACY OF RESMLP-S24/16 ON THE IMAGENET-1K VALIDATION SET.

| Method | Avg bit | Size(MB) | # of mF | # of mB | # of aF | # of aB | Top-1/5 (%) |
|-------------|---------|----------|---------|---------|---------|---------|--------------------|
| Baseline | 32 | 120 | 5.9G | 6042G | 5.9G | 189G | 79.19/94.48 |
| PTQ | 3 | 11.3 | 5.9G | 566G | 5.9G | 189G | 0.14/0.55 |
| | 4 | 15.0 | 5.9G | 755G | 5.9G | 189G | 38.85/60.79 |
| | 5 | 18.8 | 5.9G | 944G | 5.9G | 189G | 75.46/92.59 |
| K-means | 3 | 11.9 | 88M | 85G | 5.9G | 189G | 23.48/45.10 |
| | 4 | 15.6 | 177M | 170G | 5.9G | 189G | 75.72/92.61 |
| | 5 | 19.3 | 346M | 354G | 5.9G | 189G | 77.68/93.76 |
| HAWQ-V3 | 3 | 11.3 | 5.9G | 566G | 5.9G | 189G | 12.35/24.04 |
| | 4 | 15.0 | 5.9G | 755G | 5.9G | 189G | 64.01/82.31 |
| | 5 | 18.8 | 5.9G | 944G | 5.9G | 189G | 78.43/94.05 |
| GCPT | 3 | 11.9 | 86M | 88G | 5.9G | 189G | 69.70/88.86 |
| | 4 | 15.6 | 173M | 177G | 5.9G | 189G | 78.47/94.20 |
| | 5 | 19.3 | 346M | 354G | 5.9G | 189G | 79.08/94.44 |

V. CONCLUSION

In this paper, we propose a novel GCPT method for efficient post-training quantization of large NNs. GCPT incorporates gradient-aware clustering to minimize quantization-induced loss variation. Furthermore, we introduce the ABM scheme to reduce multiplication operations during inference while preserving accuracy. As demonstrated by the experiments on LLaMA-7B, LLaMA-13B, and ResMLP-S24/16, compared to the state-of-the-art quantization methods, the proposed GCPT method achieves up to 95% reduction in total BOPs while maintaining accuracy.

REFERENCES

- [1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman and S. Anadkat, et al., “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro and F. Azhar, et al., “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [3] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” *Adv. Neural Inf. Process. Syst.*, pp. 1135-1143, 2015.
- [4] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang and S. Gu, “Pushing the limit of post-training quantization by block reconstruction,” *ICLR*, 2021.
- [5] X. Yu, T. Liu, X. Wang and D. Tao, “On compressing deep models by low-rank and sparse decomposition,” *CVPR*, 2017.
- [6] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” *Low-Power Computer Vision*, pp. 291-326, 2022.
- [7] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi and V. Chandra, “LLM-QAT: Data-free quantization aware training for large language models,” *arXiv preprint arXiv:2305.17888*, 2023.
- [8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” *CVPR*, pp. 2704-2713, 2018.
- [9] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler and D. Alistarh SpQR, “A sparse-quantized representation for near-lossless LLM weight compression,” *arXiv preprint arXiv:2306.03078*, 2023.
- [10] C. Xue, C. Zhang, X. Jiang, Z. Gao, Y. Lin and G. Sun, “Oltron: Software-hardware co-design for outlier-aware quantization of LLMs with inter-/intra-layer adaptation,” *DAC*, 2024.
- [11] E. Frantar, S. Ashkboos, T. Hoefler and D. Alistarh, “GPTQ: Accurate post-training compression for generative pretrained transformers,” *ICLR*, 2023.
- [12] Z. Guan, H. T. Huang, Y. Su, H. Huang, N. Wong and H. Yu, “APTQ: Attention-aware post-training mixed-precision quantization for large language models,” *DAC*, 2024.
- [13] S. Han, H. Mao and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” *ICLR*, 2016.
- [14] A. H. Zadeh, I. Edo, O. M. Awad and A. Moshovos, “GOBO: Quantizing attention-based NLP models for low latency and energy efficient inference,” *MICRO*, pp. 811-824, 2020.
- [15] A. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” *Technical Report*, Stanford University, 2006.
- [16] H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, G. Izacard, A. Joulin, G. Synnaeve, J. Verbeek and H. Jégou, “ResMLP: Feedforward networks for image classification with data-efficient training,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 5314-5321, 2023.
- [17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li and P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 5485-5551, 2020.
- [18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and F. Li, “ImageNet large scale visual recognition challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211-252, 2015.
- [19] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. W. Mahoney and K. Keutzer, “HAWQ-V3: Dyadic neural network quantization,” *ICML*, 2021.
- [20] S. Merity, C. Xiong, J. Bradbury and R. Socher, “Pointer sentinel mixture models,” *arXiv preprint arXiv:1609.07843*, 2016.
- [21] A. Karbachevsky, C. Baskin, E. Zheltonozhskii, Y. Yermolin, F. Gabbay, A. Bronstein and A. Mendelson, “Early-stage neural network hardware performance analysis,” *Sustainability*, vol. 13, no. 2, pp. 717-737, 2021.
- [22] Y. Bisk, R. Zellers, J. Gao, Y. Choi, A. Sabharwal, J. Michaelov, S. Bosselut, N. A. Smith, and Y. Zhang, “PIQA: Reasoning about physical commonsense in natural language,” *AAAI*, vol. 34, no. 5, pp. 7432-7439, 2020.
- [23] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, “HellaSwag: Can a machine really finish your sentence?” *ACL*, pp. 4791-4800, 2019.
- [24] P. Clark, I. Cowhey, O. Etzioni, D. Khot, A. Sabharwal, T. Schoenick, and O. Tafjord, “Think you have solved question answering? Try ARC, the AI2 reasoning challenge,” *arXiv preprint arXiv:1803.05457*, 2018.
- [25] K. Sakaguchi, R. Bras, C. Bhagavatula, and Y. Choi, “WinoGrande: An adversarial Winograd schema challenge at scale,” *AAAI*, vol. 34, no. 5, pp. 8732-8740, 2020.