MemeBQ: Memory Efficient Binary Quantization of LLMs

Yuanhui Wang[1*], Kunlong Liu[1†*], Minnan Pei[2], Zhangming Li[2], Peisong Wang[2,3,4], Qinghao Hu[2,3,4‡]

[1]Sanya Nanhai Innovation and Development Base of Harbin Engineering University,
[2]C[2]DL, Institute of Automation, Chinese Academy of Sciences,
[3]Nanjing Artificial Intelligence Research of IA (AiRiA),
[4]University of Chinese Academy of Science, Nanjing
{liukunlong, wangyuanhui}@hrbeu.edu.cn, {peiminnan2023, lizhangming2023,huqinghao2014}@ia.ac.cn,
peisong.wang@nlpr.ia.ac.cn

Abstract

Recent years have witnessed growing scholarly interest in binary post-training quantization (PTQ) techniques for large language models (LLMs). While state-of-the-art (SOTA) binary quantization methods significantly reduce memory footprint and computational demands, they introduce additional memory overhead beyond binary weight tensors to mitigate performance degradation. Moreover, binary LLMs still suffer from substantial accuracy loss. To address these limitations, we propose MemeBQ, a novel binary PTQ framework for LLMs that reduces the memory overhead of auxiliary flag bitmaps in existing binary quantization methods. Specifically, we first design a greedy row clustering method, which leverages the similarity between the row vectors of weights to partition the weight rows into different groups. By sharing the common flag bitmap within each row group, we significantly mitigate the memory overhead associated with flag bitmaps. Besides, to improve the performance of binary LLMs, we propose a novel weight splitting method for each row group of weights, which determines the flag bitmap's values in a fine-grained way. Extensive experiments on OPT, Llama-2, and Llama-3 models demonstrate that MemeBQ reduces 50% extra memory demand while achieving comparable accuracy compared with current SOTA methods. Alternatively, MemeBQ outperforms SOTA binary quantization methods up to 7% with the same extra bits on reasoning benchmarks.

Code — https://github.com/88099981/MemeBQ

Introduction

In recent years, the advancement of large language models (LLMs) has significantly propelled the progress of artificial intelligence, yet the ever-expanding parameter scales (e.g., Llama-2 with 70B parameters) demand substantial computational resources for inference, impeding their deployment on edge devices or in low-resource environments. This challenge promotes researchers to propose various LLMs compression methods including model quantization(Frantar et al. 2022; Lin et al. 2024), pruning(Frantar and Alistarh 2023; Ashkboos et al. 2024), knowledge distillation(Ko et al. 2024; Feng et al. 2025), and low-rank decomposition(Tian et al. 2025; Saha et al. 2024). Among these model compression methods, post-training quantization enables significantly faster deployment and lower computational overhead, as it directly quantizes a pre-trained model, eliminating the need for the resource-intensive retraining, fine-tuning, or architectural modifications required by other techniques. As a special case of model quantization, binary post-training quantization has attracted considerable attention in recent years due to its extremely low memory demand. However, due to the significant quantization errors inherent in binary representations, most binary quantization methods (Shang et al. 2023; Huang et al. 2024; Li et al. 2024) require additional masks or bitmaps alongside the binary weights to maintain model performance, as summarized in Table 1. For instance, PB-LLM (Shang et al. 2023) categorizes salient weights (outliers) into a separate group quantized to 8 bits, while applying binary quantization to the non-salient weights. This approach necessitates an unstructured bitmap to store the category indices. BiLLM (Huang et al. 2024) separates salient and non-salient weights structurally by column. It further partitions non-salient weights into sparse and concentrated categories, requiring a binary flag bitmap to store these splitting indices. ARB-LLM (Li et al. 2024) extends this concept by partitioning weights within salient columns based on magnitude, achieving state-of-the-art results but retaining the same extra bitmap requirement as

Algorithm	Weight Bits	Extra Bits
PB-LLM	1.70	1.00
BiLLM	1.08	1.00
ARB-LLM$_{RC}$	1.08	1.00

Table 1: Extra bits introduced by prior methods

[*]These authors contributed equally.
[†]This work was completed while the author was with the Institute of Automation, Chinese Academy of Sciences.
[‡]Corresponding authors

BiLLM. The additional overhead introduced by these flag bitmaps remains non-negligible, leading to increased memory and storage consumption.

To alleviate the above issue, we analyzed the extra flag bitmap and the non-salient weights splitting mechanism, and found the following two observations:

- A row of weight matrix has a high degree of similarity as other rows of weight matrix. Specifically, the cosine similarity between a row of weight and its most similar row reaches at least 85% for Llama2-7b model. This indicates that the flag bitmap has redundancy in similar rows.
- Previous binary LLMs splits the non-salient weights using a global threshold for the whole weight matrix. The global threshold may work well in similar rows of weights, yet it will cause more quantization errors for those dissimilar rows of weights. A fine-grained splitting method for weight rows may lower down the quantization error.

Based on these observations and analyses, we propose a novel binary post-training quantization framework called MemeBQ to reduce the additional overhead issue of the bitmap mask. MemeBQ consists of two main modules called similarity search(SimS) and group bitmap splitting(GBS). Specifically, SimS leverages a greedy search algorithm to group the rows of weights according to the cosine similarity. The rows of weights within the same group share the same flag bitmap which reduces the flag bitmap size by a factor of groupsize. Concurrently, GBS splits the non-salient weights via applying k-means clustering within each row group that is clustered by SimS, this fine-grained row-group level splitting mechanism aligns with the local weight distribution.

Our key contributions are summarized as follows:

- We propose a novel bitmap reducing method named SimS algorithm via clustering rows of weights with similar direction into groups, where the rows of weights within the same group share the same bitmap mask.
- We propose a fine-grained non-salient weight splitting method, i.e., the GBS algorithm, which ensures the shared bitmap mask within each group is more congruent with the original weight distribution of grouped rows of weights, thereby significantly improving model performance.
- We conduct extensive experiments on OPT, Llama-2, and Llama-3 models, demonstrating that MemeBQ achieves performance comparable to SOTA binary PTQ methods while requiring 50% less extra memory demand. Alternatively, MemeBQ outperforms SOTA binary quantization methods up to 7% with the same extra bits on reasoning benchmarks.

Related Work

Quantization is the process of transforming model weights or activations into discrete values. Post-training quantization (PTQ) is a widely adopted method for quantizing large language models. 8-bit quantization was the main method in the early phase. For instance, the LLM.int8 method (Dettmers et al. 2022) employs vector quantization to simultaneously reduce the bit width of model activations and weights to 8 bits. ZeroQuant (Yao et al. 2022) introduces a flexible multi-bit quantization approach that accommodates mixed precision with 4-bit and 8-bit configurations. In the context of quantizing weights in large models, the GPTQ method (Frantar et al. 2022) proposes a quantization compensation strategy grounded in the Hessian matrix, the bit width of each weight can be reduced to 3 or 4 bits with minimal impact on accuracy. AWQ (Lin et al. 2024) has demonstrated that the weight values associated with channels exhibiting higher activation levels are more crucial. SpQR (Dettmers et al. 2023) takes this a step further by storing weight outliers in a higher bit width while quantizing the remaining weights to lower bit widths.

In the realm of lower-bit quantization, QuIP (Chee et al. 2023) employs an adaptive rounding process aimed at minimizing a quadratic proxy objective function. Ultimately achieving 2-bit quantization for large model weights. PB-LLM (Shang et al. 2023) proposes organizing the original weight matrix based on sensitivity, categorizing outliers into 8-bit quantization group while applying 1-bit quantization to others. BiLLM (Huang et al. 2024) applies 2-bit residual quantization to the salient weights and employs a grouped 1-bit algorithm for the non-salient weights, ultimately achieving an overall PTQ quantization of 2.08 bits. Furthermore, ARB-LLM (Li et al. 2024) introduces a method to enhance the accuracy of the scaler and zero-point through iterative refinement. However, this method also necessitates the additional storage of a 1-bit flag bitmap.

Method

Overview

As shown in Figure 1, to reduce the extra storage overhead of the bitmap mask, we design the SimS algorithm to search and group similar rows of weights, reducing the extra flag bitmap by sharing mask in the same group. Additionally, we propose the GBS algorithm to ensure the shared bitmap mask better aligns with the original weight distribution.

Similarity Search(SimS)

The SimS algorithm is introduced to eliminate redundant storage overhead caused by bitmap masks in prior binary quantization methods (e.g., BiLLM/ARB-LLM), which previously incurred an unavoidable 1-bit-per-parameter extra storage penalty. Through visualizing the bitmap mask generated by BiLLM as Figure 2, we observed the presence of similar row vector patterns, indicating that the corresponding original weights exhibit high similarity in distribution across rows, which provide the foundational insight for the SimS algorithm. The core idea of SimS is to reduce storage overhead by grouping rows of weights with similar distributions and allowing them to share the same bitmap mask. Specifically, we first divide the weights into salient weights and non-salient weights structurally following prior work(Huang et al. 2024; Li et al. 2024). Given the weight matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$ with n out-channels and m in-channels, we compute the quantization sensitivity of each element us-

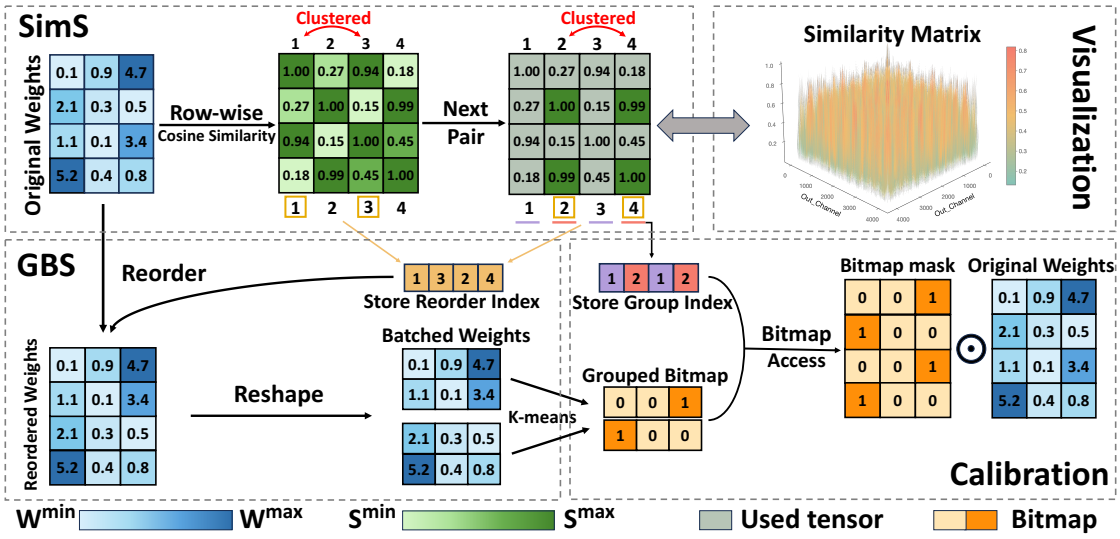

Figure 1: Overview of our MemeBQ method

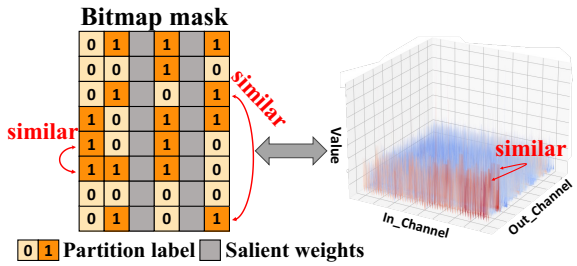

Figure 2: Bitmap mask of BiLLM method, the number 0&1 are results of weights partition, similar inter-row labels mean similar original weights distribution

ing the Hessian matrix $\mathbf{H} \in \mathbb{R}^{m\times m}$:

$$e_i = \frac{w_i^2}{[\mathbf{H}^{-1}]_{ii}^2}, \quad \text{where } \mathbf{H} = \mathbf{X}\mathbf{X}^{\mathbf{T}}. \tag{1}$$

Then weight columns with the highest l sensitivity are identified as salient weights, representing the salient weight column index by $e\_idx \in \mathbb{R}^m$, while the remainder are identified as non-salient weights. Salient weights undergo residual binary quantization to minimize quantization error following (Li et al. 2024).

Due to significant distributional differences between salient and non-salient components, similarity search is performed separately. For weight matrix $\mathbf{W} \in \mathbb{R}^{n\times m}$, we use the salient column indice e_idx to get $\mathbf{W_{salient}} \in \mathbb{R}^{n\times l}$and $\mathbf{W_{non-salient}} \in \mathbb{R}^{n\times(m-l)}$. SimS algorithm firstly forms two reconstructed matrices $\mathbf{S_{salient}} \in \mathbb{R}^{n\times n}$ and $\mathbf{S_{non-salient}} \in \mathbb{R}^{n\times n}$ based on $\mathbf{W_{salient}}$ and $\mathbf{W_{non-salient}}$:

$$s_{ij} = \frac{\mathbf{W_{i\cdot}} \cdot \mathbf{W_{j\cdot}^T}}{||\mathbf{W_{i\cdot}}||_2 \cdot ||\mathbf{W_{j\cdot}}||_2}, \quad i,j \in \{1,\ldots,n\}. \tag{2}$$

where $\mathbf{W_{i\cdot}}$ denotes the i row of $\mathbf{W_{salient}}$ or $\mathbf{W_{non-salient}}$. Subsequently, the greedy algorithm is used to find the most similar yet ungrouped tensors in the similarity matrix S. For each row i in $\mathbf{S_{salient}}$ and $\mathbf{S_{non-salient}}$, we find the index col of maximum similarity for g times where g denotes the groupsize:

$$col = \arg\max_j \mathbf{S}_{i,j}, \quad j \in \{1,\ldots,n\} \tag{3}$$

then update S and store group information in $\tilde{\mathbf{m}}^{\mathbf{g}} \in \mathbb{R}^{n\times 1}$. g_idx is used to indicate the number of current group:

$$\mathbf{S_{i\cdot}} = \mathbf{S_{\cdot j}} = -inf \tag{4}$$

$$\tilde{\mathbf{m}}_{\mathbf{i}}^{\mathbf{g}} = \tilde{\mathbf{m}}_{\mathbf{j}}^{\mathbf{g}} = g\_idx,\ g\_idx \in \{1,\ldots,\frac{n}{g}\} \tag{5}$$

What's more, we store $\tilde{\mathbf{m}}^{\mathbf{r}} \in \mathbb{R}^{n\times 1}$ for a better access through sub-tensor indexing in the latter GBS module, r_idx shows the current index in g, $\tilde{\mathbf{m}}^{\mathbf{r}}$ is updated by:

$$\begin{aligned} \tilde{\mathbf{m}}^{\mathbf{r}}_{\mathbf{g\times g\_idx}} &= \tilde{\mathbf{m}}^{\mathbf{r}}_{\mathbf{g\times g\_idx+r\_idx}} = col \\ \text{where } g\_idx &\in \{1,\ldots,\frac{n}{g}\},\ r\_idx \in \{1,\ldots,g\} \end{aligned} \tag{6}$$

All rows of weights within a group share a single bitmap mask. The detailed algorithm is presented in Algorithm 1.

Group Bitmap Search (GBS)

Previous algorithms partitioned the bitmap mask by approximating weights as a Gaussian distribution and finding the globally optimal threshold. Specifically, they first sorted the absolute values of all original weights, then iteratively searched for the best threshold within this sorted list, using the loss function:

$$\mathcal{L} = ||\mathbf{W} - \widehat{\mathbf{W}}||_F^2, \quad \text{where } \widehat{\mathbf{W}} = \alpha\mathbf{B} + \mu\,. \tag{7}$$

Algorithm 1: SimS algorithm

func SimS($\mathbf{W}$, g)

Input: $\mathbf{W} \in \mathbb{R}^{n \times m}$ - full-precision weight
g - group size

Output: $\tilde{\mathbf{m}}^{\mathbf{g}} \in \mathbb{R}^{n}$
$\tilde{\mathbf{m}}^{\mathbf{r}} \in \mathbb{R}^{n}$

$\tilde{\mathbf{d}}_{norm} \leftarrow \mathbf{W}.norm(p = 2, dim = 1)$
$\mathbf{W}_{norm} \leftarrow \mathbf{W}/\tilde{\mathbf{d}}_{norm}$ ▷ per-row normalization
$\mathbf{S} \leftarrow \mathbf{W}_{norm} \cdot \mathbf{W}_{norm}^{T}$
$\mathcal{U} \leftarrow \emptyset$ ▷ Set of used vectors
$\tilde{\mathbf{m}}^{\mathbf{g}} \leftarrow$ zeros(n) ▷ Index for inference
$\tilde{\mathbf{m}}^{\mathbf{r}} \leftarrow$ zeros(n) ▷ Index for GBS
$g\_idx \leftarrow 0$
for $i = 0, 1, 2, \ldots, \mathbf{S}.shape(0) - 1$ **do**
  if $i \in \mathcal{U}$ **then**
    continue
  end if
  $\mathcal{U} \leftarrow \mathcal{U} \cup \{i\}$
  $C \leftarrow \{i\}$ ▷ Initialize current group
  while $|C| < g$ and $|\mathcal{U}| < n$ **do**
    $j \leftarrow \arg\max_{j \notin \mathcal{U}} \mathbf{S}[i, j]$ ▷ Find most similar unused vector
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{j\}$
    $C \leftarrow C \cup \{j\}$
    $\tilde{\mathbf{m}}^{\mathbf{g}}[j] \leftarrow g\_idx$ ▷ Store current group
  end while
  $g\_idx \leftarrow g\_idx + 1$
  $\tilde{\mathbf{m}}^{\mathbf{r}}[i \cdot g : (i+1) \cdot g] \leftarrow C$ ▷ Store current group
end for
return $\tilde{\mathbf{m}}^{\mathbf{g}}$, $\tilde{\mathbf{m}}^{\mathbf{r}}$

Our research revealed that bitmap generated by this method does not necessarily align with weight distribution within groups during quantization. To address this issue, we propose GBS algorithm, which finds the most suitable bitmap mask by performing k-means clustering within each group.

During our study of the bitmap mask, we gained the following **insight**: the essence of group quantization in preserving original model performance lies in finding a set of weights with similar values, which is particularly crucial for binary quantization. Since quantization is performed row-wise, the distribution of weight magnitudes within a row is paramount. The method of grouping via a globally optimal threshold clearly fails to account for the data distribution within individual rows.

The most straightforward solution would be to search for a row-wise optimal threshold directly per row. However, experimental results showed this method to be extremely inefficient, taking approximately 79 hours to quantize a Llama2-7b model, rendering it impractical. Additionally, this method is incompatible with the grouped tensors obtained by SimS. SimS algorithm groups tensors based on their direction vectors and requires all tensors within a group to share the same bitmap mask, which means vectors within a group having consistent direction but inconsistent magnitudes. For certain column channels within a group, it's highly likely that some values exceed the optimal threshold while others fall below it. However, this intra-column variation in magnitude does not affect their original distribution within the row. Based on those observations, we conclude that methods determining the bitmap mask solely based on weight magnitude are not well-suited for SimS. Building upon the above factors, we employ the k-means algorithm to determine the weight partitioning. Specifically, We first reorder the $\mathbf{W}_{\mathbf{salient}}$ and $\mathbf{W}_{\mathbf{non-salient}}$ with their own $\tilde{\mathbf{m}}^{\mathbf{r}}$. Then process batches based on group size g to obtain $\bar{\mathbf{W}}_{\mathbf{salient}} \in \mathbb{R}^{\frac{n}{g} \times l \times g}$ and $\bar{\mathbf{W}}_{\mathbf{non\_salient}} \in \mathbb{R}^{\frac{n}{g} \times (m-l) \times g}$. For example, matrix $\bar{\mathbf{W}} \in \mathbb{R}^{\frac{n}{g} \times m \times g}$ is gained from matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$ with group size g by:

$$\mathbf{W}^{r}_{i\cdot} = \left|\mathbf{W}_{\tilde{\mathbf{m}}^{\mathbf{r}}[i]\cdot}\right| \tag{8}$$

$$\mathbf{W}^{\mathbf{b}} = \text{reshape}(\mathbf{W}^{\mathbf{r}}, (\frac{n}{g}, g, m)) \tag{9}$$

$$\bar{\mathbf{W}}_{i,j,k} = \mathbf{W}^{\mathbf{b}}{}_{i,k,j}, \quad \bar{\mathbf{W}} \in \mathbb{R}^{\frac{n}{g} \times m \times g} \tag{10}$$

The GBS algorithm applies a Euclidean distance-based batched k-means clustering to each intra-column element within a group, where n/g denotes batch size, m means the number of samples in k-means, g means feature dimension. The original weights are then partitioned based on clustering labels. We initialize two cluster centers for each batch k: $\mathbf{C}^{(k)} = [\boldsymbol{\mu}_1^{(k)}, \boldsymbol{\mu}_2^{(k)}]^{\top} \in \mathbb{R}^{2 \times g}$, for $k = 1, \ldots, \frac{n}{g}$. For each batch, whose cluster centers are $\boldsymbol{\mu}_j$, $\mathbf{M}_j$ include all $\mathbf{w}_i$ close to $\boldsymbol{\mu}_j$. Objective function of our algorithm is:

$$J = \sum_{k=1}^{\frac{n}{g}} \sum_{j=1}^{2} \sum_{\mathbf{w}_i \in \mathbf{M}_j^{(k)}} \|\mathbf{w}_i - \boldsymbol{\mu}_j^{(k)}\|^2 \tag{11}$$

We update cluster centers t times by:

$$\boldsymbol{\mu}_j^{(k,t+1)} = \frac{1}{|\mathbf{M}_j^{(k,t)}|} \sum_{\mathbf{w}_i \in \mathbf{M}_j^{(k,t)}} \mathbf{w}_i \tag{12}$$

Finally we get two sets of cluster centers $\mathbf{C} \in \mathbb{R}^{\frac{n}{g} \times 2 \times g}$ for salient weights and non-salient weights and cluster labels $\mathbf{M}_{\mathbf{salient}} \in \mathbb{R}^{\frac{n}{g} \times l}$ and $\mathbf{M}_{\mathbf{non-salient}} \in \mathbb{R}^{\frac{n}{g} \times (m-l)}$. We obtain the final bitmap mask $\mathbf{M} \in \mathbb{R}^{\frac{n}{g} \times m}$ by:

$$\mathbf{M} \overset{\text{broadcast}}{\odot} e\_idx = \mathbf{M}_{\mathbf{salient}}, \tag{13}$$

$$\mathbf{M} \overset{\text{broadcast}}{\odot} (\neg e\_idx) = \mathbf{M}_{\mathbf{non-salient}} \tag{14}$$

This design ensures that values with similar magnitudes within each row are assigned the same label as much as possible. After that, we follow ARB-LLM$_{\text{RC}}$ methods to get refined scaler α_{refine} and zero-point μ_{refine}, then we performed residual quantization on the important columns, obtaining $\mathbf{B_1} \in \mathbb{R}^{n \times l}$ and $\mathbf{B_2} \in \mathbb{R}^{n \times l}$, and conducted 1-bit quantization on the remaining columns to get $\mathbf{B} \in \mathbb{R}^{n \times (m-l)}$. More details are included in the Appendix, quantified matrix $\widehat{\mathbf{W}}$ is obtained as below:

$$\mathbf{B}_{\text{refine}} = \text{sign}(\mathbf{W} - \mu_{\text{refine}}), \ \widehat{\mathbf{W}} = \alpha_{\text{refine}} \cdot \mathbf{B}_{\text{refine}} + \mu_{\text{refine}} \tag{15}$$

In terms of calibration time, GBS replaced the previous global search with row-wise k-means clustering and was accelerated by batching. On LLaMA2-7B, ArbLLM takes 11,902s, while MemeBQ takes 12,975s; on LLaMA2-13B, ArbLLM takes 24,924s, and MemeBQ takes 25,196s. The increased time is less than 10%. We quantized the model layer by layer, so a single A800(80GB) is enough.

Method	Block Size	Weight Bits	Extra Bits	1.3B	6.7B	66B
Full Precision	-	16.00	0.00	14.62	10.86	9.34
RTN	-	3.00	0.00	13,337.38	5,797.32	6,126.09
GPTQ	128	3.00	0.00	16.45	11.31	10.55
RTN	-	2.00	0.00	11,272.65	28,363.14	1,165,864.25
GPTQ	128	2.00	0.00	121.64	20.81	46.38
PB-LLM	128	1.70	1.00	239.81	144.25	27.66
BiLLM	128	1.11	1.00	69.05	47.65	12.05
ARB-LLM$_{RC}$	128	1.11	1.00	26.63	14.92	10.30
MemeBQ	**128**	**1.11**	1.00	**19.14**	**13.08**	**9.91**
MemeBQ	128	1.11	**0.50**	27.25	16.29	11.30

Table 2: Perplexity of RTN, GPTQ, PB-LLM, BiLLM, ARB-LLM and our methods on **OPT** family. The columns represent the perplexity results on **WikiText2** datasets with different model sizes.

Memory Cost

For $\mathbf{W} \in \mathbb{R}^{n \times m}$, block size b, group size g, salient columns number l, we introduce $\tilde{\mathbf{m}}^{\mathbf{g}}$ to store group information, each elements in $\tilde{\mathbf{m}}^{\mathbf{g}}$ will be stored as FP8, when values are less than superior limit of FP8. The memory of $\widehat{\mathbf{W}}$ after standard row-wise binarization is as follows:

$$\mathcal{M}_{non-salient} = \overbrace{\lceil m/b \rceil}^{\text{multiple blocks}} \times (\overbrace{2 \times n \times 16}^{\text{FP16 } \alpha \text{ and } \mu} + \overbrace{8 \times n}^{\text{FP8 } \tilde{\mathbf{m}}^{\mathbf{g}}}) + \overbrace{(n/g) \times (m-l)}^{\text{bitmap mask}} + \overbrace{n \times m}^{\mathbf{B}} \tag{16}$$

Moreover, second-order row-wise binarization can be represented as

$$\mathcal{M}_{salient} = \overbrace{\lceil m/b \rceil}^{\text{multiple blocks}} \times (\overbrace{3 \times n \times 16}^{\text{FP16 } \alpha \text{ and } \mu} + \overbrace{8 \times n}^{\text{FP8 } \tilde{\mathbf{m}}^{\mathbf{g}}}) + \overbrace{(n/g) \times l}^{\text{bitmap mask}} + \overbrace{2 \times n \times m}^{\mathbf{B}_1 \text{ and } \mathbf{B}_2} \tag{17}$$

since row-wise μ_1 and μ_2 can be combined together as $\mu = \mu_1 + \mu_2$. We totally reduce $(n - n/g) \times m - m/b \times 8n$ bits compared to ARB-LLM$_{RC}$. Real storage of MemeBQ is 2.22GB while ArbLLM's is 2.69GB.

Experiments

Setup

The experiments of MemeBQ were primarily conducted using PyTorch framework (Paszke et al. 2019) and Hugging Face library (Wolf et al. 2019). As a PTQ approach, MemeBQ requires leveraging datasets for model weight calibration. By deploying the model layer-by-layer on a GPU, even large models like Llama-3-70B can be quantized on a single 80GB NVIDIA A800.

Models and Datasets We conducted extensive experiments on existing open-source models, including the Llama-2 (Touvron et al. 2023) and Llama-3 (Grattafiori et al. 2024) series, as well as a representative subset of OPT (Zhang et al. 2022) models. For the calibration dataset, we selected the more comprehensive C4 dataset (Raffel et al. 2020). Due to the large size of the C4 dataset, we constructed the calibration set by randomly sampling 128 sentences from c4-train.00000-of-01024 located in the en/ subdirectory. To validate the effectiveness of the quantization method, we used the Wikitext2 (Merity et al. 2016) and C4 datasets to construct the validation set for testing model perplexity(PPL), which has been proven to be the most stable metric for evaluating LLMs.

Furthermore, we evaluated the accuracy for 7 zero-shot QA datasets including PIQA (Bisk et al. 2020), BoolQ (Clark et al. 2019), OBQA (Mihaylov et al. 2018), Winogrande (Sakaguchi et al. 2021), ARC-e (Clark et al. 2018), ARC-c (Clark et al. 2018), and Hellaswag (Zellers et al. 2019). The results on these tasks provide robust evidence of the PTQ quantization method. For baseline comparisons, we primarily evaluated against existing binary LLMs like PB-LLM, BiLLM, and ARB-LLM.

Main Results

Comparison results We conducted extensive experiments on major bitmap-using methods under the uniform condition of a 128 block size. The results for OPT models can be found in Table 2. From the table, it can be observed that MemeBQ achieves a new state-of-the-art (SOTA) under the same bitmap size. Notably, on the OPT-66B model, MemeBQ with an overall bitwidth of 2.01 bits even outperforms the GPTQ method using 3 bits. Table 3 lists the results of MemeBQ on the Llama series models. The table shows that MemeBQ significantly outperforms the previous SOTA method ARB-LLM at the same bitwidth. Specifically, MemeBQ surpasses the 3-bit RTN method on both Llama-2-13B and Llama-2-70B. Furthermore, on Llama-3-8B, MemeBQ outperforms the 3-bit GPTQ method and emerges as the new SOTA.Additionally, on the Llama-2-7B model, MemeBQ with 1.58 bits achieves a perplexity (PPL) of 16.73—merely 0.3 higher than ARB-LLM with 2.08 bits. Overall, at the same bitwidth, MemeBQ significantly outperforms previous SOTA methods. Furthermore, while achieving comparable performance, MemeBQ reduces the bitmap

Model	Method	Block Size	Weight Bits	Extra Bits	7B/8B*	13B	70B
Llama-2	Full Precision	-	16.00	0.00	5.47	4.88	3.32
	RTN	-	3.00	0.00	542.80	10.68	7.53
	GPTQ	128	3.00	0.00	6.44	5.46	3.88
	RTN	-	2.00	0.00	17788.93	51145.61	26066.13
	GPTQ	128	2.00	0.00	60.45	24.48	8.18
	PB-LLM	128	1.70	1.00	66.41	236.40	28.37
	BiLLM	128	1.08	1.00	32.48	21.77	13.32
	ARB-LLM$_{RC}$	128	1.08	1.00	16.44	11.85	6.16
	MemeBQ	128	**1.08**	1.00	**8.77**	**7.42**	**4.93**
	MemeBQ	128	1.08	**0.50**	16.73	14.38	7.67
Llama-3	Full Precision	-	16.00	0.00	6.14	N/A	2.86
	RTN	-	3.00	0.00	2194.98	N/A	13592.69
	GPTQ	128	3.00	0.00	18.68	N/A	6.65
	RTN	-	2.00	0.00	1335816.13	N/A	481927.66
	GPTQ	128	2.00	0.00	1480.43	N/A	82.23
	PB-LLM	128	1.70	1.00	73.08	N/A	22.96
	BiLLM	128	1.06	1.00	55.80	N/A	66.30
	ARB-LLM$_{RC}$	128	1.06	1.00	27.42	N/A	11.10
	MemeBQ	128	**1.06**	1.00	**13.38**	N/A	**8.07**
	MemeBQ	128	1.06	**0.50**	32.31	N/A	12.67

The table gives the average bit-width of the Llama family. N/A: Llama-3 do not have 13B version. *: Llama-2 has 7B version and Llama-3 has 8B version.

Table 3: Perplexity of RTN, GPTQ, PB-LLM, BiLLM, ARB-LLM and our method on Llama Family. The columns represent the perplexity results on **WikiText2** datasets with different model sizes.

mask overhead.

Zero-Shot results We conducted comprehensive experiments on 7 zero-shot Question Answering benchmarks using similarly sized models including Llama-2-7B, Llama-3-8B, and OPT-6.7B. The results shown in Table 4 demonstrate that MemeBQ delivers significant improvements: at identical bitwidths, it substantially outperforms prior SOTA ARB-LLM, achieving a 7.27% accuracy gain on Llama-3-8B; while using fewer bits, MemeBQ maintains near-competitive accuracy with only a 0.4% margin on Llama-2-7B. Furthermore, all MemeBQ variants surpass existing SOTA methods on OPT-6.7B by 5.77% and 2.83%. These findings collectively validate MemeBQ's robustness across architectures and bitwidth configurations.

Ablation Study

Effectiveness of modules To validate the effectiveness of individual components within MemeBQ, we conducted an ablation study for each module on the Llama-2-7B model using ARB-LLM as baseline. Table 5 reveals that while the SimS algorithm effectively reduces bitmap mask overhead, it incurs performance degradation (perplexity increased from 16.44 to 21.67)—consistent with the *no free lunch* principle. Crucially, our proposed GBS algorithm compensates for this performance loss. Notably, GBS alone substantially improves model performance (perplexity reduced from 16.44 to 8.77). Ultimately, combining both algorithms, MemeBQ achieves comparable results to prior SOTA (16.73 vs. 16.44 PPL) with reduced memory overhead.

Ablation study on SimS module To achieve optimal effect, we investigated BiLLM algorithm equipped with SimS method on the Llama-2-7B model, with results detailed in Table 6.SimS is designed to minimize loss caused by bitmap reduction. With no prior mask compression baselines, we use merge-to-nearest (MTN) method as the baseline. As in Table 6, shrinking bitmaps without SimS leads to model collapse. Compared to direct MTN approaches, grouping determined by original weight distribution proved significantly effective. We evaluated k-means versus greedy clustering algorithms across datasets, revealing that the greedy method yielded superior results due to k-means' suboptimal centroid distribution for grouping. Furthermore, our exploration of similarity metrics demonstrated higher sensitivity to row vector's directionality than their magnitude. Consequently, we adopted greedy clustering with cosine similarity between row vectors as final grouping strategy.

Groupsize of SimS We systematically evaluated performance on Llama-2-7B model across varying groupsizes defined as the number of row vectors per group in the SimS module. As Table 7 demonstrates, significant performance degradation occurs at groupsize of 0 (bitmap disabled, 0 ex-

Model	Method	Weight Bits	Extra Bits	PIQA ↑	BoolQ ↑	OBQA ↑	WinoG ↑	ARC-e ↑	ARC-c ↑	Hellaswag ↑	Average ↑
Llama2-7B	Full Precision	16.00	0.00	79.11	77.71	44.20	69.06	76.30	46.25	75.99	66.95
	BiLLM	1.08	1.00	60.39	59.42	29.80	51.93	39.98	23.72	35.90	43.02
	ARB-LLM$_{RC}$	1.08	1.00	66.59	66.33	29.60	57.85	51.01	27.56	48.33	49.61
	MemeBQ	1.08	1.00	**72.74**	**68.07**	**34.40**	**64.64**	**62.96**	**31.57**	**55.91**	**55.76**
	MemeBQ	1.08	**0.50**	65.61	66.42	29.40	58.48	51.22	27.30	46.12	49.22
Llama-3-8B	Full Precision	16.00	0.00	80.58	81.10	45.00	72.93	80.22	53.50	79.18	70.36
	BiLLM	1.08	1.00	57.51	58.50	29.20	52.25	33.71	22.70	35.17	41.29
	ARB-LLM$_{RC}$	1.08	1.00	71.11	62.60	33.40	59.27	57.28	28.75	53.54	52.27
	MemeBQ	1.08	1.00	**73.29**	**74.95**	**38.40**	**68.27**	**65.45**	**35.58**	**60.86**	**59.54**
	MemeBQ	1.08	**0.50**	62.84	65.84	26.60	57.46	49.07	26.45	43.27	47.36
OPT-6.7B	Full Precision	16.00	0.00	76.44	66.12	37.40	65.43	65.66	34.73	67.21	59.00
	BiLLM	1.08	1.00	58.60	62.14	13.20	53.12	33.75	18.26	22.83	38.27
	ARB-LLM$_{RC}$	1.08	1.00	72.47	62.87	22.20	**60.62**	59.09	26.79	42.08	49.45
	MemeBQ	1.08	1.00	**73.67**	**63.73**	**36.80**	**60.62**	**61.20**	**31.23**	**59.30**	**55.22**
	MemeBQ	1.08	**0.50**	71.11	62.60	33.40	59.27	57.28	28.75	53.54	52.28

Table 4: Accuracy on 7 data sets, from binarization Llama-2, Llama-3 and OPT, and we also compare the results among BiLLM, ARB-LLM and our method to validate the quantization effect.

Algorithm	Total Bits	SimS	GBS	Wikitext2↓	C4↓
ARB-LLM$_{RC}$	2.08	✗	✗	16.44	20.12
MemeBQ	2.08	✗	✓	**8.77**	**11.42**
MemeBQ	**1.58**	✓	✗	21.67	25.80
MemeBQ	**1.58**	✓	✓	16.73	20.38

Table 5: Ablation study on Llama-2-7B

Method	Mode	Wikitext2↓	C4↓
MTN	-	70,99	48.77
Kmeans	euclidean	39.42	39.14
Kmeans	cosine	37.68	38.51
Greedy Search	euclidean	35.19	36.15
Greedy Search	cosine	**31.91**	**37.24**

MTN: Merge-to-nearest method, which groups tensors with the nearest one

Table 6: Ablation study on **SimS** without **GBS**, Llama-2-7B model and BiLLM algorithm are chosen

tra bit). MemeBQ with groupsize of 1 maintains prior methods' overhead while delivering optimal accuracy. By setting groupsize to 2, MemeBQ achieves comparable performance to ARB-LLM with only 0.5-bit overhead (1.58-bit totally). And MemeBQ with groupsize of 4 matches 2.08-bit BiLLM's accuracy at just 0.25-bit overhead (1.25-bit total). Overall, MemeBQ achieves comparable results to previous algorithms with less bit overhead.

Algorithm	Extra Bits	Groupsize	Wikitext2↓	C4↓
BiLLM	1.00	1	32.48	39.38
ARB-LLM$_{RC}$	1.00	1	16.44	20.12
MemeBQ	0.00	0	377.97	160.08
MemeBQ	0.25	4	33.81	37.74
MemeBQ	0.50	2	16.73	20.64
MemeBQ	1.00	1	**8.77**	**11.42**

Table 7: Ablation study on different groupsize, Llama-2-7B model are chosen

Conclusion

In this paper, we propose MemeBQ, a novel binary PTQ framework that deals with the extra memory overhead problem. By introducing a greedy row similarity search algorithm, MemeBQ groups similar weight tensors to share bitmap mask, reducing extra memory by 50%. The group bitmap search further optimizes quantization fidelity through fine-grained k-means clustering within groups, ensuring alignment with local weight distributions. Extensive experiments demonstrate that MemeBQ achieves comparable accuracy to SOTA methods with half the extra bits and outperforms them under equivalent bit constraints.

Acknowledgments

This work was supported by National Key R&D Program of China (2022ZD0116406), Beijing Natural Science Foundation (L244046), CCF-Baidu Open Fund and the Key Research and Development Program of Jiangsu Province (Grants No. BE2023016).

References

Ashkboos, S.; Croci, M. L.; Nascimento, M. G. d.; Hoefler, T.; and Hensman, J. 2024. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*.

Bisk, Y.; Zellers, R.; Gao, J.; Choi, Y.; et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 7432–7439.

Chee, J.; Cai, Y.; Kuleshov, V.; and De Sa, C. M. 2023. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36: 4396–4429.

Clark, C.; Lee, K.; Chang, M.-W.; Kwiatkowski, T.; Collins, M.; and Toutanova, K. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Dettmers, T.; Lewis, M.; Belkada, Y.; and Zettlemoyer, L. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332.

Dettmers, T.; Svirschevski, R.; Egiazarian, V.; Kuznedelev, D.; Frantar, E.; Ashkboos, S.; Borzunov, A.; Hoefler, T.; and Alistarh, D. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*.

Feng, Q.; Li, W.; Lin, T.; and Chen, X. 2025. Align-KD: Distilling Cross-Modal Alignment Knowledge for Mobile Vision-Language Large Model Enhancement. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 4178–4188.

Frantar, E.; and Alistarh, D. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, 10323–10337. PMLR.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Huang, W.; Liu, Y.; Qin, H.; Li, Y.; Zhang, S.; Liu, X.; Magno, M.; and Qi, X. 2024. Billm: Pushing the limit of post-training quantization for llms. *arXiv preprint arXiv:2402.04291*.

Ko, J.; Kim, S.; Chen, T.; and Yun, S. Y. 2024. DistiLLM: Towards Streamlined Distillation for Large Language Models.

Li, Z.; Yan, X.; Zhang, T.; Qin, H.; Xie, D.; Tian, J.; Kong, L.; Zhang, Y.; Yang, X.; et al. 2024. Arb-llm: Alternating refined binarizations for large language models. *arXiv preprint arXiv:2410.03129*.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Mihaylov, T.; Clark, P.; Khot, T.; and Sabharwal, A. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1): 5485–5551.

Saha, R.; Sagan, N.; Srivastava, V.; Goldsmith, A. J.; and Pilanci, M. 2024. Compressing Large Language Models using Low Rank and Low Precision Decomposition.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9): 99–106.

Shang, Y.; Yuan, Z.; Wu, Q.; and Dong, Z. 2023. Pb-llm: Partially binarized large language models. *arXiv preprint arXiv:2310.00034*.

Tian, J.; Solgi, R.; Lu, J.; Yang, Y.; Li, H.; and Zhang, Z. 2025. FLAT-LLM: Fine-grained Low-rank Activation Space Transformation for Large Language Model Compression. *arXiv preprint arXiv:2505.23966*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yao, Z.; Yazdani Aminabadi, R.; Zhang, M.; Wu, X.; Li, C.; and He, Y. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35: 27168–27183.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.