MiLo: Efficient Quantized MoE Inference with Mixture of Low-Rank Compensators

Beichen Huang [*1,2] **Yueming Yuan** [*1] **Zelei Shao** [*1] **Minjia Zhang** [1]

Abstract

A critical approach for efficiently deploying Mixture-of-Experts (MoE) models with massive parameters is quantization. However, state-of-the-art MoE models suffer from non-negligible accuracy loss with extreme quantization, such as under 4 bits. To address this, we introduce MiLo, a novel method that augments highly quantized MoEs with a mixture of low-rank compensators. These compensators consume only a small amount of additional memory but significantly recover accuracy loss from extreme quantization. MiLo also identifies that MoE models exhibit distinctive characteristics across weights due to their hybrid dense-sparse architectures, and employs adaptive rank selection policies along with iterative optimizations to close the accuracy gap. MiLo does not rely on calibration data, allowing it to generalize to different MoE models and datasets without overfitting to a calibration set. To avoid the hardware inefficiencies of extreme quantization, such as 3-bit, MiLo develops Tensor Core-friendly 3-bit kernels, enabling measured latency speedups on 3-bit quantized MoE models. Our evaluation shows that MiLo outperforms existing methods on SoTA MoE models across various tasks.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable success across various natural language processing tasks, including language understanding, reasoning, and generation (Brown et al., 2020; OpenAI, 2023; 2024a;b). However, further scaling the models poses significant challenges to computational resources and memory consumption (Kaplan et al., 2020; Narayanan et al., 2021; Wang et al., 2023). Mixture-of-Experts (MoE) have emerged as a promising solution. By incorporating sparsely activated expert layers, MoE allows scaling up LLM parameters while maintaining a similar compute requirement (Fedus et al., 2021; Du et al., 2022; Artetxe et al., 2021; Rajbhandari et al., 2022b; Dai et al., 2024; Jiang et al., 2024).

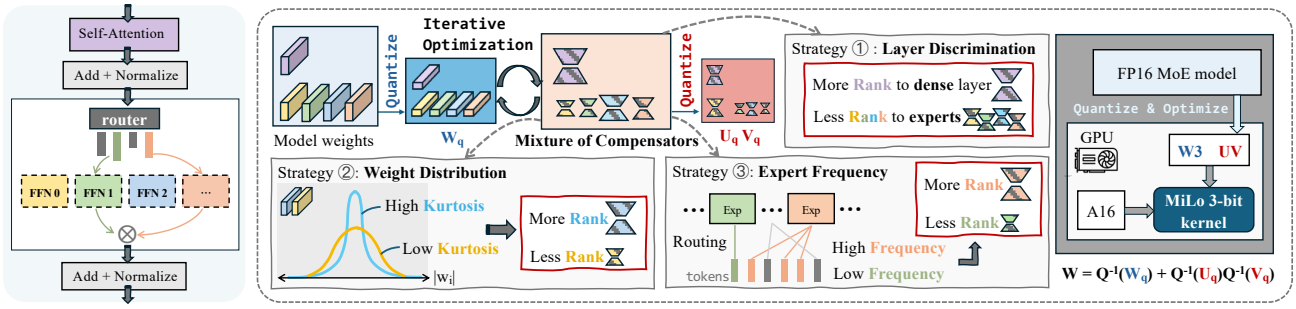
Despite its promising results, MoE models face severe memory challenges that hinder practical deployment. For example, the *Mixtral-8×7B* MoE model (Jiang et al., 2024) requires ∼90GB of memory to just host the model weights in half-precision, while a NVIDIA A100 only has 40/80GB memory. More recent MoEs, such as the Arctic MoE (Snowflake, 2024), further push the MoE boundaries with their massive scale. With a staggering 480B parameters, these MoEs require an immense amount of memory, e.g., close to 1TB, to deploy effectively.

When the memory usage exceeds GPU capacity, the inference of MoEs can resort to offloading (Eliseev & Mazur, 2023) or multi-GPU inference (Rajbhandari et al., 2022a). While these methods help mitigate the pressure on the scarce GPU memory from hosting MoE models, offloading to CPU/NMVe adds non-trivial overhead to inference latency due to limited PCIe bandwidth and multi-GPU inference significantly increases the hardware cost of deploying MoEs.

Among different approaches, model quantization techniques have been demonstrated as a promising technique to compress LLMs (Frantar et al., 2022; AutoGPTQ, 2024; Xiao et al., 2023; Lin et al., 2024). However, applying existing quantization methods to MoE models is hard:

- **SoTA MoE models suffer from non-negligible accuracy loss with extreme quantization, e.g., under 4 bits.** Traditional quantization-aware training is hard to apply to LLMs due to its high training cost (Yao et al., 2022a; Dettmers et al., 2022). Recent post-training quantization works, such as GPTQ (Frantar et al., 2022; AutoGPTQ, 2024) and AWQ (Lin et al., 2024) have demonstrated their effectiveness for dense LLMs towards 4-bit compression. However, further pushing the quantization limit to under 4-bit, e.g., 3-bit, leads to major performance loss. Tab. 1 reports the INT4 quantization time and the Wikitext2 perplexity results from GPTQ and Round-To-Nearest (RTN). INT4 quantization generally leads to a minor loss of accuracy in all

[*]Equal contribution [1]Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, United States [2]Work done while intern at UIUC. Correspondence to: Minjia Zhang <minjiaz@illinois.edu>.



(a) Mixture-of-Experts Structure (b) MiLo

Figure 1: Overview of our MiLo approach. At a high level, MiLo employs a *Quantize-Then-Compensate* approach, which augments low-bit quantized MoEs with a mixture of low-rank compensators, whose ranks are adaptively decided based on the distinctive characteristics of MoE weights. To minimize the accuracy loss, MiLo introduces an iterative optimization algorithm that jointly optimizes quantized MoEs and the mixture of low-rank compensators. MiLo includes a set of hardware-friendly INT3 kernels to achieve high measured speedups.

the settings, but directly applying INT3 quantization to MoE model cannot lead to a satisfying accuracy.

Table 1: Comparison of existing quantization methods.

Wikitext2-PPL↓	Quant-time	FP16	INT4	INT3
	Mixtral-8×7B			
RTN	321s	3.42	3.63	4.81
GPTQ	5315s	3.42	3.63	4.61
	DeepSeek-MoE			
RTN	91s	5.83	6.04	7.32
GPTQ	3355s	5.83	6.02	7.08

- **State-of-the-art methods suffer from calibration data bias and prolonged quantization time, making them hard to apply to MoEs with massive parameters.** Prevailing quantization methods, such as GPTQ and AWQ, rely on calibration data to obtain high accuracy. However, the choice of the calibration data introduces a bias, which causes overfitting during quantization. This is less desirable because recent MoE-based LLMs are still generalist models. Furthermore, calibration requires forward propagation to gain information from input dataset, which is computationally intensive and time-consuming, making it difficult to test on MoE models with massive parameters.
- **Difficulty of converting theoretical savings from extreme quantization to measured speedups for MoEs, especially with INT3 weight-only quantization and batch size >1.** While recent work reported the accuracy of 3-bit quantized MoE (Eliseev & Mazur, 2023; Li et al., 2024a), most of these work do not report latency improvement. To be specific, existing work often does not discuss the weight packing and dequantization cost associated with INT3 quantization scheme, which in fact has a big impact on the performance benefit of using INT3 quantization.

These challenges signify the need for a more advanced optimization method for MoE models. We start from some intriguing observations (§ 3.1.1) that MoE models exhibit distinct characteristics across different weights. In particular, we observe that there are distinct patterns among parameters in non-expert layers and sparsely activated experts, as well as across different experts. Additionally, while INT3 quantization effectively captures outliers, information loss tends to occur at relatively insignificant weight values, motivating error reconstruction methods.

Based on this observation, we propose MiLo to *compress MoEs by augmenting low-bit quantized MoEs with a Mixture of Low-rank compensators*. First, to avoid overfitting and the expensive calibration overhead, we employ a calibration-free quantization algorithm to obtain extreme quantized MoEs, e.g., INT3 MoE. Second, we compensate low-bit quantized MoEs with a mixture of decomposed residual matrices, i.e., the mixture of low-rank compensators, to recover the information loss with a tiny portion of memory overhead. We show that such a mixture of low-rank compensators is quite powerful, which enables an adaptive rank selection strategy based on model structures and data distributions, and can be quantized to further reduce their memory consumption without hurting effectiveness. Thirdly, we enhance quantization performance with an iterative optimization algorithm that jointly optimizes quantized MoEs and their compensators. Finally, we develop hardware-friendly Mixed Precision 3-bit GeMM kernel, using zero-bit-waste INT3 weight packing, binary manipulation based de-quantization, and multi-level pipelining to achieve high measured speedups. Our contributions are:

- We propose MiLo, a novel algorithm that effectively compresses MoE models with INT3 quantization and mixture of adaptive low-rank compensators. MiLo is training-free and does not suffer from calibration bias.
- We propose an efficient INT3× FP16 Mixed Precision GeMM CUDA kernel, for the first time, we demon-

strate that it is possible to allow SoTA MoEs to achieve measured latency 1.3x speedups than SoTA backend MARLIN with batch size > 1.
- We evaluate MiLo on SoTA MoE models Mixtral-8×7B and DeepSeek-MoE, and our evaluation results show that MiLo effectively compresses MoE models with negligible accuracy loss, i.e., recovering over 87% of accuracy on Wikitext2 perplexity with 22% compression ratio. Notably, MiLo achieves up to 3× speedups compared with baseline approaches.

2 Related Works

Post-Training Quantization (PTQ) for LLMs. In a broad taxonomy, there are two setting of PTQ: quantizing both weight and activation (Dettmers et al., 2022; Yao et al., 2022b; Xiao et al., 2023) and weight-only quantization (Park et al., 2022; Dettmers & Zettlemoyer, 2023). We focus on the second setting in this work, as weights are the primary memory bottleneck for MoEs. In this line of work, the state-of-the-art methods, such as GPTQ (Frantar et al., 2022; AutoGPTQ, 2024) and AWQ (Lin et al., 2024), manage to compress dense LLMs to 4-bit without losing much accuracy. However, existing methods often resort to calibration data to minimize the error in layer outputs caused by outliers. Calibration-based methods suffer from over-fitting to the calibration set and long quantization time. More recently, researchers have also explored calibration-free PTQ methods, such as HQQ (Badri & Shaji, 2023). HQQ captures outliers using a hyper-Laplacian distribution with closed-form solutions. However, we show that existing calibration-free PTQ methods fall short in capturing insignificant weight values.

Low-rank methods for LLM compression. Low-rank factorization techniques, i.e. SVD, are applicable to many aspects of LLM compression. ASVD (Yuan et al., 2023) uses activation to identify the salient weight, and decomposes the rest weight to low-rank matrices to compress the model. GFM (Yu & Wu, 2023) aims at decomposing the features. A recent work named LoRC (Yao et al., 2024b) brings the low-rank factorization method to the error matrix between the vanilla weight and quantized weight, and treats it as compensation to the quantization. Different from those efforts, we explore a mixture of low-rank compensators for MoE models by considering their unique characteristics.

MoE compression. Early work on MoE quantization focuses on translation tasks (Kim et al., 2023). Some heuristic strategies of mix-precision quantization for MoE are investigated in (Li et al., 2024a), revealing the bit-sensitivity of different MoE blocks. One extreme example is (Frantar & Alistarh, 2024), which aims at a sub-1-bit compression through algorithm and compression format co-design. However, multiple studies show that even 3-bit quantization hurts MoE model accuracy significantly (Eliseev & Mazur, 2023; Li et al., 2024a). On a separate line of research, researchers have also investigated pruning experts (Chen et al., 2022; Li et al., 2023), which is complementary to MoE quantization.

System support for low-bit quantized LLMs. TensorRT-LLM has the SoTA kernel support for weight-only quantization. However, it only supports weights in INT4 (W4A16) or INT8 (W8A16 and W8A8) (NVIDIA, 2024). In contrast, we provide additional support for W3A16. Additionally, Bitsandbytes supports W8A8 (bitsandbytes, 2024). AWQ has GPU kernel implementation for W4A16 (Lin et al., 2024). Llama.cpp supports 2/3/4/5/6/8-bit quantization on CPUs and GPU (llama cpp, 2024). However, their kernels cannot make effective use of Tensor Cores. More recently, MARLIN (Frantar et al., 2024) kernels have been developed to support W4A16 quantized GeMM calculation by maximizing the usage of different hardware units on NVIDIA GPUs. GPTQ has a basic GeMV W3A16 implementation for batch size 1 (e.g., memory-bound scenarios) using vector intrinsics, e.g., `__hfma2`. But it does not support batch size $>$ 1. To the best of our knowledge, this work is the first that supports W3A16 on Tensor Core with batch size > 1 with measured speedups on MoE models.

3 Methodology

3.1 Rationale

We propose the method motivated by the layer-divergence nature of sparse models and the observation of the low-bit quantization degradation in Mixtral-8×7B (Jiang et al., 2024) and DeepSeek-MoE (Dai et al., 2024).

3.1.1 Observations

Observation 1: Parameter divergence of sparse models. We observe that MoE models exhibit varying characteristics across weights. Since the weights are trained on different amounts of data, the properties of each layer may diverge within a transformer model. For example, Fig. 2 illustrates distinct patterns between the parameters in the attention projection and the expert weights.

(1) *Dense layers differ from sparsely activated ones.* During training, the dense layers and sparse expert layers were fed with different amount of tokens. Dense layers include the attention projections and the dense components (e.g., shared experts) in hybrid architectures. Fig. 2 and Fig. 4 show the case in Mixtral-8×7B (Jiang et al., 2024). The attention weight distributions are more heavy-tailed with outliers along the channel-wise dimension.

This property can be also captured by the *Kurtosis* of the matrix, defined as $K = \frac{\mathbb{E}[(X-\mu)^4]}{\sigma^4}$. Higher Kurtosis indicates a more heavy-tailed distribution, which reflects the number of outliers in a matrix (Li et al., 2024b). Tab. 2 shows the average Kurtosis across weights for different blocks, revealing

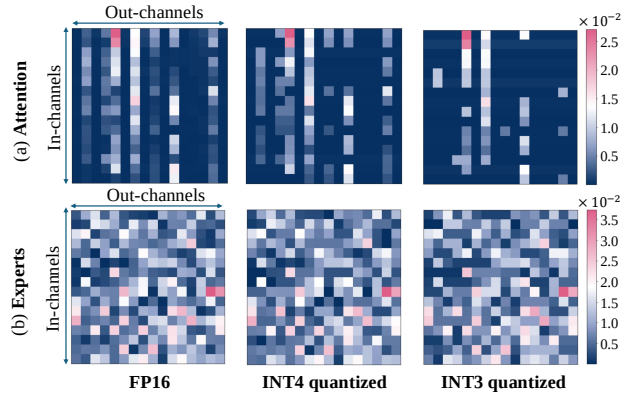


Figure 2: Mixtral-8x7B's (a) weight sampling from *attention* projection and (b) weight sampling from *expert*.

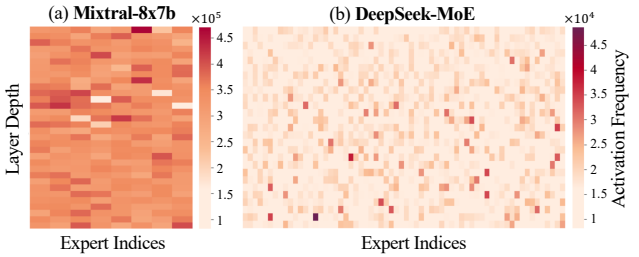


Figure 3: Heatmap of expert activation frequency in Mixtral-8×7B and DeepSeek-MoE on the WikiText-2 task. The vertical axis from top to bottom represents the layer depth, and the horizontal axis represents expert indices.

dense structures have more tail values than sparse layers.

Table 2: Kurtosis and average residual matrix rank across layers and models. The rank is measured by the number of singular values σ_i smaller than $\tau \cdot \sigma_{\max}$, where we use $\tau = 0.5$ in this table. A: Attention projection weights, E: sparse expert weights, SE: shared expert weights in DeepSeek. D represents *densely* activated layers, S represents *sparsely* activated layers.

LAYER	MIXTRAL-8×7B A(D)	MIXTRAL-8×7B E(S)	DEEPSEEK-MOE A(D)	DEEPSEEK-MOE SE(D)	DEEPSEEK-MOE E(S)
KURTOSIS	1.57	-0.53	0.016	0.32	-0.89
RES. RANK	514	1730	438	286	602

(2) *Not all the experts are equal.* Within an MoE layer, the experts are also trained on different subsets of tokens, which cause characterization divergence among experts. Besides, the experts in an MoE layer are not equally activated in identical frequency at all times. As plotted in Fig. 3, the expert frequency diverges, especially for fine-grained settings. In DeepSeek-MoE, The most frequently activated expert is activated 11.7× more often than the least activated expert within the same layer.

Insight. The diverse patterns in MoE pose unique challenges and opportunities for MoE compression, motivating novel approaches to utilize them effectively.

Observation 2: Low-bit quantization's degradation in insignificant weight values. Fig. 2 shows a sampling of unquantized half-precision weight and de-quantized INT3 weights from an expert layer and a self-attention layer in Mixtral-8×7B. Interestingly, the INT3 quantization captures the extreme values, and information loss mainly occurs at relatively *insignificant weight values*. In other words, quantizations capture the outliers adequately while sacrificing the representation of the moderate values as a tradeoff.

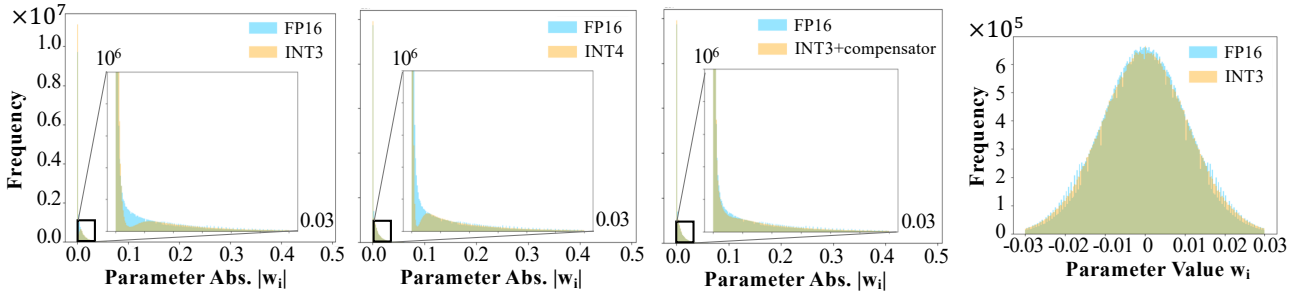
Notably, the layer-divergence also plays a role in this effect. The layers with a high Kurtosis, such as the Attention layer in Mixtral-8×7B, suffer more from low-bit quantization due to their heavy-tailed nature. This effect can be observed more in Fig. 2, where the INT3 quantized weight of attention projection (top right) shows a greater loss of information compared to the expert projection. The residual matrix is an important indicator for analyzing the quantization error. In Table 2, we measure the residual matrix rank (the number of singular values σ_i smaller than $\tau \cdot \sigma_{\max}$) across layers in Mixtral-8x7b and DeepSeek-MoE, where the rank demonstrates negatively correlation to the Kurtosis.

Insight. Extreme quantization is able to capture the outliers in MoE weights at the sacrifice of the expressiveness of insignificant weight values. We need to come up with a method to recover the information loss of those values, with the objective of fully recovering the original FP16 model quality. Ideally, the method should only add slightly more memory while efficiently representing the lost information in the extreme quantization scenario.

3.1.2 *Low-rank Error Construction*

In this work, we consider the solution *residual reconstruction*, which represents the missing information aside from the quantized matrices and corrects quantization errors. Based on the preceding analysis, to complement the quantization that captures the outlier information, we expect a method to estimate the residual that captures *the moderate values* in a matrix well. Also, we require the method to work together with the quantization - they should be optimized together and avoid representation redundancy.

Based on the observations and analysis, we consider *low-rank compensation(LoRC)* (Yao et al., 2024a) as a countermeasure. Low-rank compensation reconstructs the residual of quantization using a low-rank estimation. The weight after compensation is $\tilde{\mathbf{W}}_{LoRC} = Q^{-1}(\mathbf{W}_q) + \mathbf{UV}$, where $\mathbf{W}_q$ is the quantized weight, and $\mathbf{U} \in \mathbb{R}^{m \times r}$, $\mathbf{V} \in \mathbb{R}^{r \times n}$ is optimized to let $\mathbf{UV}$ closely approximate the residual $\mathbf{E} = \mathbf{W} - \mathbf{W}_q$. This optimization utilizes SVD on the residual matrix, where $\mathbf{E} = \mathbf{U}\Sigma\mathbf{V}$, then obtain $\mathbf{U}, \mathbf{V}$ by keeping the largest r singular values in Σ. Fig. 4(c) shows that INT3 plus low rank compensation is able to recover



(a) **Attention** Projection (b) **Expert** Projection

Figure 4: The overlapping region of quantized and half-precision distribution in each figure is shown in green. (a) Information loss analysis for attention layer Mixtral-8×7B. **Left**: INT3 weight quantization captures the outliers adequately but has large information loss at relatively insignificant weight values. **Middle**: INT4 is able to close some of the information gap but not completely. **Right**: INT3 together with low-rank matrices manage to close the information loss gap. (b) Information loss for expert layer at same $|w_i|$ range.

most information loss.

However, directly applying low-rank to an optimized quantization is a suboptimal solution. Since we expect low-rank to represent part of the information, the quantization itself should also be optimized to adapt to the "low-rank residual". Also, specific considerations based on the sparse nature of MoE are required, or the uniform low-rank compensation would introduce a large memory consumption. To solve these problems, we propose the MiLo method.

3.2 MiLo

In previous sections, we analyze the quantization and compensation separately, highlighting the their potential and challenges for compressing MoE models while establishing the building block for the proposed method. In this section, we bring two parts together.

Formally, the goal is to design an algorithm that minimizes the performance drop of MoE models after compression (without fine-tuning) with respect to their uncompressed counterparts. For a pre-trained MoE model $f(x; W)$, we propose to solve the following optimization problem $\mathcal{P}$:

$$\arg\min_{z,s,U,V} \mathcal{L}(W - Q_{z,s}^{-1}(Q_{z,s}(W)) - UV) \quad (1)$$

where $\mathcal{L}$ is loss function, W is the original weight, and U, V are low-rank matrices to approximate the error,i.e., low-rank compensator. Q and Q^{-1} are the quantization operator and de-quantization operator, defined as:

$$W_q = Q_{z,s}(W) = round((W - UV)/s + z) \quad (2)$$

$$W_{dq} = Q_{z,s}^{-1}(W_q) = s(W_q - z) \quad (3)$$

where s and z are the scaling parameter and zero-point for the quantizer, respectively. Meanwhile, the optimization should be subject to the constraint $rank(UV) \leq r$, i.e., UV has a rank that does not exceed a specified threshold r.

3.2.1 *Decomposing the optimization into sub-problems*

The above problem is non-differentiable with combinatorial constraints which cannot be solved with stochastic gradient descent methods (e.g., Adam (Kingma & Ba, 2014)). We present an optimization algorithm, which decomposes the problem $\mathcal{P}$ into two distinct subproblems. *Subproblem 1 (sp1): quantization error minimization*, which aims to reduce the discrepancy between $W - UV$ and its quantized version W_q, and *subproblem 2 (sp2): low-rank compensation maximization*, which focuses on finding low-rank matrices U and V such that UV closely approximate the quantization residual matrix $W - W_{dq}$. We then alternatively solve the subproblems until convergence.

3.2.2 *Optimizing* $\mathbf{W_q}$ *with U,V fixed*

In iteration t of $\mathcal{P}$, we first solve the *sp1* by optimizing the de-quantized weights W_{dq}^t with fixed low-rank matrices U^{t-1}, V^{t-1} from previous iteration. The *sp1* is formulated by applying $l_{p<1}$ norm as the loss function $\mathcal{L}$ and solved as a Lagrange dual problem. Formally, it is described as:

$$\arg\min_{z^t,s^t} \|W - U^{t-1}V^{t-1} - W_{dq}^t\|_{p<1} \quad (4)$$

, where W_{dq}^t is defined as Equation 3. Note that at iteration 0, the matrices U and V are unknown, and are initialized to zero. This initialization serves as the starting point for the iterative optimization.

For simplicity, we fix the scaling parameter s^t and only optimize the zero-point z^t, following the techniques in Half-Quadratic Quantization (HQQ) (Badri & Shaji, 2023). With an auxiliary variable M^t, the optimization problem 4 is:

$$\arg\min_{z^t,M^t} \|M^t\|_{p<1} + \frac{\beta}{2}\|M^t - (W - U^{t-1}V^{t-1} - W_{dq}^t)\|_2^2 \quad (5)$$

Problem 5 can be solved by further applying alternate optimization to update M^t and z^t separately, using Half-Quadratic solver (Geman & Reynolds, 1992) and generalized soft-thresholding operator (Badri & Yahia, 2016). In

each iteration k of $sp1$, we first update M_k^t as:

$$M_k^t \longleftarrow shrink_{l_p}\left(W - U^{t-1}V^{t-1} - W_{dq,k-1}^t), \beta\right) \quad (6)$$

$$shrink_{l_p}\left(x, \beta\right) = sign\left(x\right) relu(|x| - \frac{|x|^{p-1}}{\beta}) \quad (7)$$

And then z_k^t is updated as:

$$z_k^t \longleftarrow \langle W_{q,k}^t - \frac{(W - U^{t-1}V^{t-1} - M_k^t}{s}\rangle \quad (8)$$

$$W_{q,k}^t = round((W - U^{t-1}V^{t-1})/s + z^{t-1}) \quad (9)$$

, where $\langle\cdot\rangle$ represents the average over the axis of the quantization grouping. We choose HQQ to minimize the quantization error due to its low quantization overhead, making it more scalable for large-scale models such as MoE. Moreover, we do not use any calibration data in this process, which avoids calibration data bias. We refer readers to HQQ (Badri & Shaji, 2023) for more detailed steps.

3.2.3 *Solving* $\mathbf{U}, \mathbf{V}$ *with* $\mathbf{W_q}$ *fixed*

With a fixed W_q^t, the problem *sp1* can be viewed as a standard low-rank approximation problem, which is written as:

$$\arg\min_{U^t, V^t} \mathcal{L}(E^t - U^t V^t) \quad (10)$$

, where E^t is fixed as: $W - W_{dq}^t$. In the case of Frobenius norm, the problem is well studied and solved by truncated singular value decomposition, as proved in Eckart-Young-Mirsky Theorem (Eckart & Young, 1936). We first apply SVD to E^t and then update U^t, V^t with a given hyper-parameter rank r as:

$$E^t = \hat{U}\Sigma\hat{V} \quad (11)$$

$$U^t = \hat{U}_{:,1,r}(\Sigma_{1:r,1:r})^{\frac{1}{2}}; \quad V^t = (\Sigma_{1:r,1:r})^{\frac{1}{2}}\hat{V}_{1:r,:} \quad (12)$$

We alternate § 3.2.2 and § 3.2.3 until it reaches a stop condition, where more details about the stop condition are provided in Appendix A.

3.2.4 *Adaptive mixture of low-rank compensators*

Till now we have fixed the choice of the rank r for each low-rank compensator. However, one may wonder whether the sparse nature of MoE architecture leads to more effective and efficient compression. Empirically, increasing the rank improves performance but also increases memory overhead. Rather than applying a uniform rank to all weights, a more effective strategy is to *use higher ranks only where they are most effective*. The property of a matrix that reflects how changes in the rank affect the final performance is referred to as *rank sensitivity* in the following discussion. To that end, we analyze from both MoE structure and matrix property perspectives for DeepSeek-MoE and Mixtral-8×7B, and discuss the rank sensitivity of weights, considering memory constraints. The detailed experiment is provided in § 4.2.

Rank vs. model structures. In MoE models, the sparse-activation mechanism brings dense layers and sparse layers with different characteristics. Since the dense layers are always activated for input tokens, they play a more important role in the model performance. For example, the attention projections are much more rank sensitive than the linear projection with an expert. Therefore, we give *dense layers* more ranks than the sparsely activated layers.

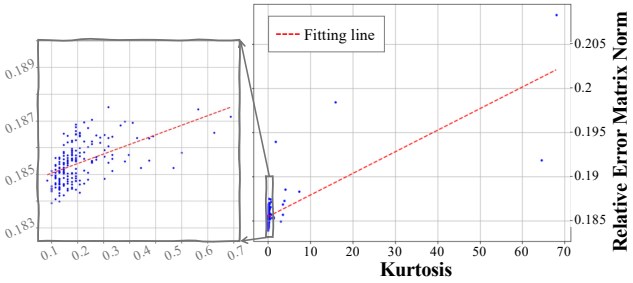


Figure 5: The correlation between relative Frobenius norm vs. Kurtosis. Each dot represents a weight matrix in layer 1 of DeepSeek-MoE.

The importance of each expert also varies according to activation frequency, as experts activated more frequently contribute more significantly to specific tasks. From Fig. 3, we noticed an uneven distribution of expert activation frequencies on Wikitext2 input, particularly among DeepSeek-MoE's fine-grained experts. Therefore, expert frequency is also a good guideline for rank sensitivity.

Rank vs. data distribution. As analysis in § 3.1.1, we noticed that the Kurtosis reflects the outlier distribution, and the heavy-tail distributed weights suffer more in information loss under extreme quantization. Fig. 5 demonstrates the positive correlation between Kurtosis and relative quantization error $||W - W_{dq}||_F / ||W||_F$. We further look into how the low-rank compensator helps in bridging the quantization error. We plot the data distribution of a self-attention layer under INT3, INT4 and INT3+LoRC in Fig. 4(a). Compared with INT3 and INT4 quantization (in left and middle figure), the introduction of low-rank matrices refills the non-outliers, effectively compensating on a heavy-tail distributed weight. And for a weight with lower Kurtosis, as shown in Fig. 4(b), the pattern is not that obvious. Therefore, *higher Kurtosis* indicates a higher rank to bridge the information loss brought by the loss of insignificant weight.

These findings provide the foundation for constructing an adaptive mixture of low-rank compensators. Below, we outline several low-rank compensation policies. While this study primarily evaluates these specific policies, MiLo can readily accommodate a wide range of other strategies.

- **Uniform-{r}**: We set a uniform rank r for all layers, including self-attention layers and expert layers.
- **Dense-{r}**: We only set rank to dense layers, while keeping the rank of sparse layers to 0. For Mixtral-8×7B model, the dense layers are self-attention layers, and for DeepSeek-MoE, dense layers contain self-attention layers, shared-experts, and dense FFN layers.
- **Sparse-{r}**: We assign rank r to sparse activated lay-

ers, i.e. experts, for both Mixtral-8×7B and DeepSeek-MoE models and keep rank to 0 for other layers.

- **Frequency-{r}**: We assign higher rank to experts with higher frequency, and control the average rank to be r.
- **Kurtosis-{r}**: We set higher rank to weights with higher Kurtosis, and control the average rank to be r.

Insight. Overall, we find that dense layers are the most rank sensitive structure and therefore merit higher ranks than sparse layers. The significant benefit is attributed to the fact that dense layers are activated for every token, and thus a higher rank compensator benefits all the inputs. From data distribution perspective, kurtosis and expert frequency work well in different scenarios. For models with balanced experts, e.g. Mixtral-8×7B, Kurtosis is a good indicator of rank. And for those models with unbalanced experts, e.g., DeepSeek-MoE, assigning rank according to expert frequency leads to more performance improvement.

3.2.5 Quantized low-rank mixture compensators

Previous paper has found that the low rank compensation matrices can be quantized to INT8 (Yao et al., 2024b). Following this line, we reinforce this conclusion by showing that the low rank compensation matrices can be quantized to INT3 by symmetric quantization, with minor loss of accuracy. The symmetric INT3 quantization function is:

$$Q_{symm}(W) = round(\frac{7 \times W}{2s}) + 4 \tag{13}$$

where s is the scale factor, equals to the maximum value of the quantization group. Quantizing the low rank matrices to INT3 further reduces the memory overhead brought by the compensator, while retain the accuracy benefit. More results in the evaluation section § 4.2.

Overall, the MiLo algorithm is described in Algorithm 1. When performing MiLo to each weight, we determine the rank r according to the layer structure and data distribution as analyzed in previous section, and perform the optimization until the stop condition is satisfied. The low rank matrices U, V are further quantized to INT3 using symmetric quantization. The outputs of the algorithm are the zero point z and INT3 low rank matrices U, V.

3.3 Hardware-Friendly INT3 Kernel for MoE Inference

As described in § 2, the state-of-the-art kernel implementation for quantized GeMM is MARLIN (Frantar et al., 2024), which supports W4A16. Despite demonstrating promising results, many design choices should be reconsidered when developing efficient W3A16 GeMM kernels. One of the key difficulties lies in INT3 itself: it is not a power of 2, and modern data types typically do not support INT3 values directly. *How should we realize an efficient workflow for INT3 data storage, transfer and calculation of W3A16?*

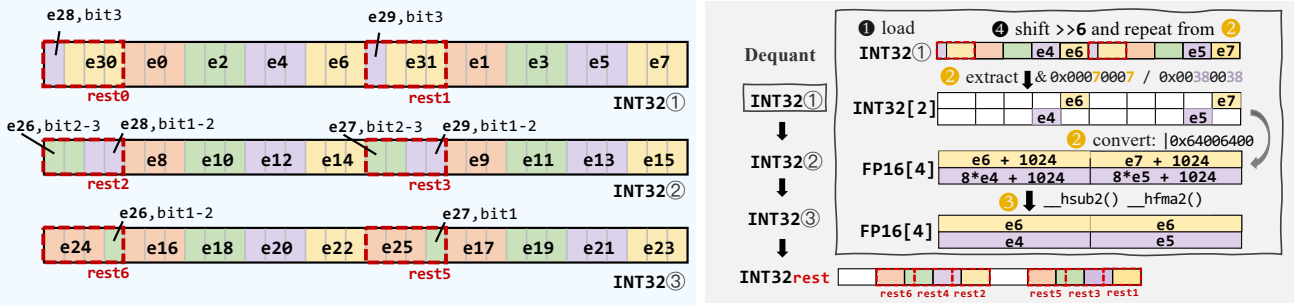
Algorithm 1 MiLo

Input: weight W
Set rank r from model structure or data distribution
Initialize $U_0 = 0, V_0 = 0$
repeat
  // Do quantization to update z
  repeat
    Update M_k^t as Equation (6)
    Update z_k^t as Equation (8)
  until the value of $W - U^t V^t - W_{dq}$ converge
  // Do compensation to update U^t, V^t
  Update U^t, V^t as Equation (12)
  Update the error ϵ_t as Equation (14)
until Stop condition is satisfied
Quantize U, V using Equation (13)
Output: zero point: z; low rank matrices in INT3: U, V

Zero bit waste 3-bit weights packing. We start with the INT3 weight data storage layout. One can pack multiple 3-bit values into a larger data type. For example, one can store ten 3-bit values in an INT32. However, this approach is inefficient as it leaves 2 bits unused. To achieve maximally efficient storage without any bit waste, we choose a packing strategy that fully utilizes each bit. The packing strategy is illustrated in Fig. 6 (a). We group every 32 consecutive INT3 weights and pack them into three INT32 values. In each INT32 we store 8 weights (e.g. e0, e1, ...e7) and some remaining parts(e.g. rest0, rest1). By adding another 3 bit-shift operations and |= (i.e., bitwise OR assignment) operations, we can combine these remaining bits(represents as rest0r̃est6 in the figure) on the boundary into a new INT32 object that also contains 8 weights(i.e. e24, e25,...e31).

In addition, we perform weight reshuffling for every 16 × 64 block of weights, which corresponds to the matrix handled by a single warp to facilitate bulk loading. Weights are managed in groups (32 weights packed into 3 INT32s) to correctly dequantize values like e24 through e31. This requires loading data in units of 3 INT32s, which introduces an alignment issue. To address this, we split the weight matrix into two matrices: the first stores the initial two INT32s, and the second stores the last INT32.

Efficient I2F(INT3-to-FP16) dequantization. Naively applying type-casts from INT3 to FP16 is slow. Inspired by (Kim et al., 2022), we apply *binary manipulations* to efficiently convert INT3 to FP16. Different from that work, which convert INT8/INT4 to FP16, we extend it to convert INT3 to FP16. Moreover, we convert *two* INT3s to FP16s at a time, using register level parallelism, leveraging the fact that 2 FP16 elements can fit in a 32-bit register. The whole procedure for symmetric quantization is as follows, and we use INT32① from Fig. 6(b) as an example: ❶ Load the data into register. ❷ Extract [e6, e7] and [e4, e5] in another two 32-bit registers, and through binary manipulations we



(a) Bit Packing to **INT32x3**

(b) Dequantization from **INT32x3**

Figure 6: Figure (a) shows the zero-bit-waste 3-bit weight packing. Figure (b) shows the dequantization process. The detailed dequantization of INT32(1) is demonstrated.

turn them into [1024 + e6,1024 + e7] and [1024 + 8e4,1024 + 8e5]. ❸ For symmetric quantization, we use `__hsub2` and `__hfma2` to get [e6-4, e7-4] and [e4-4, e5-4], while for asymmetric quantization, we would get [e6, e7],[e4, e5] ❹ Lastly, we do the bit shift operation and repeat steps 2,3 on INT32② >> 6 to get [e0, e1], [e2, e3]. In the scaling step, We use `__hmul2` for symmetric quantization and `__hfma2` for asymmetric quantization.

MoE-specific tile shape tuning. For certain expert layers, e.g., Mixtral-8×7B have both GeMM size 4096×14336 and 14336×4096, the thread synchronization overhead brought by global reduction between thread blocks can be a bottleneck, and changing tile shape cuts down the number of synchronization. Therefore we enable 2 different tile shapes (256, 64) and (64, 256) to improve the performance.

Optimized global to share memory data transfer. In each global-share data transfer step, we need to transfer a matrix tile, which is 64×256 or 256 × 64 in our setting, since each weight costs 3 bit, if we split the work between all threads, each thread would need to transfer 12 bytes. However, it's inefficient in Ampere GPU for 2 reasons: ❶We cannot fully utilize the global-share memory loading bandwidth, since each thread loads 16 bytes of data in a single memory transaction. ❷NVIDIA Ampere architecture introduces `cp.async` instruction `cp.async`, which allows for asynchronous moving data from global memory directly into shared memory, bypassing the L1 cache, which improves the overall compute efficiency. However, `cp.async` is designed to support data types such as INT4 and INT8. To address this, we use 3/4 of the threads and each thread transfer 16 bytes. Specifically, in the case we use 8 warps (can not only hide latency but also get a larger tile size) per SM, we would use warp 0-2 and 4-6 for weight loading. This allows each thread to fully utilize the global-share memory loading bandwidth via 16-byte aligned data loading and `cp.async` to overlap data loading with computation.

INT3 dequantization and matrix multiplication pipelining. Since the dequantization can leverage CUDA core (to do bit operation), and the matrix multiplication takes place on Tensor Core, similar as MARLIN, we construct a mini-pipeline to overlap these operations.

4 Experiment

In this secession, we perform comprehensive experiments to evaluate the proposed MiLo and kernel. A brief implementation description is in Appendix C

Evaluations. The evaluation of MiLo is performed on 6 representative benchmarks, including language modeling(Wikitxt-2 (Merity et al., 2016)), and common sense reasoning (PIQA(Bisk et al., 2020), HellaSwag(Zellers et al., 2019), Lambada (Radford et al., 2019), MMLU(Hendrycks et al., 2020), TriQA(Joshi et al., 2017)). We report the performance on MMLU and TriQA with 5-shot and all others with zero-shot. The average accuracy of zero-shot evaluation is also reported. The evaluation of the kernel of MiLo is performed on 3 different batchsize settings.

Baselines. For the MiLo comparison, we focus on *weight-only grouped* quantization because the memory consumption of MoE models is primarily dominated by the model weights, which also aligns with our motivation. All methods use a quantization group size of 64 for a fair comparison.

- RTN (round-to-nearest), which directly applies PTQ to the MoE model weights.
- HQQ, which is the method introduced in (Badri & Shaji, 2023) that uses half quadratic quantization.
- GPTQ, which is introduced in (Frantar et al., 2022). It employs Hessian information to obtain closed form solutions for weight quantization.

Models. We benchmark our method on two state-of-the-art MoEs Mixtral-8×7B (Jiang et al., 2024) and DeepSeek-MoE (Dai et al., 2024), given that they both achieve high model quality and have severe challenges to deploy on single GPU due to their high memory consumption.

Table 3: Evaluation and Comparison of MiLo.

W3A16	Memory	Wikitext2 PPL↓	HellaSwag↑	Lambada↑	PIQA↑	Avg↑	MMLU↑	TriQA↑
Mixtral-8×7B								
RTN	20.5 GB	4.8133	0.7840	0.7118	0.7910	0.7623	0.5936	0.6941
GPTQ	18.4 GB	4.7304	0.7770	0.7436	0.7954	0.7720	0.6361	0.6853
HQQ	20.5 GB	4.6119	0.7788	0.6974	0.7916	0.7559	0.6093	0.7066
MiLo-s1	20.8 GB	<u>4.0335</u>	**0.8223**	<u>0.7512</u>	**0.8133**	**0.7956**	<u>0.6707</u>	<u>0.7582</u>
MiLo-s2	21.0 GB	**3.9076**	<u>0.8160</u>	**0.7572**	<u>0.8112</u>	<u>0.7948</u>	**0.6769**	**0.7642**
DeepSeek-MoE								
RTN	7.67 GB	7.3295	0.6981	0.6509	0.7829	0.7106	0.3503	0.5000
GPTQ	6.97 GB	6.8234	0.7380	0.6862	0.7791	0.7344	--[1]	0.5461
HQQ	7.67 GB	7.0821	0.7138	0.6667	0.7725	0.7177	0.3563	0.5424
MiLo-s1	7.98 GB	<u>6.4226</u>	<u>0.7460</u>	<u>0.7147</u>	<u>0.7894</u>	<u>0.7500</u>	<u>0.4192</u>	<u>0.5935</u>
MiLo-s2	8.33 GB	**6.2605**	**0.7515**	**0.7217**	**0.7900**	**0.7544**	**0.4197**	**0.5998**

[1] Longer than 24hrs to run

4.1 Main Results

We propose two rank strategies with different memory consumption for both models, marked as s1 and s2, to demonstrate the effectiveness and adaptiveness of MiLo. The rank strategies are detailed in Table 4.

Table 4: Rank strategies for MiLo main evaluation.

		Rank Strategy
Mixtral-8×7B	MiLo-s1	Dense-512 + Kurtosis-16
	MiLo-s2	Dense-1024 + Kurtosis-32
DeepSeek-MoE	MiLo-s1	Dense-800
	MiLo-s2	Dense-1024+Freqency-32

The experiment results are shown in Table 3, where the best results are highlighted in bold and the second-best are underlined. All the settings achieve substantial performance gains with only a slight increase in memory usage. For Mixtral-8×7B, MiLo-s1 improves the average zero-shot accuracy by 10% with just 1.4% additional memory usage compared to HQQ, while MiLo-s2 surpasses GPTQ by 17% in Wikitext2 perplexity. For DeepSeek-MoE, MiLo-s1 delivers a direct message: a simple compensator to dense layers with a small portion of additional memory leads to huge performance improvement. And MiLo-s2 pushes the improvement even forward, reaching 17% of accuracy improvement in MMLU. Both the iterative algorithm and the specialized mixture of compensator strategies drive this remarkable progress, as the low-rank matrices optimization and intrinsic properties of MoE are jointly leveraged and optimized.

4.2 Analysis Results

How does the iterative optimization bring benefits? Generally, the iterative optimization converge the optimization and lead to performance improvement. The error ϵ_t, which is defined in Equation (14), verse iteration is shown in Fig. 10. The Frobenius norm decreases monotonically and converges at around 20 iterations.

Which adaptive rank selection policy works better? In § 3.2.4, we discuss the policies for setting ranks in MiLo. Here, we compare the performance of the rank strategies as shown in Table 7. To focus solely on the rank strategy and eliminate iterative optimization effects, we fix the MiLo iterations to 1. From model structure perspective, we compare the Uniform, Dense, and Sparse strategies, with Dense outperforming the others for both models. Specifically for Mixtral-8×7B, Dense strategy achieves a 9.6% reduction in Wikitext2 perplexity, whereas the other two strategies yield only around a 2% improvement compared to the HQQ baseline. From the perspective of expert data distribution, we fix the rank of dense layer to 512 and compare the strategies of Uniform, Kurtosis, and Frequency. The Kurtosis strategy shows good performance improvement on both models since it captures weights with more outliers and with larger quantization error as analyzed in Fig. 5. Frequency is a fairly good strategy, which brings more improvement to those models with unbalanced expert frequency, e.g. DeepSeek-MoE. These results have demonstrated the importance and opportunities of designing mixtures of adaptive low-rank compensators for a variety of MoE models.

Extra benefits from quantizing the low-rank compensators? Yes. We compare the INT8 and INT3 compensators by evaluating Wikitext2 perplexity on Mixtral-8×7B across a range of rank settings, as shown in Table 5. Compared to INT8 compensators, INT3 only uses 37.5% of memory resulting in just a 0.2% increase in perplexity. Although there are occasional instances where the INT3 compensator causes a notable error surge in individual weights, as measured in Frobenius norm, the overall performance impact remains minimal. Overall, INT3 compensators achieve memory savings with negligible performance loss, aligning well with our motivation and methodology.

Does MiLo add high compression overhead? The INT3 quantization time versus the MMLU for Mixtral-8×7B is plotted in Fig. 8, with MiLo iterations set to 20. As a

Figure 7: Rank Strategy Comparison under Memory Constraint.

Model distribution, Memory constraint = 200MB			Expert data distribution	
Model	Rank Strategy	Wikitext2 PPL↓	Rank Strategy	Wikitext2 PPL↓
Mixtral-8×7B[1]	Uniform-28	4.5262	Uniform-32	4.1645
	Dense-512	4.1683	Kurtosis-32	4.1044
	Sparse-32	4.5986	Frequency-32	4.1698
DeepSeek-MoE[2]	Uniform-22	6.9243	Uniform-16	6.4633
	Dense-512	6.4743	Kurtosis-16	6.3030
	Sparse-24	6.9770	Frequency-16	6.4570

[1,2] Mixtral-8×7B HQQ baseline: 4.6119; DeepSeek-MoE HQQ baseline: 7.0821

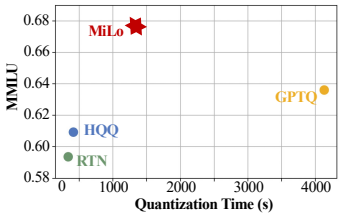


Figure 8: Quantization time vs. MMLU accuracy.

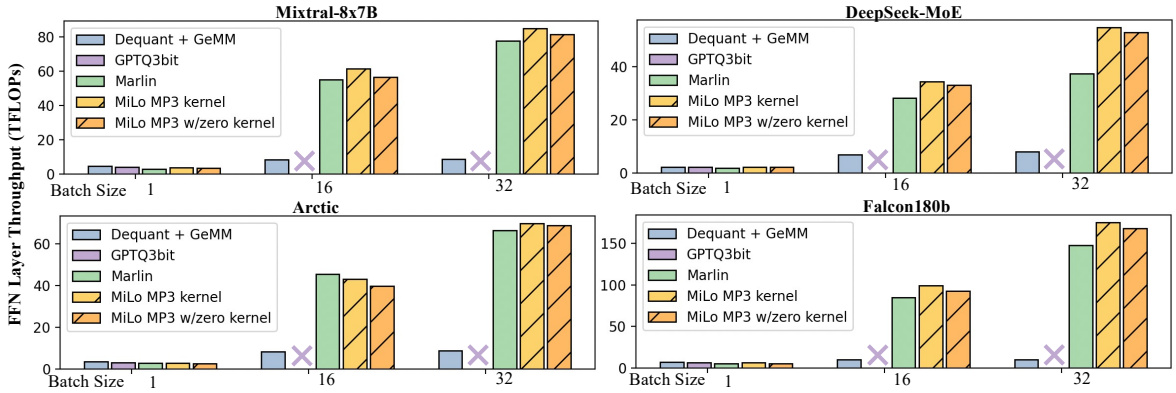


Figure 9: GeMM TFLOPS results on different model FFN layer.

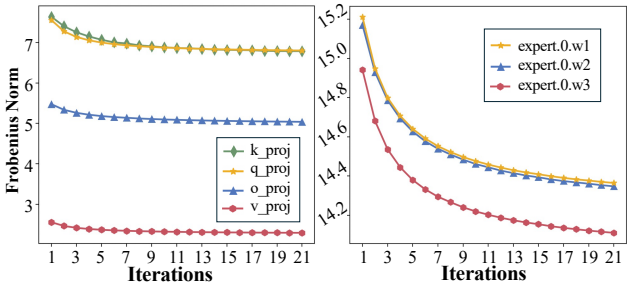


Figure 10: The change of F-norm versus iteration. (Left) The convergence curve of attention matrices. (Right) The convergence curve of expert matrices.

Table 5: INT8/INT3 low-rank compensator results on Wikitext2 PPL for Mixtral-8×7B.

Rank	MiLo Compensator Memory		Wikitext2 PPL↓	
	INT8	INT3	INT8	INT3
16	296 MB	106 MB	4.5014	4.5084
32	525 MB	212 MB	4.4682	4.4786
64	983 MB	424 MB	4.4054	4.4174

calibration-free method, our method gives 3× speedup compared to GPTQ while delivering the best accuracy. Although MiLo is slower than the other two calibration-free methods, HQQ and RTN, it remains within an acceptable timeframe.

4.3 System Performance Results

Our system performance evaluation includes two parts: one end-to-end latency experiments on Mixtual-8x7B, and one mixed precision GeMM TFLOPS experiments on different model's FFN layer. We consider the following baselines: (1) Pytorch, a widely-used deep learning framework, also serves as default backend for inference, we set group size to 64. (2) Dequant + GeMM, we create a strong baseline by using MiLo's 3-bit weight packing and de-quantization method but use CUTLASS for GeMM calculation. (3) GPTQ3bit, which is introduced in (Frantar et al., 2022). It realizes an INT3 × FP16 GeMV, and only support per-channel quantization. (4) MARLIN, which is introduced in (Frantar et al., 2024). It provides a highly optimized INT4 × FP16 GeMM, which only supports per-channel configuration under HQQ.

End-to-end latency. We perform end-to-end latency tests on the Mixtral-8x7B model, with results shown in Tab. 6. The Pytorch baseline runs out of memory easily because the unquantized Mixtral-8x7B exceeds the 40GB memory capacity. GPTQ3bit only supports batch size 1 and throws runtime errors with batch size > 1. MiLo achieves approximately 1.3× speedup over MARLIN across all three batch sizes, because MiLo's 3-bit quantization and its hardware-friendly kernels.

Table 6: Comparison of latency for Mixtual 8×7B.

Kernel Batchsize	1	16	32
PyTorch	OOM	OOM	OOM
GPTQ3bit	0.098	–	–
MARLIN	0.112	0.135	0.138
MiLo	**0.088**	**0.102**	**0.105**

GeMM TFLOPS test. Fig. 9 reports the TFLOPS of different solutions for MLP layers from various models. The results show that MiLo outperforms the other quantized

GeMM kernels when batch size > 1. When the batch size $= 1$, both Dequant+GeMM and GPTQ3-bit achieve better performance results. Dequant+GeMM achieves slightly better performance than GPTQ3-bit at batch size 1, because it uses MiLo's weight packing and dequantization method. These results validate the effectiveness of MiLo's kernel.

5 CONCLUSION

We present MiLo, a novel method that significantly improves the inference efficiency of MoEs, with negligible accuracy loss, using calibration-free quantization and mixture of low-rank compensators. We develop hardware-friendly W3A16 GeMM kernels for compressed MoE models, which delivers real latency reduction. Areas for future exploration include combining MiLo with other MoE compression techniques, such as pruning and distillation.

REFERENCES

Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X. V., Du, J., Iyer, S., Pasunuru, R., Anantharaman, G., Li, X., Chen, S., Akin, H., Baines, M., Martin, L., Zhou, X., Koura, P. S., O'Horo, B., Wang, J., Zettlemoyer, L., Diab, M. T., Kozareva, Z., and Stoyanov, V. Efficient large scale language modeling with mixtures of experts. *CoRR*, abs/2112.10684, 2021.

AutoGPTQ. An easy-to-use LLM quantization package with user-friendly APIs, based on GPTQ algorithm (weight-only quantization), 2024. https://github.com/AutoGPTQ/AutoGPTQ/.

Badri, H. and Shaji, A. Half-quadratic quantization of large machine learning models, November 2023. URL https://mobiusml.github.io/hqq_blog/.

Badri, H. and Yahia, H. A non-local low-rank approach to enforce integrability. *IEEE Transactions on Image Processing*, 25(8):3562–3571, 2016.

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

bitsandbytes. bitsandbytes, 2024. https://github.com/bitsandbytes-foundation/bitsandbytes.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20)*, December 2020.

Chen, T., Huang, S., Xie, Y., Jiao, B., Jiang, D., Zhou, H., Li, J., and Wei, F. Task-specific expert pruning for sparse mixture-of-experts. *arXiv preprint arXiv:2206.00277*, 2022.

Dai, D., Deng, C., Zhao, C., Xu, R., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Dettmers, T. and Zettlemoyer, L. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pp. 7750–7774. PMLR, 2023.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., Zoph, B., Fedus, L., Bosma, M. P., Zhou, Z., Wang, T., Wang, Y. E., Webster, K., Pellat, M., Robinson, K., Meier-Hellstern, K. S., Duke, T., Dixon, L., Zhang, K., Le, Q. V., Wu, Y., Chen, Z., and Cui, C. Glam: Efficient scaling of language models with mixture-of-experts. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569. PMLR, 2022.

Eckart, C. and Young, G. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

Eliseev, A. and Mazur, D. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021.

Frantar, E. and Alistarh, D. Qmoe: Sub-1-bit compression of trillion parameter models. *Proceedings of Machine Learning and Systems*, 6:439–451, 2024.

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

Frantar, E., Castro, R. L., Chen, J., Hoefler, T., and Alistarh, D. MARLIN: mixed-precision auto-regressive parallel inference on large language models. *CoRR*, abs/2408.11743, 2024.

Geman, D. and Reynolds, G. Constrained restoration and the recovery of discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(3):367–383, 1992. doi: 10.1109/34.120331.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de Las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G.,

Lavaud, L. R., Saulnier, L., Lachaux, M., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of Experts. *arXiv preprint arXiv:2401.04088*, 2024.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Kim, Y. J., Henry, R., Fahim, R., and Hassan, H. Who says elephants can't run: Bringing large scale MoE models into cloud scale production. In Fan, A., Gurevych, I., Hou, Y., Kozareva, Z., Luccioni, S., Sadat Moosavi, N., Ravi, S., Kim, G., Schwartz, R., and Rücklé, A. (eds.), *Proceedings of The Third Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pp. 36–43, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.sustainlp-1.6. URL https://aclanthology.org/2022.sustainlp-1.6.

Kim, Y. J., Fahim, R., and Awadalla, H. H. Mixture of quantized experts (moqe): Complementary effect of low-bit quantization and robustness. *arXiv preprint arXiv:2310.02410*, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Li, P., Zhang, Z., Yadav, P., Sung, Y.-L., Cheng, Y., Bansal, M., and Chen, T. Merge, then compress: Demystify efficient smoe with hints from its routing policy. *arXiv preprint arXiv:2310.01334*, 2023.

Li, P., Jin, X., Cheng, Y., and Chen, T. Examining post-training quantization for mixture-of-experts: A benchmark. *arXiv preprint arXiv:2406.08155*, 2024a.

Li, S., Ning, X., Wang, L., Liu, T., Shi, X., Yan, S., Dai, G., Yang, H., and Wang, Y. Evaluating quantized large language models, 2024b. URL https://arxiv.org/abs/2402.18158.

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

llama cpp. llama-cpp, 2024. https://github.com/ggerganov/llama.cpp.

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

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. *arXiv preprint arXiv:2104.04473*, 2021.

NVIDIA. NVIDIA TensorRT, 2024. https://developer.nvidia.com/tensorrt.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

OpenAI. OpenAI GPT-4o API, 2024a. https://platform.openai.com/docs/models/gpt-4o.

OpenAI. Introducing OpenAI o1, 2024b. https://openai.com/o1/.

Park, G., Park, B., Kim, M., Lee, S., Kim, J., Kwon, B., Kwon, S. J., Kim, B., Lee, Y., and Lee, D. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *International Conference on Machine Learning, ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18332–18346. PMLR, 2022a.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18332–18346. PMLR, 2022b.

Snowflake. Snowflake Arctic: The Best LLM for Enterprise AI — Efficiently Intelligent, Truly Open, 2024. URL https://www.snowflake.com/en/blog/

arctic-open-efficient-foundation/-language-models-snowflake/.

Wang, Z., Jia, Z., Zheng, S., Zhang, Z., Fu, X., Ng, T. S. E., and Wang, Y. GEMINI: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP'23)*, October 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. 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*, pp. 38087–38099. PMLR, 2023.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022*, 2022a.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35: 27168–27183, 2022b.

Yao, Z., Wu, X., Li, C., Youn, S., and He, Y. Exploring post-training quantization in llms from comprehensive study to low rank compensation. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024*, pp. 19377–19385. AAAI Press, 2024a.

Yao, Z., Wu, X., Li, C., Youn, S., and He, Y. Exploring post-training quantization in llms from comprehensive study to low rank compensation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19377–19385, 2024b.

Yu, H. and Wu, J. Compressing transformers: features are low-rank, but weights are not! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11007–11015, 2023.

Yuan, Z., Shang, Y., Song, Y., Wu, Q., Yan, Y., and Sun, G. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023.

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

A Stop Condition

We use Frobenius norm to measure the error ϵ_t after each iteration of $\mathcal{P}$. ϵ_t is defined as:

$$\epsilon_t = \|W - W_{dq}^t - U^t V^t\|_F \tag{14}$$

Since this provides an indirect measure of the optimization function, a monotonic decrease in ϵ_t is not guaranteed. In such case, we apply a sliding window average of the error over three iterations, denoted as $\hat{\epsilon}_t$, and stop the iteration if:

$$\frac{\hat{\epsilon}_{t-1} - \hat{\epsilon}_t}{\hat{\epsilon}_{t-1}} < 1e^{-4} \tag{15}$$

In practice, we find that a few tens of iterations (e.g.,20) are sufficient for the optimization to reach a nearly converged output. Based on this, we propose an early-stop strategy, which terminates the algorithm at iteration 20 or stops the process if the error begins to diverge. This early-stop strategy is applied in all the experiments unless stated otherwise.

B Trade-off Between Rank and Performacne Gains

Fig. 11 illustrates the rank-accuracy relationship by comparing memory consumption and Wikitext2 perplexity as rank increases, emphasizing the trade-off between memory overhead and performance gains.

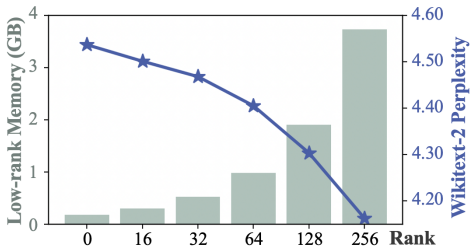


Figure 11: Additional memory consumption from using low rank compensators vs. perplexity varying the rank.

C Implementation

We use `Pytorch` in version 2.4.1+cu121 and `Transformers` in version 4.44.0 to implement our algorithm. The quantization is performed using functions from HQQ library, and low rank compensation is realized using function `torch.svd_lowrank`, which approximates the largest singular values. The algorithm is implemented as described in Algo. 1, and we use the early stop at 20 to terminate the iteration. The following experiments and evaluations are performed using *lm-evaluation-harness*[1]. We conduct experiments using a single NVIDIA A100 GPU with 40GB of memory.

[1] https://github.com/EleutherAI/lm-evaluation-harness