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. The MiLo code is open-sourced on GitHub: `https://github.com/Supercomputing-System-AI-Lab/MiLo`.

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) has 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 an 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.

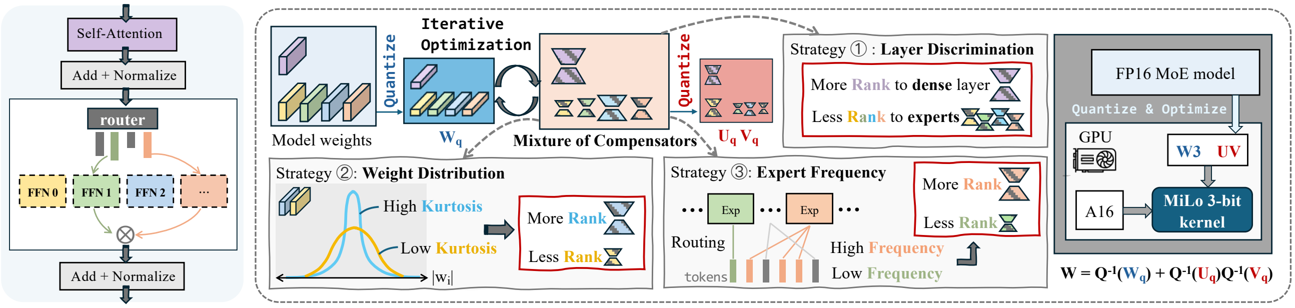
When the memory usage exceeds GPU capacity, the inference of MoEs can resort to offloading (Eliseev & Mazur, 2023) or multi-GPU inference (Rajbhandari et al., 2022a). While these methods help mitigate the pressure on the scarce GPU memory from hosting MoE models, offloading to CPU/NMVe adds non-trivial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

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 quan-

[*] Equal contribution [1] SSAIL Lab, The Grainger College of Engineering, University of Illinois Urbana-Champaign, Urbana, United States [2] Work done while intern at UIUC. Correspondence to: Minjia Zhang <minjiaz@illinois.edu>.

Proceedings of the 8^{th} *MLSys Conference*, Santa Clara, CA, USA, 2025. Copyright 2025 by the author(s).



(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.

tization 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 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 3-bit quantization scheme, which in fact has a big impact on the performance benefit of using 3-bit 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 demonstrate that it is possible to allow SoTA MoEs to achieve measured latency 1.2x 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, which only supports weights in INT4 (W4A16) or INT8 (W8A16 and W8A8) (NVIDIA, 2024), but not in 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 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 amounts 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

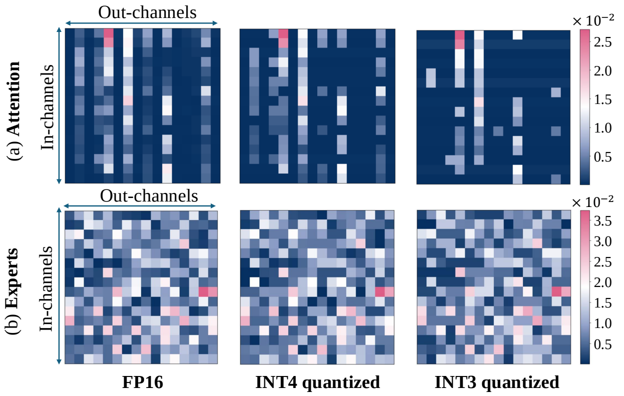


Figure 2: Mixtral-8x7B's (a) weight sampling from *attention* projection and (b) weight sampling from *expert*.

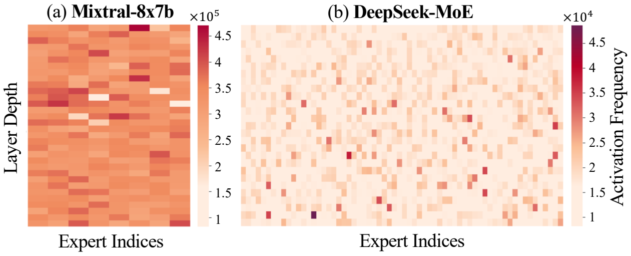


Figure 3: Heatmap of expert activation frequency in Mixtral-8×7B and DeepSeek-MoE on the WikiText-2 task. The vertical axis from top to bottom represents the layer depth, and the horizontal axis represents expert indices.

outliers in a matrix (Li et al., 2024b). Tab. 2 shows the average Kurtosis across weights for different blocks, revealing 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 $\tau = 0.5$. A: Attention projection weights, E: sparse expert weights, SE: shared expert weights in DeepSeek. D represents *densely* activated layers, S represents *sparsely* activated layers.

	MIXTRAL-8×7B		DEEPSEEK-MOE		
LAYER	A(D)	E(S)	A(D)	SE(D)	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 trained on different subsets of tokens, which diverges the characterization among experts. Besides, the experts in an MoE layer are not always equally activated at the same frequency. As plotted in Fig. 3, the expert frequency diverges, especially for fine-grained MoEs. In DeepSeek-MoE, the most used expert is activated 11.7× more 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 un-quantized half-precision weights 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.

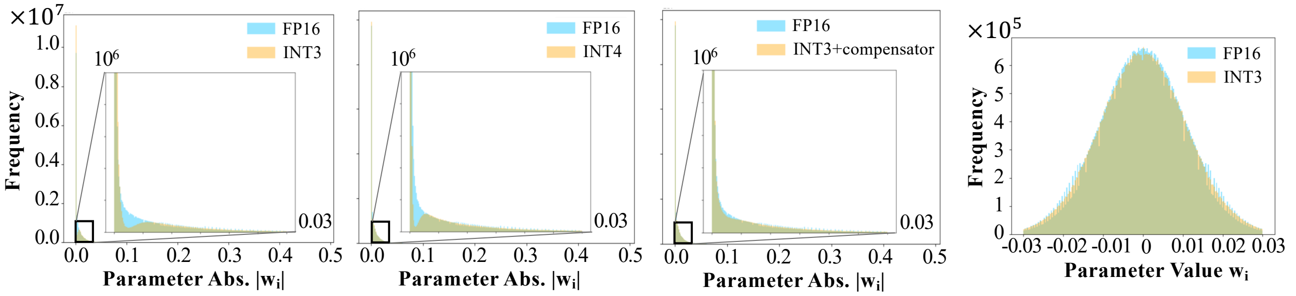
Notably, the layer-divergence also plays a role in this effect. The layers with a high Kurtosis, such as the Attention layer in Mixtral-8×7B, suffer more from low-bit quantization due to their heavy-tailed nature. This effect can be observed more in Fig. 2, where the INT3 quantized weight of attention 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-8×7B and DeepSeek-MoE, where the rank demonstrates negative 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 $\mathbf{UV}$ 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 recovers most information loss.



(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.

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 quantization and compensation separately, highlighting their potential and challenges in compressing MoE models while laying the groundwork for our method. In this section, we bring the 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 $\mathcal{P}$:

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

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:

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

$$W_{dq} = Q^{-1}_{z,s}(W_q) = s(W_q - z) \tag{3}$$

where s and z are vectors of scaling parameters 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 the 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^t_{dq} 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:

$$\arg\min_{z^t,s^t} \|W - U^{t-1}V^{t-1} - W^t_{dq}\|_{p<1} \tag{4}$$

, where W^t_{dq} 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:

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

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^t_k as:

$$M^t_k \longleftarrow shrink_{l_p}\left(W - U^{t-1}V^{t-1} - W^t_{dq,k-1}), \beta\right) \tag{6}$$

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

And then z_k^t is updated as:

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

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

, 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:

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

, 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 hyperparameter rank r as:

$$E^t = \hat{U}\Sigma\hat{V} \tag{11}$$

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

3.2.4 *Stop Condition*

We alternate § 3.2.2 and § 3.2.3 until it reaches a stop condition. We use Frobenius norm to measure the error ϵ_t after each iteration of $\mathcal{P}$. ϵ_t is defined as:

$$\epsilon_t = \|W - W_{dq}^t - U^t V^t\|_F \tag{13}$$

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:

$$\frac{\hat{\epsilon}_{t-1} - \hat{\epsilon}_t}{\hat{\epsilon}_{t-1}} < 1e^{-4} \tag{14}$$

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.

3.2.5 *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.

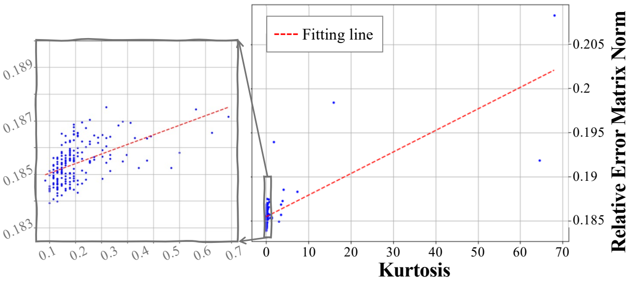


Figure 5: The correlation between relative Frobenius norm vs. Kurtosis. Each dot represents a weight matrix in layer 1 of DeepSeek-MoE.

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.

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 layers, 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.

3.2.6 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:

$$Q_{symm}(W) = round(\frac{7 \times W}{2s}) + 4 \tag{15}$$

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.

Algorithm 1 MiLo

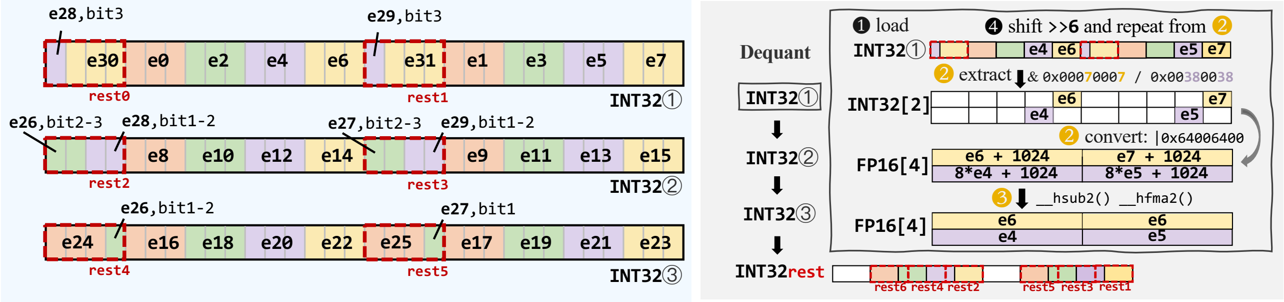
```
Input: weight $W$
Set rank $r$ from model structure or data distribution
Initialize $U_0 = 0, V_0 = 0$
repeat
  // Do quantization to update $z$
  repeat
    Update $M_k^t$ as Equation (6)
    Update $z_k^t$ as Equation (8)
  until the value of $W - U^t V^t - W_{dq}$ converge
  // Do compensation to update $U^t, V^t$
  Update $U^t, V^t$ as Equation (12)
  Update the error $\epsilon_t$ as Equation (13)
until Stop condition is satisfied
Quantize $U, V$ using Equation (15)
Output: zero point: $z$; low rank matrices in INT3: $U, V$
```

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?*

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 rest0, rest1, · · · , rest5 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.



(a) Bit Packing to **INT32**x3

(b) Dequantization from **INT32**x3

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

Efficient I2F(INT3-to-FP16) de-quantization (MiLo Dequant). 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: **1**. Load the data into register. **2**. Extract [e6, e7] and [e4, e5] in another two 32-bit registers, and through binary manipulations we turn them into [1024 + e6,1024 + e7] and [1024 + 8e4,1024 + 8e5]. **3**. 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] **4**. 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.

Asynchronous global weight load. MiLo leverages the asynchronous memory transfer features introduced in NVIDIA's Ampere architecture to efficiently load neural network weights from global memory. By utilizing the `cuda::memcpy_async` API, MiLo performs non-blocking transfers of weights directly into shared memory. This approach eliminates the need for threads or registers to handle the data movement, freeing them for computation. As a result, weight loading can proceed in parallel with ongoing calculations, effectively hiding the latency typically caused by accessing global memory.

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 tile shapes (256, 64), (128, 128) and (64, 256) to improve the performance.

4 EXPERIMENT

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

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, and these results are reported in percentages. The average accuracy of zero-shot evaluation is also reported. The evaluation of the kernel of MiLo is performed on 3 different batch size 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 a single GPU due to their high memory consumption.

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 5.

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	78.40	71.18	79.10	76.23	59.36	69.41
GPTQ	18.4 GB	4.7304	77.70	74.36	79.54	77.20	63.61	68.53
HQQ	20.5 GB	4.6119	77.88	69.74	79.16	75.59	60.93	70.66
MiLo-s1	20.8 GB	4.0335	**82.23**	75.12	**81.33**	**79.56**	67.07	75.82
MiLo-s2	21.0 GB	**3.9076**	81.60	**75.72**	81.12	79.48	**67.69**	**76.42**
				DeepSeek-MoE				
RTN	7.67 GB	7.3295	69.81	65.09	78.29	71.06	35.03	50.00
GPTQ	6.97 GB	6.8234	73.80	68.62	77.91	73.44	--[1]	54.61
HQQ	7.67 GB	7.0821	71.38	66.67	77.25	71.77	35.63	54.24
MiLo-s1	7.98 GB	6.4226	74.60	71.47	78.94	75.00	41.92	59.35
MiLo-s2	8.33 GB	**6.2605**	**75.15**	**72.17**	**79.00**	**75.44**	**41.97**	**59.98**

[1] Longer than 24hrs to run

Table 4: Rank Strategy Comparison under Memory Constraint.

	Model Strategy, memory constraint = 200MB			Sparse Layer Strategy, with Dense Layer rank = 512		
Model	Rank Strategy	Wikitext2 PPL↓	MMLU ↑	Rank Strategy	Wikitext2 PPL↓	MMLU ↑
Mixtral-8×7B[1]	Uniform-28	4.5262	61.58	Uniform-32	4.1645	66.67
	Dense-512	4.1683	65.75	Kurtosis-32	4.1044	67.98
	Sparse-32	4.5986	59.87	Frequency-32	4.1698	66.47
DeepSeek-MoE[2]	Uniform-22	6.9243	37.76	Uniform-16	6.4633	40.45
	Dense-512	6.4743	40.22	Kurtosis-16	6.3030	41.07
	Sparse-24	6.9770	35.93	Frequency-16	6.4570	38.22

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

Table 5: 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+Frequency-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 further, 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 iterative optimization bring benefits? Generally, the iterative optimization leads to performance improvement. The error ϵ_t, which is defined in Equation (13), versus iteration is shown in Fig. 7. The Frobenius norm decreases monotonically and converges at around 10 iterations.

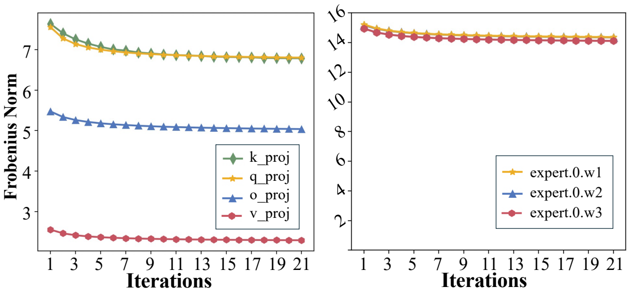


Figure 7: The convergence curve of expert matrices (Left) and attention matrices (Right).

Which adaptive rank selection policy works better? In § 3.2.5, we discuss the policies for setting ranks in MiLo. Here, we compare the performance of the rank strategies as shown in Table 4, and full evaluation results are in Appendix E. To focus solely on the rank strategy and eliminate iterative optimization effects, we fix the MiLo iterations to 1. From the 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 sparse layer's 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 6. 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.

Table 6: 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

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 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.

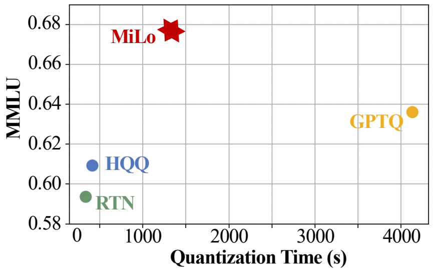


Figure 8: Quantization time vs. MMLU accuracy.

4.3 System Performance Results

We evaluate our system performance through three components: end-to-end latency benchmarking on Mixtral-8×7B, mixed-precision GeMM throughput (TFLOPS) analysis on the MLP layers of various models, and an ablation study to assess the impact of individual optimizations. All the system performance experiments are performed on an NVIDIA A100 GPU with 40GB VRAM.

4.3.1 *End-to-end Performance*

We perform MiLo algorithm on the Mixtral-8×7B model and compare the end-to-end latency using three different backends, and we also bring the un-quantized model as a reference, with results shown in Table 7. We consider the following backend settings: (1) PyTorch, which runs the un-quantized model, aiming at showing the effectiveness and the necessity of quantization and optimized backends. (2) GPTQ3bit Backend, which uses the kernel introduced in (Frantar et al., 2022). It realizes INT3 × FP16 GeMV kernel, which only supports quantized inference with batch size 1 and asymmetric per-channel quantization setting. (3) MARLIN Backend, which uses the MARLIN kernel introduced in (Frantar et al., 2024). It provides a highly optimized INT4 symmetric per-channel quantization. (4) MiLo Backend, which uses the INT3 × FP16 GeMM kernel introduced in § 3.3. MiLo backend supports both symmetric and asymmetric quantization for batched inference and fine-grained quantization. In this comparison, we choose asymmetric quantization and a group size of 64 for MiLo backend, which provides better model accuracy but also adds additional computational overhead.

The PyTorch baseline runs out of memory because the Mixtral-8×7B model takes ∼90GB memory, which exceeds the VRAM of an A100 GPU. GPTQ3bit Backend shows similar behavior with MiLo Backend at batch size of 1, but fails to support larger batch size settings. Compared with MARLIN Backend, MiLo Backend delivers a 1.2× speedup on batch size 1 and 1.26× speedup when batch size larger than 1, thanks to its INT3 quantization and the efficient asymmetric quantization support.

The end-to-end speedup (compared with MARLIN Backend) is better than the corresponding results in GeMM throughput test reported in § 4.3.2, because MiLo algorithm is asymmetric while MARLIN kernel does not inherently support this setting. When integrating MARLIN kernel for this experiment, we need to handle the zero-point calculations separately, which brings extra computation overhead. By fusing the asymmetric de-quantization operation and GeMM into a single kernel, MiLo kernel reduces additional traffic to GPU global memory and brings extra speedups.

Table 7: End-to-end latency for Mixtral 8×7B.

Backend / Batch size	1	16	32
PyTorch	OOM	OOM	OOM
GPTQ3bit Backend	0.102	–	–
MARLIN Backend	0.123	0.141	0.145
MiLo Backend	**0.102**	**0.112**	**0.113**

4.3.2 *Mixed-Precision GeMM Performance*

Fig. 9 shows the TFLOPS achieved by different mixed-precision GeMM solutions for the MLP layers of various

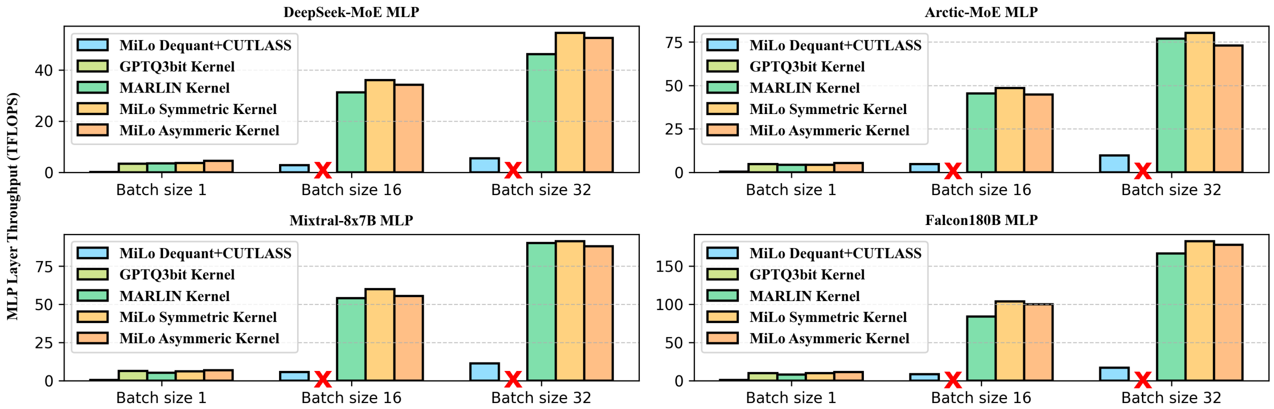


Figure 9: GeMM TFLOPS results on different model MLP layer.

models. The figure compares the following configurations: (1) MiLo Dequant + CUTLASS: It uses two operators. For the de-quantization, it uses MiLo Dequant introduced in § 3.3. For GeMM, it uses CUTLASS. The two operators are not fused. We use symmetric quantization with a group size of 64. (2) GPTQ3bit Kernel: It fuses asymmetric per-channel de-quantization and INT3 × FP16 GeMV, which is introduced in (Frantar et al., 2022). (3) MARLIN Kernel: It is the implementation in (Frantar et al., 2024) with fused symmetric de-quantization of a group size 128 and INT4 × FP16 GeMM. (4) MiLo Kernel: Results of both symmetric and asymmetric kernels described in § 3.3 with a group size of 64 are reported. The detailed shape of the GeMM for this experiment is in Appendix C. When the batch size is 1, both MiLo Symmetric Kernel and GPTQ3bit Kernel achieve the highest throughput. It is because GeMM here is highly memory-bound, and both solutions utilize 3-bit weights for data transfer, reducing memory overhead. For the batch size 16, we observe that MiLo Symmetric Kernel outperforms MARLIN Kernel, by 16%, 7%, 12%, 24% on MLPs of DeepSeek-MoE, Arctic-MoE, Mixtral-8×7B, and Falcon180B, respectively. As the batch size increases to 32, the problem becomes compute-bound. Yet our kernel still demonstrates the highest throughput, with 17% higher than the second best kernel on the DeepSeek-MoE MLP, due to the reduction in global synchronization overhead.

4.3.3 *Ablation Study*

We perform an ablation study on MLP layers using MiLo Asymmetric Kernel, to show the benefit brought by optimizations in § 3.3, including MiLo Dequant, Asynchronized global weight load, and MoE-specific tile shape tuning. The results are shown in Fig. 10, where Baseline represents the MiLo Asymmetric Kernel. The experiment uses 3-bit quantization with a group size of 64, and the batch size is 16. The MLP sizes increase from left to right. We conclude:

(1) Asynchronized global weight load proves to be the most critical component across all models, with its removal resulting in the largest performance degradation, because it overlaps weight loading with computation, significantly reducing pipeline stalls and maximizing GPU utilization.

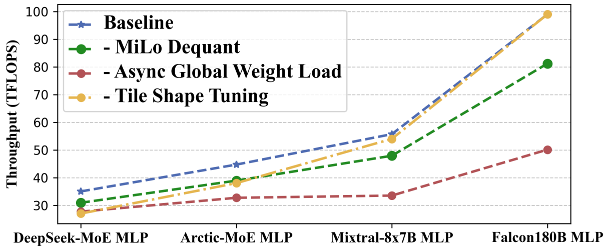


Figure 10: Ablation study of the proposed techniques.

(2) MiLo Dequant becomes increasingly important as the MLP size grows. Once Asynchronous global weight load overlaps memory transfers with computation, the remaining bottleneck shifts to the compute phase. This bottleneck becomes more pronounced in larger models, where MiLo Dequant effectively reduces the associated overhead.

(3) MoE-specific tile shape tuning has a significant impact when working with smaller matrices, such as those found in DeepSeek-MoE MLPs. However, its effect diminishes as the matrix size increases. This observation is consistent with our expectation that, for larger matrices, the relative cost of reduction operations is less significant compared to the dominant compute workload.

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.

Acknowledgement

We sincerely appreciate the insightful feedback from the anonymous reviewers. We thank Niranjan Uma Naresh for helpful discussions. This research was supported by the National Science Foundation (NSF) under Grant No. 2441601. The work utilized the Delta system at the National Center for Supercomputing Applications (NCSA) through allocation CIS240055 from the ACCESS program. ACCESS is an advanced computing and data resource program funded by the U.S. NSF under the Office of Advanced Cyberinfrastructure awards #2138259, #2138286, #2138307, #2137603, and #2138296. The Delta advanced computing resource is a collaborative effort between the University of Illinois Urbana-Champaign and NCSA, supported by the NSF (award OAC 2005572) and the State of Illinois. This work utilized the Illinois Campus Cluster and NCSA NFI Hydro cluster, both supported by the University of Illinois Urbana-Champaign and the University of Illinois System.

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 Trade-off Between Rank and Performance 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.

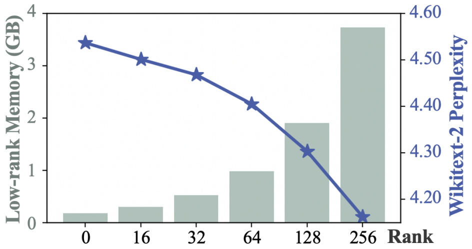


Figure 11: Additional memory consumption from using low rank compensators vs. perplexity varying the rank.

B 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

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

Model	Rank Strategy	Wikitext2 PPL↓	HellaSwag↑	Lambada↑	PIQA↑	MMLU↑	TriQA↑
Model Strategy, Memory constraint = 200MB							
Mixtral-8x7B	Uniform-28	4.5262	79.63	74.01	80.63	61.58	72.15
	Dense-512	4.1683	81.00	72.34	80.84	65.75	74.71
	Sparse-32	4.5968	78.45	71.66	80.03	59.87	70.89
DeepSeek-MoE	Uniform-22	6.9243	72.19	70.77	78.56	37.76	55.83
	Dense-512	6.4743	74.08	73.60	78.40	40.22	58.35
	Sparse-24	6.9770	71.51	66.56	78.12	35.93	54.55
Sparse Layer Strategy (With Dense Layer rank = 512)							
Mixtral-8x7B	Uniform-32	4.1645	81.13	72.07	81.17	66.67	73.95
	Kurtosis-32	4.1044	81.71	74.50	81.22	67.98	76.21
	Frequency-32	4.1698	80.95	77.21	80.84	66.47	75.28
DeepSeek-MoE	Uniform-16	6.4633	73.90	72.92	78.56	40.45	58.41
	Kurtosis-16	6.3030	74.36	72.09	78.29	41.07	60.14
	Frequency-16	6.4570	73.61	69.55	78.89	38.22	58.84

Table 8: Performance comparison of Mixtral-8x7B and DeepSeek-MoE across different rank strategies.

single NVIDIA A100 GPU with 40GB of memory.

C Shape of GeMM in Throughput Tests

Below we list the shapes of the FFN layer matrices for different models used in our GeMM throughput experiments:

Table 9: Matrices' shape in different model's FFN layer.

	DeepSeek-MoE	Arctic-MoE
w1	(2048, 11008)	(7168, 4864)
w2	(11008, 2048)	(4864, 7168)
w3	(2048, 11008)	(7168, 4864)
	Mixtral-8x7B	Falcon180B
w1	(4096, 14336)	(14848, 14848 × 5)
w2	(14336, 4096)	(14848 × 5, 14848)
w3	(4096, 14336)	–

D Correctness Test

To ensure the correctness of our kernel, we conducted a comprehensive series of tests. These included **Functional Correctness Tests** to verify the kernel's basic operations, **Error Handling Tests** to evaluate its robustness against invalid or unexpected inputs, and **Boundary Conditions Tests** to assess its behavior at the extremes of operational parameters. The correctness criterion was defined as achieving a relative error of less than 0.005 across all tests, which were conducted using 5 different random seeds. The results demonstrated that our kernel passed all the tests successfully.

Functional Correctness Tests We evaluated functional correctness using real-world matrices. Specifically, in `test_mixtral_shape()`, we tested 4 different matrix shapes from the Mixtral8x7B model with batch sizes ranging from 1 to 1024. Similarly, in `test_llama_shape()`, we tested 16 different matrix shapes from the Llama2 model, again with batch sizes ranging from 1 to 1024. These tests confirmed that the kernel produced correct outputs across all configurations.

Error Handling Tests We verified the kernel's ability to handle errors under three specific conditions:
1. The group size must be set to 64.
2. The shape of the weight matrix (k, n) must be a multiple of the tile shape $(64, 256)$, $(128, 128)$ or $(256, 64)$.
3. The tile shape configuration must be restricted to $(64, 256)$,$(128, 128)$ or $(256, 64)$.
These tests ensured the kernel could detect and appropriately respond to invalid configurations.

Boundary Conditions Tests Boundary conditions were tested on two dimensions: the batch size and the reduction dimension (the input dimension of the weight matrix).
1. Batch Size Dimension: We focused on scenarios where the batch size is not a multiple of 16, as tensor cores perform 16x8x16 matrix multiplications. In such cases, padding is required to ensure compatibility when the batch size is not divisible by 16.
2. Reduction Dimension: We examined cases where the reduction dimension is not a multiple of 4 * tile_shape[0]. This is because we group 4 tiles into one pipeline calculated by the threadblock, and in these situations, the matrix handled by a threadblock during one pipeline stage terminates early.

These tests confirmed that the kernel performs correctly and efficiently, even under edge cases and challenging scenarios.

E Rank Strategy Evaluations

Table 8 presents a complementary evaluation of rank strategies, using Wikitext2 perplexity, zero-shot and few-shots evaluations. All the evaluations support the previous conclusion that dense layers deserve a higher rank compared with sparse layers, and the Kurtosis value is a good index to identify the sparse layers that deserve a higher rank.

F Artifact Appendix

F.1 Abstract

This artifact description provides a comprehensive workflow for MiLo, which covers how to run the algorithm, rank strategy generation, and evaluation on publicly available benchmarks, along with the quantized kernel settings and evaluation.

We provide instructions on how to obtain, build, and run the software, as well as the necessary steps to reproduce the results presented in the paper. Additionally, the experiment scripts are editable to accommodate further implementations and verifications.

F.2 Artifact check-list (meta-information)

- **Algorithm:** The MiLo algorithm, which employs an iterative optimization to optimize quantized MoE models with a mixture of low-rank compensators.
- **Program:** Python, CUDA
- **Dataset:** For the evaluation, we include Wikitext2, PIQA, HellaSwag, Lambada, MMLU, and TriQA, all of which are publicly available through Huggingface.
- **Hardware:** See F.3.2.
- **Metrics:** The metrics for algorithm include perplexity, accuracy, exact match, memory consumption, and execution time. The metrics for the backend kernel include TFLOPS and execution time.
- **Output:** Quantization time and quantized model memory; Wikitext2 perplexity; zero-shot and few-shots benchmark accuracy.
- **Experiments:** See F.5.
- **How much disk space required (approximately)?:** 100GB
- **How much time is needed to prepare workflow (approximately)?:** 15 min
- **How much time is needed to complete experiments (approximately)?:** Basic experiments (quantization, perplexity, zero-shot tasks) take about 2 hours; Running all the experiments (including few-shot tasks) requires about 12 hours.
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT license

F.3 Description

F.3.1 Code Access

The MiLo algorithm, benchmarks, and scripts are available at Github: `Supercomputing-System-AI-Lab/MiLo`

F.3.2 Hardware dependencies

The MiLo algorithm should be able to execute on Nvidia GPUs with sufficient GPU memory (e.g., 40GB). The MiLO backend kernel is currently only compatible with the NVIDIA Ampere architecture (e.g., NVIDIA A100). For the algorithm, we recommend testing on an NVIDIA A100 GPU with 40GB/80GB memory.

F.3.3 Software dependencies

The software is performed using Python 3.10, and CUDA version 12.4.0. The dependent Python packages can be found in the `requirements.txt` file.

F.3.4 Data sets

The evaluation datasets include Wikitext2, HellaSwag, Lambada, PIQA, MMLU and TriQA, all of which are publicly available and can be downloaded from Huggingface.

F.4 Installation

First, please access the code by

```
$ git clone --branch MiLo-beta https://gith
ub.com/Supercomputing-System-AI-Lab/MiLo.git
```

To better reproduce and avoid capability issues, we recommend using Python 3.10 and CUDA version 12.4.0.

We provide the scripts for the recommended environment setup. Please follow the instructions to create the Conda environment and install the MiLo package & kernels.

```
$ conda create -n milo python==3.10
$ conda activate milo
$ bash conda_env_setup.sh
```

And for the kernel setup, please run the following script:

```
$ bash kernel_setup.sh
```

F.5 Experiment workflow

MiLo quantization algorithm experiments. MiLo provides bash scripts to reproduce the results from the paper. The main results for the quantization and evaluation of Mixtral-s1 in Table 3 can be reproduced by executing the bash scripts provided as:

```
$ cd MiLo
$ bash examples/Mixtral_s1.sh <YOUR_DIR>
```

Please change `<YOUR_DIR>` to your local directory to save the model. This script includes MiLo quantization, evaluation on Wikitext2 perplexity, and zero-shot evaluation. These experiments take around 2 hours. The similar scripts are provided for Mixtral-s2 and DeepSeek-s1 and DeepSeek-s2 in the examples folder.

The few-shot evaluations can be executed using a separate script as:

```
$ bash examples/MiLo_fewshots_eval.sh <YOUR_
DIR> <MODEL_NAME>
```

Please change `<YOUR_DIR>` to your quantized model directory and `<MODEL_NAME>` to ”DeepSeek” or ”Mixtral”. Please note that this might take 10 hours to run on an NVIDIA A100 40GB GPU, and we separate the evaluation for the convenience of testing.

MiLo INT3 kernel experiments. MiLo provides bash scripts to reproduce the results presented in the paper. The kernel GeMM throughput results shown in Figure 9 can be obtained by running the following bash script:

```
$ cd MiLo
$ bash examples/kernel_GeMM_performance.sh
```

Similarly, the kernel end-to-end latency results reported in Table 6 can be reproduced using the following script:

```
$ cd MiLo
$ bash examples/kernel_end2end_latency.sh
```

Due to recent code modifications, some additional overhead has been introduced, leading to slight deviations from the results presented in the paper. To ensure the validity of our work, we also provide results from the state-of-the-art INT4 kernel, Marlin, for comparison. Please note that this might take an hour on an A100 GPU since we need to install MARLIN and quantize the Mixtral-8x7B model to INT4.

F.6 Evaluation and Expected Results

MiLo quantization algorithm experiments. The evaluation scripts will print the quantization time and quantized model memory to the terminal output. The evaluation results will be saved to `<YOUR_DIR>/eval_result.json`.

MiLo INT3 kernel experiments. We provided four scripts for the four kernel experiments in correspondence, as listed below. The expected results are included within the scripts.

1. GeMM correctness results under different settings. The correctness is checked by default. An assertion error will be triggered on incorrect output.
2. GeMM throughput results.
3. Customized GeMM throughput results. This test supports the group size 64 quantization setting and `tile_shape` (64, 256), (128, 128) and (256, 64).
4. End-to-end first token latency. The results correspond to Table 7 in the main text of the paper.

F.7 Experiment customization

MiLo quantization algorithm experiments. By editing the bash script, you can experiment with customized quantization configurations to examine the quantization algorithm.

For example, you can modify the launching command as below to quantize the Mixtral-7x8b model with a uniform rank of 32:

```
python utils/MiLo_quant_main.py
    --base_dir <YOUR_DIR>
    --model_id Mixtral
    --dense_rank 32
    --sparse_rank 32
```

MiLo INT3 kernel experiments. Here we show how to test the MiLo INT3 kernel with varying matrix configurations.

For example, you can launch the example script as below to collect the GeMM results using batch size 16, weight output dimension 7168, weight input dimension 2048, tile shape (128, 128):

```
$ bash examples/kernel_custom_GeMM.sh
    --batch_size 16
    --weight_output_dimension 7168
    --weight_input_dimension 2048
    --tile_shape 128,128
```

F.8 Methodology

Submission, reviewing and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging