MIXLLM: LLM QUANTIZATION WITH GLOBAL MIXED-PRECISION BETWEEN OUTPUT-FEATURES AND HIGHLY-EFFICIENT SYSTEM DESIGN

Zhen Zheng [1] **Xiaonan Song** [1] **Chuanjie Liu** [1]

ABSTRACT

Quantization has become one of the most effective methodologies to compress LLMs into smaller size. However, the existing quantization solutions still show limitations of either non-negligible accuracy drop or low system efficiency. In this paper, we propose MixLLM that explores the optimization space of mixed-precision quantization between output features, based on the insight that different features matter differently in the model. MixLLM identifies the important output features in the global view rather than within each single layer, effectively assigning larger bit-width to output features that need it the most to achieve high accuracy and low memory usage. We present the sweet spot of quantization configuration of algorithm-system co-design with high accuracy and system efficiency. To address the system challenge, we design the two-step dequantization to make use of the Tensor Core easily and fast data type conversion to reduce dequantization overhead, and present the software pipeline to overlap the memory access, dequantization and the MatMul to the best. Extensive experiments show that with only 10% more bits, the perplexity increase can be reduced from about 0.5 in SOTA to within 0.2 for Llama 3.1 70B, while MMLU-Pro loss can be reduced from 1.92 to 0.99 over the SOTA of three popular models. Besides its superior accuracy, MixLLM also achieves state-of-the-art system efficiency. Code is released at https://github.com/microsoft/MixLLM.

1 INTRODUCTION

Large language models (LLMs) (Bubeck et al., 2023; Meta, Cited 2024) have shown remarkable performance on various tasks. But their large memory consumption and massive computation cost have become an obstacle for the efficient deployment (Xia et al., 2023; 2024). Quantization has become one of the most effective solution to compress LLMs into smaller size (Frantar et al., 2022; Lin et al., 2024; Xiao et al., 2023; Yao et al., 2022), by representing the weight or activation with smaller bit-width. However, the existing quantization solutions still show limitations of either non-negligible accuracy drop or system inefficiency.

There is a triangle of characteristics for LLM quantization: *accuracy*, *memory consumption* of parameters, and *system efficiency* of execution, The existing quantization solutions have different focus and trade-off in the triangle: 1) The weight-only method targets to solve the memory consumption problem, and can speedup the small-batched decoding execution that faces the memory-wall problem (Xia et al., 2023; Kim et al., 2024). But their accuracy drop of 4-bit quantization can be a challenge for the production workloads sensitive to accuracy, as illustrated in recent studies (Wu et al., 2023; Kumar et al., 2024). Besides, the weight-only method can lead to system performance drop for large-batched workloads due to the dequantization overhead. 2) The weight-activation quantization represents the activation with low-bit values along with the weights, potentially lead to higher system efficiency. But it can lead to larger accuracy drop than the weight-only method as the activation is usually harder to quantize (Zhao et al., 2024; Ashkboos et al., 2024; Lin et al., 2025). Besides, it introduces more dequantization overhead for the activation that can hurt the system efficiency.

Contributions. In this paper, we provide an extensive analysis of the general quantization principles, and propose MixLLM with the following contributions:

► **High accuracy with low memory consumption: mixed-precision between output features on the weight, with global salience identification.** Given that different neurons matter differently to the model's output, we use different bit-width for different output features (i.e., output channels) for the weight quantization (Fig.1). Rather than using a uniform number of outliers within each layer according to the estimated salience w.r.t. each single layer (Zhao et al., 2024; Huang et al., 2024), MixLLM identifies the salience of different output features globally according to the estimated loss to the model's output. This is because

[1]Microsoft. Correspondence to: Zhen Zheng <zhengzhen@microsoft.com>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

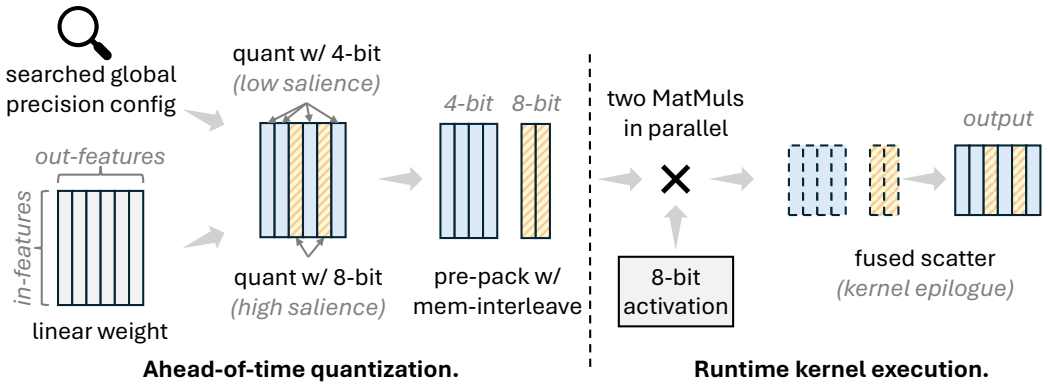

Figure 1: Illustration of the quantization with mixed-precision between output features and kernel execution.

different layers can have different importance to the model. Besides, the mixed-precision between output features makes the system design easier than between input features because the calculation of different output features are disjoint sub-problems.

► **High accuracy with good system efficiency: the co-designed quantization configuration and GPU kernel optimization.** We observe the sweet spot of several quantization decisions to achieve both good accuracy and system efficiency. MixLLM uses 8-bit for activation quantization as it can retain a good accuracy. Besides, MatMul execution tends to be bound more on the larger weight tensor rather than the smaller activation tensor, which weakens the need to push the activation smaller (refer to Sec.3.1). MixLLM uses symmetric quantization for 8-bit and asymmetric for 4-bit for good accuracy, both in group-wise manner. Such configuration makes it challenging to achieve good system efficiency. We design the two-step dequantization to enable using fast int8 Tensor Core for such configuration, along with the fast integer-float conversion to reduce the dequantization overhead. We make the software pipeline design to tackle the system challenge of group-wise and 4-bit asymmetric quantization, and present the state-of-the-art group-wise int8 quantization GPU kernel, supporting asymmetric 4-bit and symmetric 8-bit weight. We also provide the quantization-aware Roofline analysis in this paper.

Extensive evaluation shows that, with 10% 8-bit and 90% 4-bit (i.e., W4.4A8), MixLLM significantly outperforms SOTA quantization algorithms while achieving the state-of-the-art system efficiency on both A100 and H100 GPUs.

2 Background, Related Work, and Discussion

Quantization maps the tensor X into the target range with smaller bit-width through $X_q = clamp(\lfloor \frac{X}{s} \rceil + z, range)$, where s is the scale and z is the zero point. The scale and zero point can be calculated from the whole channel/token vector or a smaller group. The group-wise scheme results in higher accuracy but requires more complex GPU kernel design. The symmetric quantization uses 0 as the zero point, which simplifies the computations and enables many works to design the per-channel/per-token quantized kernels by multiplying the scales at the epilogue of the whole MatMul (matrix multiplication) for dequantization (Xiao et al., 2023; TensorRT-LLM, Cited 2024). However, it usually leads to larger loss than the asymmetric one, especially for smaller bit-widths like 4-bit.

2.1 Related Work and Discussion

This paper focuses on pure post-training quantization (PTQ), without the training on the weight (i.e., QAT) or the quantization parameters (e.g., training the rotation matrix like in SpinQuant (Liu et al., 2024)).

Systems that affect the quantization requirement. The continuous batching technology (Yu et al., 2022) enables to batch the decoding tasks from different requests together to enlarge the batch dimension of MatMul during LLM inference. The chunked-prefill method (Holmes et al., 2024; Agrawal et al., 2024; Zheng et al., 2024) advances the continuous batching by merging the prefill and decoding tasks into the same batch, further enlarging the MatMul shapes. These technologies pushes many LLM jobs to become compute-bound and motivate the demand to reduce computation.

Weight-only quantization and its limitation. There emerges a wide range of technologies to improve the accuracy of weight-only quantization. GPTQ (Frantar et al., 2022) advances OBC (Frantar & Alistarh, 2022) on OBS-based (Hassibi et al., 1993) weight compensation with blocked updating and reordering. AWQ (Lin et al., 2024) proposes to scale the weight according to the characteristic of activation. OmniQuant (Shao et al., 2024) proposes the learnable scaling and weight clipping factors. SpQR (Dettmers et al., 2024), SqueezeLLM (Kim et al.,

2024) and OWQ (Lee et al., 2024) separate the outliers from the quantiation and with half precision. QuIP (Chee et al., 2023) and QuIP# (Tseng et al., 2024a) aim to achieve extremely low-bit quantization with incoherence processing, QTIP (Tseng et al., 2024b) also leverages incoherence processing. AQLM (Egiazarian et al., 2024) and PV-Tuning (Malinovskii et al., 2024) target extremely low-bit quantization. SliM-LLM (Huang et al., 2024) focuses on extremely low-bit quantization using intra-layer mixed-precision. ZeroQuant(4+2) (Wu et al., 2023) and Quant-LLM (Xia et al., 2024) aim to improve accuracy with FP6 quantization.

The weight-only quantization does not reduce the computation but introduces the extra dequantization operations, as it requires to dequantize the weight into float16 for execution. The current weight-only quantization faces two challenges: 1) From the accuracy aspect, there is still an accuracy gap between the 4-bit quantization and the float16 model, especially for many real business scenarios sensitive to the small accuracy drop, as discussed in the recent works (Wu et al., 2023; Xia et al., 2024). 2) It can lead to system efficiency drop on busy servers as the recent LLM inference serving systems will usually batch the processing of different requests together on the server and form large MatMuls. The large MatMuls are compute-bound and will suffer from the dequantization overhead (Lin et al., 2025).

Weight-activation quantization and the challenges. The weight-activation quantization helps to make use of the low-bit computing unit. LLM.int8() (Dettmers et al., 2022) observes the activation outlier problem and separates outliers from quantization with half precision. ZeroQuant (Yao et al., 2022) proposes the per-token activation quantization and group-wise weight quantization. SmoothQuant (Xiao et al., 2023) addresses the activation outlier problem through smoothing, and AffineQuant (Ma et al., 2024) proposes the general affine transformation for quantization. RPTQ (Yuan et al., 2023) reorders the channels to cluster similar scaled values together. SpinQuant (Liu et al., 2024) and QuaRot (Ashkboos et al., 2024) leverages matrix rotation properties to alleviate the outlier phenomenon. Atom (Zhao et al., 2024) uses the mixed-precision between input features to improve accuracy of 4-bit activation quantization. QServe (Lin et al., 2025) is a holistic solution of W4A8 quantization.

Even though the weight-activation quantization has the advantage of reduced MatMul computation (i.e., smaller bit-width computation with higher throughput), it faces the challenge of accuracy drop caused by the activation quantization. The SOTA low-bit weight-activation solutions (Ashkboos et al., 2024; Liu et al., 2024; Lin et al., 2025) still have a gap to the 4-bit weight only quantization. Besides the accuracy drop, the activation quantization will introduce more dequantization overhead than the weight-only one, which makes it challenging to design efficient GPU kernels.

When enabling the asymmetric quantization, the result of $(X_q - z)$ may exceed the range of the bit-width of X_q, making it hard to use the corresponding Tensor Core computation. Systems like Atom (Zhao et al., 2024) thus avoid using the asymmetric quantization, with the cost of larger accuracy drop. The integer quantization requires integer-to-float (I2F) conversion to apply scales. However, the I2F instruction is more expensive than the common computations on modern GPUs (Abdelkhalik et al., 2022) and can lead to large system performance drop for group-wise quantization ($> 10\%$ drop in our practice). Besides, the throughput of Tensor Core is much higher than SIMT Cores, 624 TOPS of int8 Tensor Core vs. 19.5 TFLOPS/TOPS of FP32/INT32 SIMT Cores on A100 GPU. There lacks a well designed software pipeline to overlap the Tensor Core computation and SIMT Core based dequantization for the group-wise and asymmetric weight-activation quantization.

Performance challenge of the float16 outlier separation Outlier separation with half precision works to improve the accuracy while using small bit-width for the non-sensitive weights (Kim et al., 2024; Dettmers et al., 2024), by separating the outliers into an extra sparse tensor in float16. However, it is hard to achieve the peak performance due to the inefficiency of the sparse computation on the GPU, especially when the batch size is large and the linear layer becomes compute-bounded. (As discussed in Flash-LLM (Xia et al., 2023), the hardware utilization can be lower than 10% for the sparse MatMul, while its dense counterpart can usually achieve more than 60%.) This is because the unstructured tensor computation cannot make use the fast Tensor Core easily, but has to use the SIMT Core in float16 for computation and float32 for accumulation[1]. Moreover, sparse computing makes it more difficult to fully utilize the hardware due to the non-continuous memory pattern and the extra index computation.

3 METHODOLOGY

3.1 Quantization Design and Decision in MixLLM

We make the following design and decision to optimize the quantization algorithm and system.

3.1.1 Global Mixed-precision

Different elements of the weight show different salience to the network's loss when being quantized (Kim et al., 2024; Dettmers et al., 2024). The outlier separation method can improve the accuracy by using float16 for high-salience el-

[1]Flash-LLM (Xia et al., 2023) optimizes the unstructured sparse MatMul, but can only speedup the small-batched scenarios.

ements, but can suffer from the inefficient sparse MatMul. We observe that the elements with high salience tend to show distribution along the output channels for most of the linear layers in many LLMs. Based on this observation, we can assign larger bit-width to the output channels of high salience, and smaller bit-width to the others. Through the experiments, we get the same conclusion with the existing works (Kim et al., 2024; Dettmers et al., 2024) that there is only a small set of elements with high salience to quantization. Thus we only need to assign the large bit-width to a small portion of the output channels to achieve good accuracy while retaining a small memory consumption.

The structured mixed-precision between output channels can be friendly to the system efficiency and kernel development, due to the nature that different output features are disjoint in the MatMul and the computation of them are different sub-problems. Fig.1 illustrates the mixed-precision quantization and execution in MixLLM. It divides the linear into independent sub-problems, and finally gathers the output of the sub-problems together to form the result. This optimization space is orthogonal to the existing quantization methodologies and can be applied together with them.

One critical problem is how to identify the high-salience output channels in the model. The fixed threshold (Dettmers et al., 2024) or the fixed number/ratio (Zhao et al., 2024; Lee et al., 2024) of high salience elements computed by the local loss of layers can be sub-optimal to the end-to-end model, as different layers show different importance to the model's final output (Gromov et al., 2024; Men et al., 2024; Dong et al., 2019). A high salience channel w.r.t. a layer may not be a high salience channel of the end-to-end model. In MixLLM, we compute the high salience channels globally according to their impact to the model's final loss (Sec.3.2). As a result, different layers will have different number of high salience channels. Fig.2 shows the distribution of the top 10% high-salient out features in Llama 3.1 8B, showing huge difference between different layers.

Note that this design is different from the mixed-precision in Atom (Zhao et al., 2024) from two aspects. 1) MixLLM first addresses the problem of identifying the high-salience channels globally rather than locally. 2) MixLLM applies the mixed-precision between output features rather than input features, which is more system/algorithm flexible as the output features are disjoint naturally. It also differs from SliM-LLM (Huang et al., 2024) which also only considers the local loss to determine the mixed precision and does not focus on the system performance problem.

3.1.2 Quantization Decision

MixLLM makes the same decision with QServe (Lin et al., 2025) on activation quantization to use 8-bit, as the 4-bit activation can lead to a large accuracy drop but does not lead to significant system efficiency improvement as MatMul execution tends to be bound more on the larger weight tensor rather than the smaller activation tensor according to the compute intensity $I = \frac{2MNK}{MKB_{act}+KNBweight}$ (M is the token number, K/N are the in/out features, and B_{act} and B_{weight} are the bytes per element of activation and weight).

Instead of using per-token activation quantization, MixLLM uses group-wise RTN method. On the one hand, Tab.1 shows that simple group-wise RTN quantization outperforms token-wise smoothing method. On the other hand, the weight is already group-wise in MixLLM, and the group-wise activation does not lead to significantly more dequantization overhead in the system. We observe symmetric quantization is enough for the 8-bit activation (refer to MixLLM W8A8 in Tab.1), while asymmetric is essential for the 4-bit weight. The group-wise method with asymmetric can lead to difficulty for the kernel to make use int8 Tensor Core, for which we design a *two-step dequantization* method with the property of the mix of symmetric and asymmetric (Sec.3.3).

3.2 Global Precision Search Algorithm

MixLLM determines the precision of all output features in all layers globally. It identifies the salience of these features with respect to the final loss of the model, and assigns larger bit-width to the features leading to larger loss. Specifically, it calculates the salience S of a channel c as:

$$S_c = |l(c_q) - l(c_0)| \tag{1}$$

which is the distance of the model's loss between quantizing and not quantizing channel c. In Eq.1, $l()$ is the loss function of the model w.r.t. a single channel, c_q is the quantized weight of the channel and c_0 is the original one.

We use the Taylor Expansion method to estimate the loss function $l(c)$, ignoring the high-order items:

$$l(c) \approx l(c_0) + g^T(c - c_0) + \frac{1}{2}(c - c_0)^T H(c - c_0) \tag{2}$$

where $g = \mathbb{E}[\frac{\partial}{\partial c} l(c)]$ is the loss's gradient w.r.t. the channel, and $H = \mathbb{E}[\frac{\partial^2}{\partial c^2} l(c)]$ is the second-order gradient (i.e., Hessian matrix). It is infeasible to calculate the Hessian matrix. We approximate the Hessian with the (empirical) Fisher information matrix (FIM) F on the calibration dataset $\mathcal{D}$:

$$H \approx F = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} g_d g_d^T \tag{3}$$

Note that F is w.r.t. a channel, differing from the diagonal FIM in the recent works that ignores any cross-neuron interactions (Kwon et al., 2022; Kim et al., 2024).

Based on this approximation, the second order loss factor $\frac{1}{2}(c - c_0)^T(g_d g_d^T)(c - c_0)$ can be further simplified to $\frac{1}{2}(g_d^T(c - c_0))^2$, simplifying the expensive chained matrix multiplication into a single vector product. Finally, the

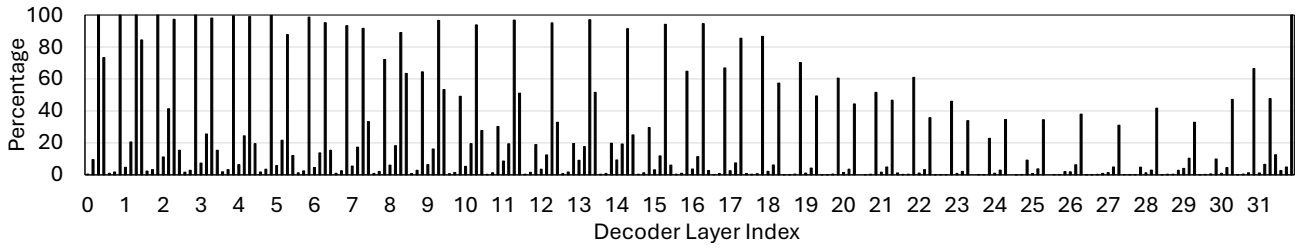

Figure 2: The percentage of high-salient out features within each linear layer of Llama 3.1 8B model according to each feature's contribution to the final loss after quantizing to 4-bit, with 10% high-salient features globally. Each decoder layer contains `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj` in order.

salience can be calculated by:

$$S_c = \frac{1}{|D|}\sum_{d\in D}|g_d^T(c_q - c_0) + \frac{1}{2}(g_d^T(c_q - c_0))^2| \quad (4)$$

We do not ignore the first order information during the calculation, differing from the recent quantization works (Frantar et al., 2022; Dettmers et al., 2024; Kim et al., 2024). Note that what we require is the loss itself rather than the arguments of the loss function, thus we do not need to ignore the first order factor to simplify the arguments calculation.

Algorithm 1 Global precision search procedure.

Input: Weight and gradient of all linear layers ($W_i \in \mathbb{R}^{O\times I}$, $G_i \in \mathbb{R}^{O\times I}$ for layer $i \in [1..L]$). Total number of output channels with large bit-width (T).

Output: Channel index for large and small bit-width ($\mathcal{C}^{lg}$ and $\mathcal{C}^{sm}$).

$\mathcal{C}^{global} \leftarrow ()$ ▷ *Global channel information*
for $i = 1$ **to** L **do**
  $W_i^{delta} \leftarrow$ quantize(W_i) - W_i
  $S^{1st} \leftarrow$ sum($G_i \odot W_i^{delta}$, dim=1)
  $S^{2nd} \leftarrow 0.5 * (S^{1st})^2$
  $S \leftarrow |S^{1st} + S^{2nd}|$ ▷ *O channels' salience* $\in \mathbb{R}^O$
  for $cid = 1$ **to** O **do**
    $\mathcal{C}^{global} \leftarrow \mathcal{C}^{global} \cup$ (tuple(i, cid, S_{cid}))
  end for
end for
Sort $\mathcal{C}^{global}$ by salience in descending order.
$\mathcal{C}^{lg}, \mathcal{C}^{sm} \leftarrow \mathcal{C}^{global}_{:T}, \mathcal{C}^{global}_{T:}$

Algo.1 illustrates the procedure of the global precision search. It calculates the salience of all the output channels of all linear layers and sort them in descending order. Given the global threshold T as the number of large-bit precision channels, the first T channels are intended to be quantized with 8-bit, and the other channels will be quantized with smaller bit-width (i.e., 4-bit in this paper).

3.3 Efficient Quantization Computation System

Two-step dequantization to make use of int8 Tensor Core. As for the W4A8 computation, the dequantized weight and activations are $(W_q - z)s_w$ and $A_q s_a$, where W_q and z are uint4 datatype (4-bit unsigned integer), A_q is int8 datatype, and s_w and s_a are float16 datatype. Directly dequantizing the tensors into float16 datatype before the MatMul computation will prevent us using the fast 8-bit Tensor Core on the GPU. Instead, MixLLM uses a two step dequantization within each group. Specifically, MixLLM first partially dequantizes the weight into $(W_q - z)$, and then multiply it by A_q with the 8-bit Tensor Core. Finally, it multiplies this MatMul result by the two scales within each group. Note that we use int8 datatype for $(W_q - z)$ so that there is no overflow problem.

Fast I2F with partially fusing into Tensor Core instruction. In the above two-step dequantization computation, the step 2 is the multiplication between integer and float tensor, requiring integer to float conversion (I2F). As I2F instruction is expensive on the modern GPUs, we make use of the range-dependent fast I2F transformation to convert the I2F instruction into two add/sub instructions. Specifically, it is based on the fact that there exists a certain range where an integer value's binary is the same as a corresponding float binary. As shown in Fig.4, the binary with the first 9 bits as `010010110` represents a series of consecutive int32 and float32 values, respectively. We can add a bias to an integer value to make it within this consecutive range, and then subtract a corresponding bias in float to restore its value in float. We take the middle value in this range as the bias to maximize the data range that can be safely converted, whose hexadecimal number is $mid =$ `0x4b400000` (i.e., in the remaining 23 bits, the first bit is 1 and the other bits are 0). This allows to convert a consecutive range of 2^{23} int32 numbers to float32. The range of dot product of k int8 elements is in $2^{16}k$, thus the above fast I2F conversion allows the k value to be 128. We use quantization group size as 128 and can use the fast I2F safely:

```
// bias_int = as_int(mid), bias_fp = as_float(mid);
```

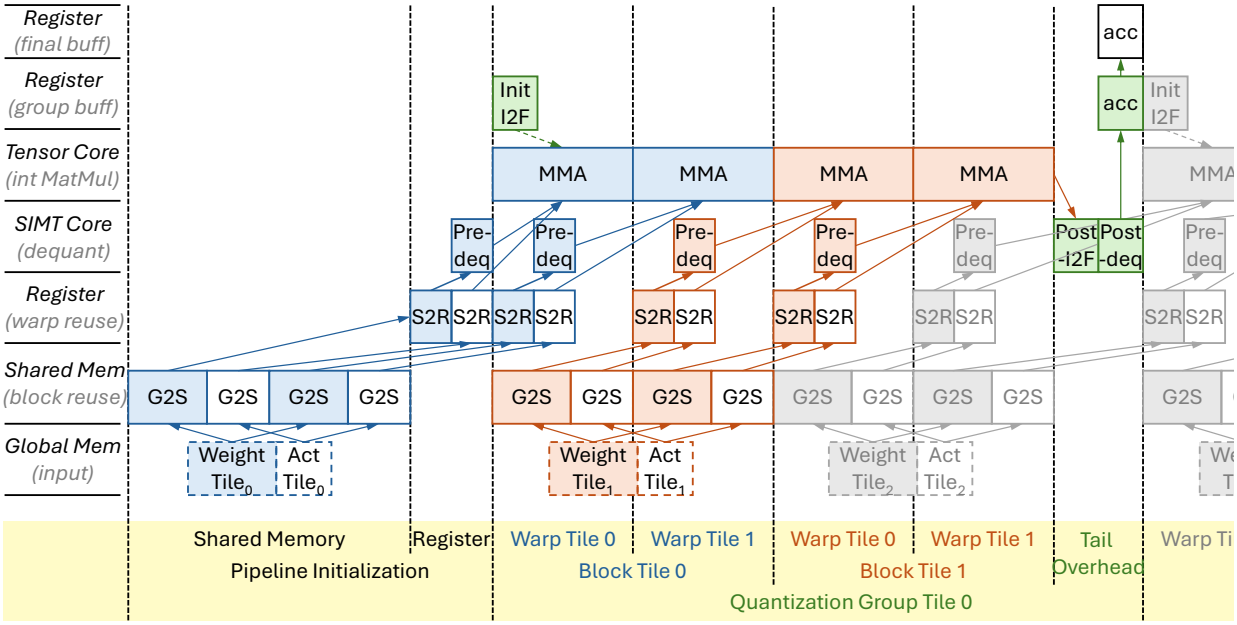

Figure 3: The GPU kernel software pipeline of group-wise W4A8/W8A8 quantized MatMul. It assumes perfect overlapping. G2S: load global to shared memory; S2R: load shared memory to register; MMA: matrix multiply-accumulation; I2F: integer to float conversion; deq: dequantize; acc: accumulate. While this pipeline is modeled on the NVIDIA A100 architecture, its fundamental principles remain applicable to subsequent generations, such as Hopper and Blackwell, subject to minor architectural refinements. For instance, newer architectures can utilize the Tensor Memory Accelerator (TMA) to load activation tensors directly, bypassing registers before they reach the Tensor Cores. Furthermore, these architectures support warp specialization for memory loading as an alternative to a uniform execution scheme.

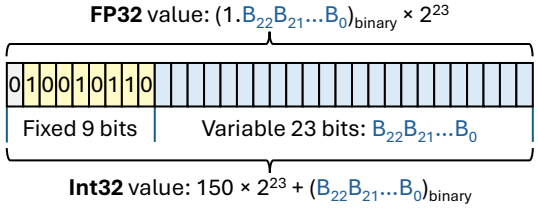

Figure 4: The float and integer value of binary `(010010110xx...x)`, each within a consecutive range.

```
int tmp = src_int + bias_int;
int dst_float = *((float*)&tmp) - bias_fp;
```

We further fuse the integer subtraction into the Tensor Core mma (Matrix Multiply-Accumulate $D = AB + D$) instruction. We initialize the accumulator D as the `bias_int` before MatMul computation of each quantization group, and will only need to subtract the `bias_fp` after the MatMul. In another word, the expensive I2F is converted into a single float subtraction. The above I2F simplification brings more than 20 TOPS performance improvement for the 512/4096/4096 (M/N/K) quantized MatMul computation on an A100 GPU.

End-to-end software pipeline of the quantized linear kernel on the GPU. Fig.3 shows the software pipeline of the quantized kernel. Besides the basic warp tile and block tile, we introduce the quantization group tile for the per-group quantized computation. It uses two output buffers for the output accumulation at register level, one for the per-group accumulation, and the other for the global accumulation. This allows to apply the per-group scales on the group-level buffer. We initialize the group buffer with the `bias_int` at the beginning of the group tile, and subtract `bias_fp` at the end of the group tile. As for the *two-step dequantization*, the first step is within the warp tile where each input element will subtract the zero-point before feeding into MMA, the second step is at the end of the group tile by multiplying the scale. We use the vectorized intrinsic to perform four int8 subtract in a single instruction (`vsub4`) (Lin et al., 2025). Besides, to improve the global memory loading efficiency, we prepack the memory layout of the weight tensor ahead-of-time to avoid the runtime permutation of the input elements. This software pipeline can overlap the memory loading, the dequantization computation with SIMT Core, and the MatMul computation with Tensor Core to the best, and minimizes the overhead of group-wise dequantization.

3.4 Parallel Execution of Sub-problems of Different Bit-width

As for the execution shown in Fig.1, MixLLM executes different sub-problems in parallel on the GPU with CUDA Graph. Finally, the MatMul execution of the two parts write to the same target tensor with different channel indices to generate the final output. We implement this function with the fused epilogue of MatMul to scatter the output to the corresponding indices, which is basically costless.

3.5 Quantization-aware Roofline Discussion

To systematically analyze the performance bounds of MixLLM quantization schemes, we extend the traditional Roofline model to account for the heterogeneous compute pipelines on NVIDIA GPUs. Using the NVIDIA A100 (80GB SXM) as our target architecture, we evaluate the system against its peak hardware limits: 2039 GB/s of High Bandwidth Memory (HBM2e) throughput, 624 TOPS of dense INT8 Tensor Core compute, and 19.5 TFLOPS of FP32 SIMT compute. Introducing sub-byte quantization and asymmetric zero-points fundamentally alters the arithmetic intensity (operations per byte) and introduces multiple competing compute ceilings across the execution units.

Memory Traffic. From a memory bandwidth perspective, arithmetic intensity is governed by the quantized data footprint and associated metadata. For a group size of $G = 128$, the W8A8 symmetric scheme fetches 8-bit weights, 8-bit activations, and a 16-bit scale, yielding an effective weight footprint of $1 + 2/G$ bytes per element. The W4A8 asymmetric scheme compresses weights to 4 bits, requiring a 16-bit scale and a 4-bit zero-point, yielding an effective footprint of $0.5+2.5/G$ bytes per weight. Consequently, W4A8 drastically shifts the memory-bound slope upward, allowing significantly higher attainable throughput in bandwidth-constrained regimes, such as small batch decoding. This motivates quantization with lower bits for weight in MixLLM.

Compute Ceilings: Tensor Core vs. SIMT Dequantization. In the compute-bound regime, performance is dictated by the interplay between the Tensor Cores (handling the GEMM) and the SIMT cores (handling dequantization).

For both W8A8 and W4A8, output scaling requires a transition from the Tensor Core's `int32` accumulators to the SIMT core's `float32` pipeline. For every group size of $G = 128$, the Tensor Cores perform 128 INT8 multiply-accumulate (MAC) operations (256 operations total) to produce a single `int32` partial sum. Before the FP32 scales (s_w and s_a) can be applied, the SIMT cores must first issue an `int32`-to-`float32` cast instruction. Therefore, the post-GEMM workload per group consists of this explicit cast instruction followed by the FP32 scale multiplications. Even with this added casting overhead, the application's required operational ratio—256 INT8 operations per 3 SIMT instructions (cast + multiplies)—remains well below the A100 hardware's provisioned ratio of 32:1 (624 INT8 TOPS vs. 19.5 FP32 TFLOPS), thanks to the fast I2F and partial fusion methodology in MixLLM. Because the SIMT requirement is so light, this post-GEMM casting and scaling can be hidden behind the Tensor Core accumulation. This further motivates the group-wise 8-bit quantization than per-channel/per-token solutions.

However, W4A8 introduces a heavier pre-GEMM dequantization step:

$$Y = ((W_q - z) \times A_q) \cdot s_w \cdot s_a$$

Before the Tensor Cores can execute the INT8 MACs, the 4-bit weights must be unpacked and the zero-point subtracted. By leveraging the `vsub4` intrinsic, the hardware can perform 4 `int8` subtractions within a single SIMT instruction, highly optimizing the $(W_q - z)$ computation. Despite this instruction-level parallelism, the SIMT integer pipeline must still continuously feed the Tensor Cores. Because this pre-GEMM workload scales at $O(N^2)$ and competes for instruction fetch and register bandwidth concurrently with the $O(N^3)$ Tensor Core math, it creates an auxiliary, lower compute ceiling. Thus, the effective peak compute for W4A8 is strictly lower than the absolute 624 TOPS achieved by W8A8.

Intersection Point (Ridge Point) Analysis. The traditional Roofline ridge point defines the intersection between memory bandwidth and peak compute, marking the transition from a memory-bound to a compute-bound workload. For pure INT8 Tensor Core execution on the A100, this intersection occurs at roughly 306 OPs/Byte (624 TOPS / 2039 GB/s). Because the W4A8 scheme is constrained by the SIMT `vsub4` dequantization ceiling rather than the peak Tensor Core ceiling, its horizontal Roofline ceiling drops. Consequently, the intersection between the W4A8 memory traffic line and its compute ceiling shifts to the left. This leftward shift indicates that W4A8 workloads saturate their maximum achievable compute at a lower arithmetic intensity than W8A8, entering the compute-bound regime earlier due to SIMT instruction overhead.

4 EVALUATION

4.1 Setup

As for MixLLM evaluation, we use 0%, 10%, 20%, 50% and 100% of 8-bit based on the 4-bit quantization, respectively. Meanwhile, we use 8-bit for activation quantization. Both the weight and activation are group-wise quantized with group size 128. The 4-bit part is asymmetric quantized and the 8-bit part (including that in weight) is symmetric. Similar to the recent work (Lin et al., 2025; Liu et al., 2024), we enable GPTQ and clipping in MixLLM.

Table 1: Perplexity evaluation (↓) on wikitext2/c4 (gray for c4), sequence length 2048. `NA` means no support. `Abn` means the value is too large ($> 10^5$). For MixLLM, `pn` means $n\%$ 8-bit.

baselines		Llama 3.1/3.2 1B	Llama 3.1/3.2 8B	Llama 3.1/3.2 70B	Qwen2.5 0.5B	Qwen2.5 1.5B	Qwen2.5 7B	Qwen2.5 32B	Mistral 7B v0.3
float16		9.75/12.72	6.24/8.95	2.81/6.68	13.07/17.55	9.26/13.11	6.85/10.44	5.02/8.95	5.32/7.84
Basic RTN	W4A16	11.72/15.56	6.82/9.72	3.55/7.43	15.54/20.55	10.35/14.35	7.23/10.88	5.27/9.14	5.51/8.04
	W5A16	10.15/13.25	6.40/9.15	3.16/9.52	13.61/18.17	9.52/13.38	6.95/10.53	5.09/8.99	5.38/7.91
SmoothQuant	W8A8	9.89/12.91	6.34/9.08	2.92/6.77	13.84/18.40	9.63/13.49	7.17/10.85	5.12/9.04	5.35/7.88
QuaRot	W4A4	Abn/Abn	8.34/11.95	6.16/9.91	NA/NA	Abn/Abn	8.15/12.05	6.26/9.98	5.83/8.50
	W4A8	Abn/Abn	6.60/9.67	3.43/7.10	NA/NA	Abn/Abn	7.03/10.68	5.23/9.10	5.40/7.99
QServe	W4A8	Abn/Abn	6.64/9.49	3.49/7.07	Abn/Abn	Abn/Abn	7.39/11.06	5.55/9.31	5.44/7.98
MixLLM	W4A8 (p0)	10.36/14.09	6.54/9.62	3.30/7.24	14.43/19.61	9.66/13.79	7.03/10.75	5.21/9.08	5.42/8.02
	W4.4A8 (p10)	10.05/13.51	6.42/9.33	3.02/6.83	13.42/18.13	9.44/13.43	6.92/10.57	5.12/9.01	5.36/7.93
	W4.8A8 (p20)	9.95/13.25	6.37/9.22	2.97/6.79	13.32/17.99	9.40/13.35	6.90/10.53	5.09/9.00	5.35/7.90
	W6A8 (p50)	9.85/12.98	6.30/9.09	2.86/6.73	13.21/17.78	9.33/13.25	6.88/10.49	5.05/8.98	5.33/7.87
	W8A8 (p100)	9.76/12.75	6.25/8.97	2.81/6.68	13.12/17.60	9.28/13.14	6.86/10.45	5.02/8.96	5.32/7.84

Baselines and configurations. In Sec.4.2 and Sec.4.4, we compare to the pure PTQ methods of weight-activation quantization, neither the QAT nor the methods training the quantization parameters (e.g., training the rotation matrix like in SpinQuant(Liu et al., 2024)). Specifically, we compare MixLLM with the SOTA PTQ weight-activation methods, including the most widely used SmoothQuant (Xiao et al., 2023) and the recent SOTA QuaRot(Ashkboos et al., 2024) (of both W4A4 and W4A8) and QServe (Lin et al., 2025). The 8-bit tensors are all symmetric quantized in all baselines. We use symmetric and per-channel/token quantization in QuaRot, following the setting in its paper. We follow the official configurations to use 0.85/0.15 as the alpha/beta parameter for SmoothQuant, and 0.3/0.7 for QServe. We disable the KV quantization of QuaRot and QServe in our experiments to make the comparison fair. In Appendix.4.6, We also compare the ppl with GPTQ (Frantar et al., 2022), AWQ(Lin et al., 2024), SqueezeLLM(Kim et al., 2024), OmniQuant(Shao et al., 2024), AffineQuant(Ma et al., 2024), Atom(Zhao et al., 2024) and SpinQuant(Liu et al., 2024).

Models and Datasets. We evaluate MixLLM and the baselines on Llama 3.1 8B and 70B (Meta, Cited 2024), Llama 3.2 1B, Qwen2.5 0.5B, 1.5B, 7B and 32B (Group, Cited 2024), and Mistral 7B v0.3 (Jiang et al., 2023). We use wikitext2 (Merity et al., 2017) as the calibration set for MixLLM, and the default pile dataset (MIT-Han-Lab, Cited 2024) for SmoothQuant and QServe. MixLLM uses 128 samples with sequence length of 2048. SmoothQuant and QServe uses 64 samples with sequence length of 1024 to prevent OOM.

Metrics. We compare the perplexity (ppl) between all the baselines on wikitext2 and C4 (Raffel et al., 2020) datasets. We also compare a set of popular downstream tasks on Llama 3.1 8B, Qwen2.5 7B, and Mixtral 7B v0.3 through lm-eval (Gao et al., 2024), including BBH (Suzgun et al., 2023), GPQA (Rein et al., 2023), MMLU-Pro (Wang et al., 2024), MuSR (Sprague et al., 2024), ARC challenge (Boratko et al., 2018), and HellaSwag (Zellers et al., 2019). We conduct the system experiments on NVIDIA A100 (80G) GPUs with CUDA 12.1, except for that Sec.4.3.2 discusses the performance on NVIDIA H100 GPU. We use PyTorch 2.4.1 and transformers 4.45.2.

4.2 Perplexity Evaluation

Tab.1 shows the perplexity on Wikitext2 and C4 dataset for the commonly used open source LLMs, of different baselines. It shows that: **1)** Using 4.4 bits of weights with MixLLM can achieve the similar accuracy with 5 bits RTN weight-only quantization, even with 8-bit activation quantization enabled in MixLLM. This is mainly because MixLLM assigns the high-salience output channels with larger bit-widths than the uniform 5-bit solution. **2)** As for the weight-activation quantization baselines, MixLLM W4.4A8 shows a comparable accuracy with SmoothQuant with much smaller bit-width (60% of that in SmoothQuant). MixLLM W4.4A8 shows better accuracy than QuaRot and QServe with only 10% larger bit-width. It shows MixLLM achieves a good balance of memory consumption and accuracy. **3)** Note that MixLLM W8A8 quantization shows nearly lossless performance compared to the float16 baseline. Besides, the MixLLM W4A8 also outperforms the SOTA QuaRot and QServe for many cases, due to using group-wise quantization for the activation in MixLLM rather than the per-token method in QuaRot and QServe. This is part of the motivation that MixLLM uses group-wise quantization for the activation.

4.3 System Performance

4.3.1 Performance on A100 GPU

We have evaluated MixLLM for the single linear layer of token numbers ranging from 1 to 1024 with in features 4096 and out features 4096/14336, and compared it with the SOTA W4A16 (TRT-LLM) and QServe (Lin et al., 2025), shown in Fig.5. It also shows MixLLM kernel per-

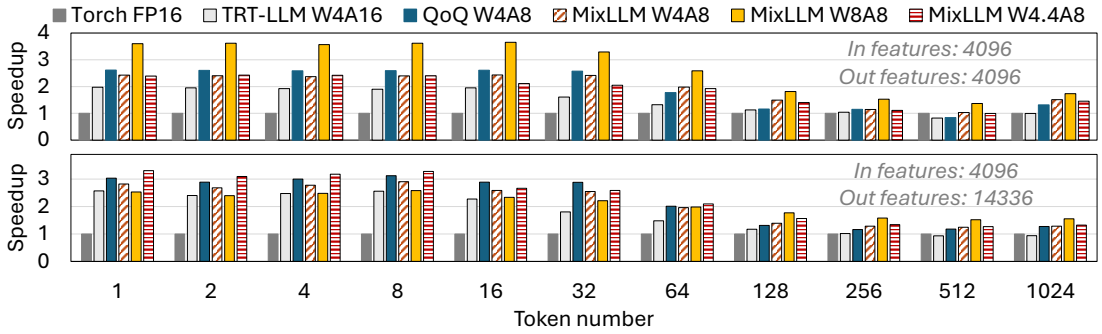

Figure 5: The speedup of two types of single linear layers over torch FP16 baseline on A100 GPU.

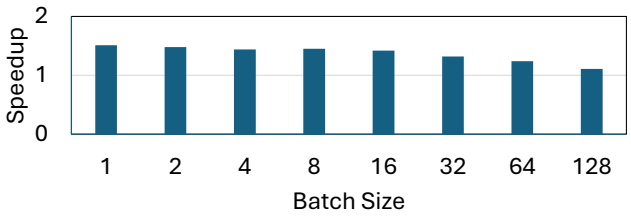

Figure 6: Speedup of MixLLM W8A8 of Qwen 2.5 7B over float16 baseline on A100 GPU.

formance of different percent of 8-bits (W4A8 0% 8-bit, W4.4A8 10% 8-bit, and W8A8 100% 8-bit). It shows that: **1)** MixLLM outperforms the float16 counterpart for all token numbers, achieving 1.96×, 2.76×, and 1.88× averaged speedup with MixLLM W4A8, W8A8, and W4.8A8 respectively for output feature 4096, and 2.45×, 2.15×, 2.34× for output feature 14336. **2)** MixLLM outperforms the SOTA W4A16 solution, achieving 1.31×, 1.80×, and 1.25× averaged speedup with MixLLM W4A8, W8A8, and W4.8A8 respectively for output feature 4096, and 1.36×, 1.31×, 1.33× for output feature 14336. **3)** MixLLM achieves better performance than QServe with similar bit-width, achieving 1.03×, 1.41×, and 0.99× averaged speedup with MixLLM W4A8, W8A8, and W4.8A8 respectively for output feature 4096, and 1.08×, 1.04×, 1.05× for output feature 14336.

We have also integrated MixLLM into vLLM, achieving 1.41× speedup of output token throughput (token/sec) than the float16 baseline given the configuration of batchsize 2 and input/output length 1000/1000, using W4.4A8 for Mixtral-7B on a single A100 GPU. Fig.6 shows the speedup over float16 baseline using MixLLM W8A8 quantization for Qwen2.5 7B model on A100 GPU, with input/output length 1000/1000 and a range of batch size from 1 to 128.

4.3.2 Performance on H100 GPU

We evaluate the performance of single linear layers on the NVIDIA H100 GPU, comparing the MixLLM W4A8 kernel against the W4A16 baseline (Fig. 7). To the best of

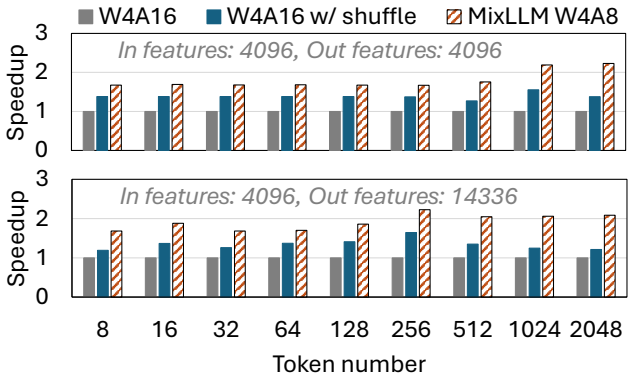

Figure 7: The speedup of two types of single linear layers over W4A16 baseline on H100 GPU.

our knowledge, an H100-optimized implementation of the QoQ solution (Lin et al., 2025) is currently unavailable. The W4A16 baseline was established by identifying the optimal configuration through exhaustive tuning of the official CUTLASS group-wise and asymmetric-quantized W4A16 examples. Our analysis considers both a naive 4-bit layout and an advanced shuffled 4-bit layout. Results indicate that MixLLM consistently outperforms the W4A16 solutions across all configurations. Specifically, for an output feature size of 4096, MixLLM achieves average speedups of 1.81× and 1.39× over the naive and shuffled layouts, respectively; for a size of 14336, these speedups are 1.91× and 1.34×.

4.4 Downstream Tasks Evaluation

We first evaluate GSM8K (Cobbe et al., 2021) on Qwen2.5 7B model, to validate the quantization efficacy on long-reasoning tasks. Using MixLLM W4.4A8 quantization, the `strict-match` metric only drops from 0.8 (the float16 model) to 0.792, only a 0.008 drop. Besides this, we evaluate a large number of the downstream tasks on three popular LLMs, shown in Tab.2. The result shows that: **1)** MixLLM W4.4A8 outperforms all the 4-bit weight quantizations, with only 10% more bit-width. For example, for the MMLU-

Table 2: Downstream tasks evaluation (↑) on Llama-3.1-8B/Qwen2.5-7B/Mistral-7B-v0.3. The above is the average of the three models. BBH is 3 shot, MMLU pro is 5 shot, and others are zero shot.

	BBH	GPQA	MMLU-Pro	MuSR	ARCc	HellaSwag
float16	48.62 46.52/54.09/45.25	30.86 31.08/33.11/28.39	35.52 32.91/43.86/29.80	41.07 37.99/44.51/40.72	52.24 53.41/51.02/52.30	79.43 78.92/78.94/80.43
SmoothQuant W8A8	47.82 46.37/52.57/44.52	30.90 31.40/33.94/27.36	35.04 32.61/42.98/29.52	42.06 39.05/46.39/40.73	51.74 53.33/50.00/51.88	79.20 78.88/78.48/80.24
QuaRot W4A4	41.10 36.96/45.42/40.92	27.53 25.41/28.94/28.23	27.60 22.99/34.40/25.42	39.46 37.92/40.68/39.77	45.99 43.00/46.33/48.63	74.85 72.87/73.54/78.14
QuaRot W4A8	46.95 44.95/52.98/42.92	30.28 30.96/30.71/29.18	33.60 29.95/42.45/28.41	41.65 39.05/45.58/40.32	51.39 50.00/52.30/51.88	78.55 77.83/77.84/79.98
QServe W4A8	45.78 40.98/51.23/45.14	30.02 28.99/32.50/28.56	32.84 28.16/41.72/28.63	39.92 37.60/41.59/40.57	50.54 51.28/49.15/51.19	78.10 76.90/77.52/79.89
MixLLM W4A8	46.92 43.44/44.75/52.59	29.90 29.58/28.26/31.87	33.75 30.18/29.59/41.49	41.70 38.81/43.11/43.19	51.82 51.71/51.88/51.88	78.61 77.94/79.71/78.17
MixLLM W4.4A8	48.17 46.27/52.58/45.66	30.09 29.17/31.75/29.36	34.53 31.08/43.26/29.26	41.74 39.32/44.79/41.11	52.70 53.67/51.96/52.47	79.00 78.20/78.58/80.21
MixLLM W8A8	48.84 46.84/54.35/45.34	30.93 30.51/33.21/29.07	35.54 33.00/43.80/29.83	40.94 37.32/44.91/40.59	52.10 53.24/50.94/52.13	79.42 78.98/78.88/80.40

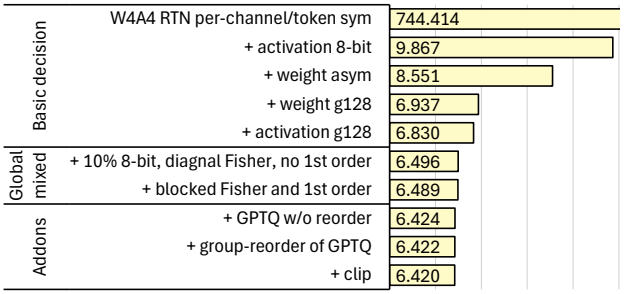

Figure 8: The perplexity (wikitext2) of Llama 3.1 8B model with different configurations.

Pro task, the average metric of MixLLM W4.4A8 is improved by 1.69, 6.93, and 0.93 over QServe, QuaRot W4A4, and QuaRot W4A8, respectively. **2)** MixLLM W8A8 is nearly lossless, showing higher accuracy than SmoothQuant. This comes from the group-wise quantized activation of MixLLM.

4.5 Ablation Study

Fig.8 shows the perplexity of Llama 3.1 8B model by adding different optimizations gradually. With the basic RTN quantization, using 8-bit for activation, and asymmetric and group-wised weight quantization contribute significantly to the accuracy improvement. This demonstrates the effectiveness of the decisions made in Sec.3.1.2. Based on these decisions, the 10% of 8-bit output features improves the accuracy to a high level, for which using blocked Fisher and including the first-order Taylor factor also contributes to the accuracy. Finally, applying GPTQ and clipping can further improve the accuracy.

4.6 Comparison with More Related Work

Comparison with Atom. Tab.4 shows the perplexity of Atom and MixLLM with the similar bit-width (i.e., W4.4A8). We use 512 outliers in Atom so its weight is around 4.4-bit on average (using Atom's opensourced code with git commit 7e3618b). It shows that MixLLM has much better accuracy than Atom. We also evaluated the kernel performance of Atom, showing that our W4.4A8 kernel achieves 1.56x speedup than Atom's W4A8 kernel for the linear layer with sequence length 1024. This means our proposed memory and computation pipeline has much better performance than the related mixed-precision work.

Comparison with SliM-LLM. Tab.5 shows the perplexity of SliM-LLM W4A16 (it only supports weight-only) and MixLLM with different bit-width. SliM-LLM W4A16 is evaluated using the opensourced code. It is the mix of 3/4/5-bits in its code, and the precision is searched within each single layer, rather than using the global precision search. Note that MixLLM can support the mix of any bit-width. We evaluated MixLLM with the mix of 4-bit/6-bit (W4.2A8, 4.2-bit on average with 90% 4-bit and 10% 6-bit), and the mix of 3-bit/4-bit/6-bit (W3.9A8, 3.9-bit on average with 30% 3-bit, 60% 4-bit, and 10% 6-bit) in this table. We notice even MixLLM W4A8 (uniform 4-bit for the weight) can defeat SliM-LLM W4A16. This is understandable because SliM-LLM itself focuses on 2-bit and 3-bit optimization, and its paper's main body does not show any 4-bit results. In contrast, MixLLM W3.9A8 can defeat MixLLM W4A8. This demonstrates that MixLLM's global precision search can better allocate bit-width to the important weight elements to achieve the nearly lossless results. Note that the percentage of bit-widths is determined intuitively here, so that it may not be the optimal percentage to achieve the optimal accuracy. How to determine the

Table 3: PPL (wikitext2) comparison with the reported numbers in the related works.

Model	FP16	GPTQ W4A16	AWQ W4A16	SqueezeLLM W4A16 0.45%	OmniQuant W4A16/W4A4	AfineQuant W4A16/W4A4	Atom W4A4 128 outliers	SpinQuant W4A8	MixLLM W4.4A8
LLaMA 2 7B	5.47	5.59	5.60	5.57	5.58/14.26	5.58/12.69	6.03	5.7	5.55
LLaMA 3 8B	6.14	6.46	6.55	-	-	-	7.57	6.5	6.32

Table 4: PPL of Atom and MixLLM with similar bit-width.

Models	Llama 2 7B	Llama 2 13B
Atom W4.4A8	5.64	5.03
MixLLM W4.4A8	5.54	4.93

Table 5: PPL (wikitext2) of SliM-LLM and MixLLM.

Models	Llama		Qwen2.5		
	3.2 1B	3.1 8B	0.5B	1.5B	7B
SliM-LLM W4A16	10.80	6.55	14.85	9.68	7.02
MixLLM W3.9A8	10.20	6.50	13.68	9.51	6.97
MixLLM W4A8 (uniform)	10.36	6.54	14.43	9.66	7.03
MixLLM W4.2A8	10.07	6.43	13.53	9.45	6.93
MixLLM W4.4A8	10.05	6.42	13.42	9.44	6.92

portion of different bit-width can be a new research problem.

We compare MixLLM with more recent quantization works according to the reported numbers in their papers (Tab.3), showing that MixLLM achieves superior accuracy to a broad range of related works with similar memory consumption.

4.7 Overhead of Global Precision Search

Tab.6 shows the global precision search overhead described in Sec.3.2. As noted in Sec.4.1, the calibration dataset has 128 samples with sequence length of 2048. We use a single A100 GPU for the 1.5B, 7B and 8B models, and 4 A100 GPUs for the 70B models to perform the search. We make use of `device_map` in huggingface for multi-GPU execution, which is sequential execution of layers on different devices. The 7B/8B models require about 7 minutes and the 70B models require less than 60 minutes to complete the search. Considering that the quantization only needs to be performed once, the searching algorithm is practical for the real workloads. In contrast, SliM-LLM takes more than 3 hours to determine the precision of Llama 3.1 8B model and more than 4 hours for Qwen2.5 7B model according to our experiments.

Table 6: The overhead of global precision search.

Models	Llama 3.1		Mistral	Qwen2.5	
	8B	70B	7B v0.3	1.5B	7B
Time (min)	7	55	7	2	7

Table 7: The average percentage of 8-bit out features in the seven classes of linear layers in Llama 3.1 8B, with 10% global 8-bit out features in MixLLM.

layer (xx_proj)	q	k	v	o	gate	up	down
avg 8-bit (%)	3.93	12.36	71.22	18.70	0.73	1.46	53.82

Table 8: PPL of enabling KV Quantization in MixLLM.

Models	Llama 3.1 8B	Qwen2.5 7B
MixLLM W4.4A8 w/o KV quant	6.42	6.92
MixLLM W4.4A8 KV8	6.42	6.96

4.8 High Precision Distribution

Fig.2 shows the percentage of 8-bit out features in each of the linear layers of Llama 3.1 8B, with 10% global 8-bit out features searched by MixLLM (i.e., W4.4A8). It shows that high-salient (i.e., 8-bit) features are distributed very differently in different linear layers. Specifically, the `v_proj` and `down_proj` layers show much higher percentage of high-salient features than other layers, for which Tab.7 shows the average percentage of different classes of linear layers.

4.9 One-pass vs. Progressive Search

As described in Sec.3.2, MixLLM searches the high-salience features within a single pass rather than iteratively identifying the high-salience parts in a smaller step, as we observe the single-pass method show similar results with the iterative method and saves a lot of computation overhead than the latter. We have tried the progressive procedure on Llama 3.1 8B and Mistral 7B models, which identifies smaller portions of the high-salience features iteratively. Results show that the accuracy is the same to the one-pass method to two decimal places. However, the progressive method shows much higher search time due to the repeated procedure. The one-pass method takes 7 minutes for each of the two models to search 10% high-salience features, while the progressive method that searches 2% high-salience iteratively takes 30 minutes to find top 10% features.

4.10 Working with KV Quantization

It is straightforward to apply any KV quantization technology together with MixLLM's weight-activation quantization, as KV quantization is orthogonal to the weight-

Table 9: PPL (wikitext2) when using different number of samples for salience search (W4.4A8 quantized).

#samples	128	64	32	16
Llama 3.1 8B	6.42	6.42	6.42	6.42
Qwen2.5 7B	6.92	6.92	6.92	6.93

activation quantization. Tab.8 shows the perplexity (wikitext2) of Llama 3.1 8B and Qwen 2.5 7B for enabling and disabling KV quantization. It shows that the 8-bit KV quantization is nearly lossless upon MixLLM.

4.11 Calibration Dataset for Salience Search

We use 128 samples for the calibration to search the high-salience channels in MixLLM, which is a common configuration in the existing solutions. We also find that a smaller dataset can still identify the high-salience channels in MixLLM. The perplexity of using different samples for MixLLM salience search are shown in Tab.9. This shows that shrinking the sample size from 128 to 16 for the global salience search has negligible effect to the accuracy.

We also use different dataset families between salience searching and perplexity evaluation to evaluate the effect of input distribution shift. When using c4 dataset for salience search and evaluate the perplexity on wikitext2 (W4.4A8 quantized), the Llama 3.1 8B and Qwen2.5 7B has the same perplexity with that using wikitext2 for the calibration. This shows that changing the calibration dataset does not affect the accuracy for the same task. In another word, when the data distribution between the calibration dataset and real input is different, MixLLM's salience search algorithm still works well.

5 Summary

We have presented MixLLM, achieving high accuracy with low memory consumption and high system efficiency with the rarely explored optimization space of mixed-precision quantization between output features. MixLLM identifies the salience of each output feature according to the loss w.r.t. the global model rather than each single layer. By assigning larger bit-width to the features need it the most, MixLLM achieves the superior accuracy to SOTA with low memory consumption. The sub-problems of different bit-widths are disjoint and can run in parallel efficiently on the GPU. We have identified the sweet spot of the quantization configuration that is friendly to both accuracy and system efficiency. To address the challenge of system efficiency, we design the two-step dequantization to enable using int8 Tensor Core and the fast integer-float conversion to reduce the dequantization overhead. We have designed the end-to-end software pipeline to overlap the memory access, the dequantization computation with SIMT Core and the MatMul with Tensor Core to the best. Experiment results show that MixLLM achieves superior accuracy to existing works and state-of-the-art system efficiency with low memory cost.

References

Abdelkhalik, H., Arafa, Y., Santhi, N., and Badawy, A. A. Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis. In *IEEE High Performance Extreme Computing Conference, HPEC 2022, Waltham, MA, USA, September 19-23, 2022*, pp. 1–8. IEEE, 2022.

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in LLM inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pp. 117–134. USENIX Association, 2024.

Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Jaggi, M., Alistarh, D., Hoefler, T., and Hensman, J. Quarot: Outlier-free 4-bit inference in rotated llms. *CoRR*, abs/2404.00456, 2024. doi: 10.48550/ARXIV.2404.00456. URL https://doi.org/10.48550/arXiv.2404.00456.

Boratko, M., Padigela, H., Mikkilineni, D., Yuvraj, P., Das, R., McCallum, A., Chang, M., Fokoue-Nkoutche, A., Kapanipathi, P., Mattei, N., Musa, R., Talamadupula, K., and Witbrock, M. A systematic classification of knowledge, reasoning, and context within the ARC dataset. In Choi, E., Seo, M., Chen, D., Jia, R., and Berant, J. (eds.), *Proceedings of the Workshop on Machine Reading for Question Answering@ACL 2018, Melbourne, Australia, July 19, 2018*, pp. 60–70. Association for Computational Linguistics, 2018.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S. M., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y. Sparks of artificial general intelligence: Early experiments with GPT-4. *CoRR*, abs/2303.12712, 2023.

Chee, J., Cai, Y., Kuleshov, V., and Sa, C. D. Quip: 2-bit quantization of large language models with guarantees. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano,

R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *CoRR*, abs/2208.07339, 2022.

Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. Spqr: A sparse-quantized representation for near-lossless LLM weight compression. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ: hessian aware quantization of neural networks with mixed-precision. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 293–302. IEEE, 2019.

Egiazarian, V., Panferov, A., Kuznedelev, D., Frantar, E., Babenko, A., and Alistarh, D. Extreme compression of large language models via additive quantization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

Frantar, E. and Alistarh, D. Optimal brain compression: A framework for accurate post-training quantization and pruning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers. *CoRR*, abs/2403.17887, 2024.

Group, A. Qwen2.5: A party of foundation models. https://qwenlm.github.io/blog/qwen2.5/, Cited 2024.

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

Holmes, C., Tanaka, M., Wyatt, M., Awan, A. A., Rasley, J., Rajbhandari, S., Aminabadi, R. Y., Qin, H., Bakhtiari, A., Kurilenko, L., and He, Y. Deepspeed-fastgen: High-throughput text generation for llms via MII and deepspeed-inference. *CoRR*, abs/2401.08671, 2024.

Huang, W., Qin, H., Liu, Y., Li, Y., Liu, X., Benini, L., Magno, M., and Qi, X. Slim-llm: Salience-driven mixed-precision quantization for large language models. *CoRR*, abs/2405.14917, 2024.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de Las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., and Keutzer, K. Squeezellm: Dense-and-sparse quantization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=0jpbpFia8m.

Kumar, T., Ankner, Z., Spector, B. F., Bordelon, B., Muennighoff, N., Paul, M., Pehlevan, C., Ré, C., and Raghunathan, A. Scaling laws for precision. *arXiv preprint arXiv:2411.04330*, 2024.

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. A fast post-training pruning framework for transformers. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Lee, C., Jin, J., Kim, T., Kim, H., and Park, E. OWQ: outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In Wooldridge, M. J., Dy, J. G., and Natarajan, S. (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium*

on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada, pp. 13355–13364. AAAI Press, 2024.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W., Wang, W., Xiao, G., Dang, X., Gan, C., and Han, S. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In Gibbons, P. B., Pekhimenko, G., and Sa, C. D. (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024.

Lin, Y., Tang, H., Yang, S., Zhang, Z., Xiao, G., Gan, C., and Han, S. Qserve: W4A8KV4 quantization and system co-design for efficient LLM serving. *MLSys*, 2025.

Liu, Z., Zhao, C., Fedorov, I., Soran, B., Choudhary, D., Krishnamoorthi, R., Chandra, V., Tian, Y., and Blankevoort, T. Spinquant: LLM quantization with learned rotations. *CoRR*, abs/2405.16406, 2024.

Ma, Y., Li, H., Zheng, X., Ling, F., Xiao, X., Wang, R., Wen, S., Chao, F., and Ji, R. Affinequant: Affine transformation quantization for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Malinovskii, V., Mazur, D., Ilin, I., Kuznedelev, D., Burlachenko, K., Yi, K., Alistarh, D., and Richtárik, P. Pv-tuning: Beyond straight-through estimation for extreme LLM compression. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Men, X., Xu, M., Zhang, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect. *CoRR*, abs/2403.03853, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Meta. Llama 3. https://ai.meta.com/blog/meta-llama-3, Cited 2024.

MIT-Han-Lab. Pileval. https://huggingface.co/datasets/mit-han-lab/pile-val-backup, Cited 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof q&a benchmark. *CoRR*, abs/2311.12022, 2023.

Shao, W., Chen, M., Zhang, Z., Xu, P., Zhao, L., Li, Z., Zhang, K., Gao, P., Qiao, Y., and Luo, P. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Sprague, Z., Ye, X., Bostrom, K., Chaudhuri, S., and Durrett, G. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., and Wei, J. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 13003–13051. Association for Computational Linguistics, 2023.

TensorRT-LLM. Tensorrt-llm. https://github.com/NVIDIA/TensorRT-LLM, Cited 2024.

Tseng, A., Chee, J., Sun, Q., Kuleshov, V., and Sa, C. D. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024a.

Tseng, A., Sun, Q., Hou, D., and Sa, C. D. QTIP: quantization with trellises and incoherence processing. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *CoRR*, abs/2406.01574, 2024.

Wu, X., Xia, H., Youn, S., Zheng, Z., Chen, S., Bakhtiari, A., Wyatt, M., Aminabadi, R. Y., He, Y., Ruwase, O., Song, L., and Yao, Z. Zeroquant(4+2): Redefining llms quantization with a new fp6-centric strategy for diverse generative tasks. *CoRR*, abs/2312.08583, 2023.

Xia, H., Zheng, Z., Li, Y., Zhuang, D., Zhou, Z., Qiu, X., Li, Y., Lin, W., and Song, S. L. Flash-llm: Enabling low-cost and highly-efficient large generative model inference with unstructured sparsity. *Proc. VLDB Endow.*, 17(2): 211–224, 2023.

Xia, H., Zheng, Z., Wu, X., Chen, S., Yao, Z., Youn, S., Bakhtiari, A., Wyatt, M., Zhuang, D., Zhou, Z., Ruwase, O., He, Y., and Song, S. L. Quant-llm: Accelerating the serving of large language models via fp6-centric algorithm-system co-design on modern gpus. In Bagchi, S. and Zhang, Y. (eds.), *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*, pp. 699–713. USENIX Association, 2024.

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 Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *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 Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Yu, G., Jeong, J. S., Kim, G., Kim, S., and Chun, B. Orca: A distributed serving system for transformer-based generative models. In Aguilera, M. K. and Weatherspoon, H. (eds.), *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pp. 521–538. USENIX Association, 2022.

Yuan, Z., Niu, L., Liu, J., Liu, W., Wang, X., Shang, Y., Sun, G., Wu, Q., Wu, J., and Wu, B. RPTQ: reorder-based post-training quantization for large language models. *CoRR*, abs/2304.01089, 2023.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In Korhonen, A., Traum, D. R., and Màrquez, L. (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4791–4800. Association for Computational Linguistics, 2019.

Zhao, Y., Lin, C., Zhu, K., Ye, Z., Chen, L., Zheng, S., Ceze, L., Krishnamurthy, A., Chen, T., and Kasikci, B. Atom: Low-bit quantization for efficient and accurate LLM serving. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*, 2024.

Zheng, Z., Ji, X., Fang, T., Zhou, F., Liu, C., and Peng, G. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching. *arXiv preprint arXiv:2412.03594*, 2024.