Enabling Fast 2-bit LLM on GPUs: Memory Alignment, Sparse Outlier, and Asynchronous Dequantization

Jinhao Li*
Shanghai Jiao Tong University
China

Shiyao Li*
Tsinghua University & Infinigence-AI
China

Jiaming Xu*
Shanghai Jiao Tong University & Infinigence-AI
China

Shan Huang
Shanghai Jiao Tong University
China

Yaoxiu Lian
Shanghai Jiao Tong University
China

Jun Liu
Shanghai Jiao Tong University
China

Yu Wang
Tsinghua University
China

Guohao Dai†
Shanghai Jiao Tong University & Infinigence-AI
China
daiguohao@sjtu.edu.cn

ABSTRACT

Large language models (LLMs) have demonstrated impressive abilities in various domains while the inference cost is expensive. Many previous studies exploit quantization methods to reduce LLM inference cost by reducing storage and accelerating computation. The state-of-the-art methods use 2-bit quantization for mainstream LLMs (*e.g.* Llama2-7b, etc.). However, challenges still exist in reducing LLM inference cost with 2-bit quantization: **(1) Nonnegligible accuracy loss for 2-bit quantization.** Weights are quantized by groups, while the ranges of weights are large in some groups, resulting in large quantization errors and nonnegligible accuracy loss (*e.g.* >3% for Llama2-7b with 2-bit quantization in GPTQ and Greenbit). **(2) Limited accuracy improvement by adding 4-bit weights.** Increasing 10% extra average bit more 4-bit weights only leads to <0.5% accuracy improvement on a quantized Llama2-7b model. **(3) Time-consuming dequantization operations on GPUs.** Mainstream methods require a dequantization operation to perform computation on the quantized weights, and the 2-order dequantization operation is applied because scales of groups are also quantized. These dequantization operations lead to >50% execution time, hindering the potential of reducing LLM inference cost.

To tackle these challenges and enable fast and low-cost LLM inference on GPUs, we propose the following techniques in this paper. **(1) Range-aware quantization with memory alignment.** We point out that the range of weights by groups varies. Thus, we only quantize a small fraction of groups with the larger range using 4-bit with memory alignment consideration on GPUs. **(2) Accuracy-aware sparse outlier.** We point out that the distribution of the sparse outliers with larger weights is different in 2-bit and 4-bit groups, and only a small fraction of outliers require 16-bit quantization. Such design leads to >0.5% accuracy improvement with <3% average increased bit for Llama2-7b. **(3) Asynchronous dequantization.** We point out that calculating the scales of each group is independent of the loading weights of each group. Thus, we design the asynchronous dequantization on GPUs, leading to up to 3.92× speedup. We conduct extensive experiments on different model families and model sizes. We achieve 2.85-bit for each weight considering all scales/zeros for different models. The end-to-end speedup for Llama2-7b is 1.74× over the original model, and we reduce both runtime cost and hardware cost by up to 2.70× and 2.81× with less GPU requirements.

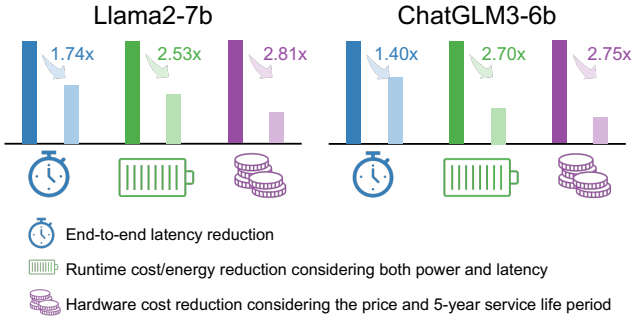

Figure 1: Benefit of end-to-end speedup, runtime inference cost/energy reduction, and hardware cost reduction on mainstream LLMs.

1 INTRODUCTION

Large language models (LLMs) demonstrate remarkable capabilities in various domains, excelling in tasks like natural language understanding and generation [9, 20]. However, their computational

*These authors contributed equally to this work.
†Corresponding Author.

DAC '24, June 23–27, 2024, San Francisco, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

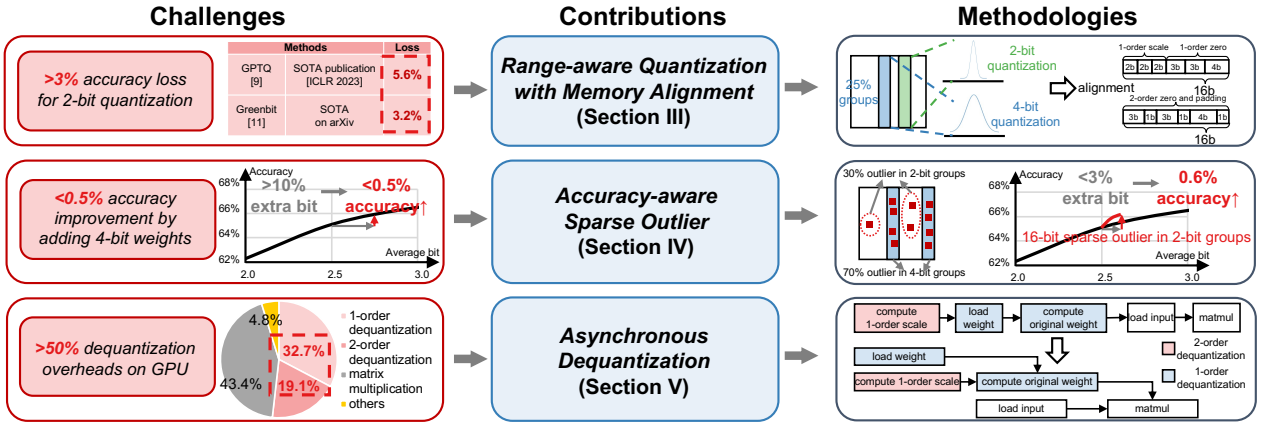

Figure 2: Challenges in fast and low-cost quantized LLM inference. We propose three novel contributions: range-aware quantization with memory alignment, accuracy-aware sparse outlier, and asynchronous dequantization, to solve these challenges.

expense during inference is noteworthy. For instance, OpenAI reveals that the GPT-4 Turbo incurs a cost of millions of dollars each day [22]. This substantial cost underscores the financial considerations associated with deploying and utilizing such advanced language models [23]. Many previous studies exploit quantization methods to reduce LLM inference cost by reducing storage and accelerating computation [8, 11, 14, 15, 26]. The state-of-the-art methods have realized 2-bit quantization for mainstream LLMs, exemplified by the Llama-2 family [25]. These methods quantize weights by group to improve accuracy and the scales are also quantized (2-order) to further reduce average bit [12].

However, there are still existing challenges of reducing LLM inference cost with the 2-bit quantization method. **(1) Nonnegligible accuracy loss for 2-bit quantization.** Both the post-training quantization (PTQ, *e.g.,* GPTQ [11]) and the quantization-aware training (QAT, *e.g.,* LLM-QAT [16] and Greenbit [12]) approaches incur the 3.2% to 5.6% accuracy loss for Llama2-7b with 2-bit quantization [11, 12]. **(2) Limited accuracy improvement by adding 4-bit weights.** LLMs quantized with 4-bit exhibit an impressive ability to retain accuracy, showcasing less than 1% accuracy loss. Consequently, it is a common practice to improve accuracy by adding 4-bit quantization. However, Figure 2 left middle illustrates that increasing 10% extra average bit with more 4-bit weights only leads to <0.5% accuracy improvement on a quantized Llama2-7b model. **(3) Time-consuming dequantization operations on GPUs.** Mainstream methods require a dequantization operation to perform computation on 2-bit weights. Furthermore, a 2-order dequantization operation is applied because the scales of groups are also quantized. Figure 2 left bottom shows that, the dequantization operation takes over 50% of the total execution time. These challenges become the crucial factors impeding fast and low-cost LLM inference.

In response to these challenges, we enable fast and low-cost LLM inference on GPUs with the following contributions in this paper:

- **Range-aware quantization with memory alignment.** We point out that the range of weights by groups varies. Thus, only 25% of the weights are quantized using 4-bit with memory alignment. Such a method reduces the accuracy loss for 2-bit Llama2-7b quantization from 8.7% to 2.9%.
- **Accuracy-aware sparse outlier.** We point out that only a small fraction of outliers exist in weights quantized using 2-bit. We quantize these sparse outliers with <3% increased average weight bit and improve the accuracy by >0.5%.
- **Asynchronous dequantization.** We point out that calculating the scales of each group is independent of the loading group weights. Thus, we design the asynchronous dequantization and accelerate the GPU kernel by up to 3.92×.

We conduct extensive experiments on different model families (*e.g.,* Llama2 [25] and ChatGLM3 [28]) and model sizes (*e.g.,* Llama2-7b to Llama2-70b). We achieve 2.85-bit for each weight considering all scales/zeros for different models. The end-to-end speedup for Llama2-7b is 1.74×, and we reduce both runtime cost and hardware cost by up to 2.70× and 2.81× with less GPU requirements.

2 PRELIMINARIES AND BACKGROUNDS

2.1 Uniform Quantization

A group of weights with range of (w_{min}, w_{max}) are quantized to range $(0, 2^N - 1)$. Scale (s) and Zero-point (z) map the floating point weights to N bits integers. Scale is the half-precision step size of the quantizer and Zero-point is an N bits integer [19]. The quantization and dequantization operations are:

$$s = \frac{w_{max} - w_{min}}{2^N - 1} \tag{1}$$

$$z = round(\frac{-w_{min}}{s}) \tag{2}$$

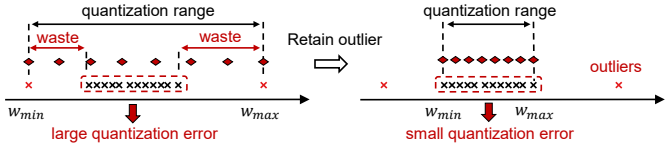

Figure 3: Example of quantization errors.

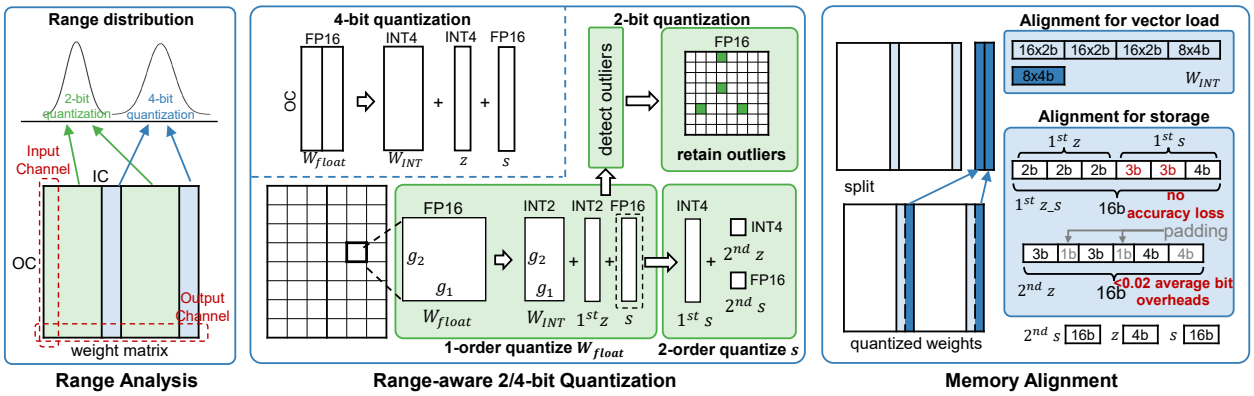

Figure 4: Overview of range-aware quantization method. We first analyze the range distribution of different input channels, and then quantize the weights with range-aware 2/4-bit quantization. Last, we propose three techniques for memory alignment.

$$w_{int} = clamp(0, 2^N - 1, round(\frac{w}{s}) + z) \tag{3}$$

$$w_{float} = (w_{int} - z) \times s \tag{4}$$

For 2-bit quantization, weights with input dimensions are quantized by group, while the scales of groups within output dimensions are also quantized to further reduce the average bit. Here, the average bit, denoted as $\overline{bit}$ with 1-order and 2-order quantization is as follows:

$$\overline{bit} = \begin{cases} N + \dfrac{N+16}{g_1}, & \text{1-order} \quad (5) \\ N + \dfrac{N+N_2}{g_1} + \dfrac{N_2+16}{g_1 \times g_2}, & \text{2-order} \quad (6) \end{cases}$$

where g_i represents the size of group and N_2 represents the bit width with 2-order.

2.2 Mixed-precision Quantization

In LLMs, certain weights exhibit disproportionately magnitudes compared to the majority. As shown in Figure 3, by applying normal uniform quantization, the quantization range is expanded by these outliers and thus, the majority of weights are quantized with few steps, which leads to large quantization error. Mixed-precision quantization methods like SpQR and SqueezeLLM filter and retain outliers that negatively impact quantization error and store outliers with 16-bit sparse matrix [8, 14]. These methods quantize the weights matrix into a dense matrix and a 16-bit sparse matrix. The Compressed Sparse Row (CSR) is used to represent sparse matrices, optimizing memory usage. In CSR, three arrays encapsulate the matrix data. The values array (values) efficiently arranges non-zero elements in row-major order, while the column indices (col_ind) array meticulously records the corresponding column index for each non-zero element. The row pointer (row_ptr) array serves as a guide, storing the starting index for each row in the values array. Outlier retention of weights is common for low-bit quantization in LLMs. Each outlier is stored using one 32-bit combination: a 16-bit weight value and a corresponding 16-bit column index. Additionally, for every row, a 32-bit number is allocated to store the total count of outliers up to that row. This results in an overall 32-bit storage overheads for each weight outlier.

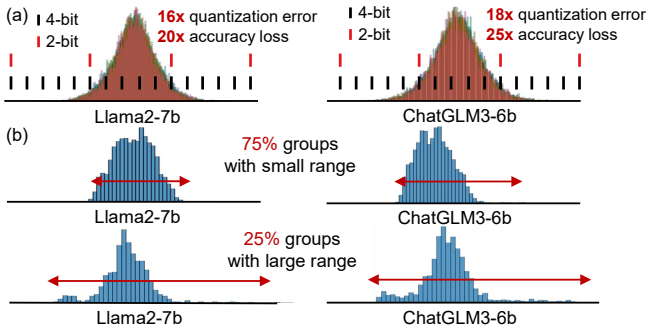

Figure 5: (a) The challenge of quantization with 2-bit is large quantization errors and accuracy loss. (b) The range of 25% groups is large in Llama2-7b and ChatGLM3-6b.

3 RANGE-AWARE QUANTIZATION WITH MEMORY ALIGNMENT

Challenge. The quantization method is widely used to reduce LLM inference cost [8, 11, 15] by reducing storage and accelerating computation. Quantization uses discrete values to represent continuous weights, leading to quantization errors. We depict the distribution of weights in typical LLMs and the comparison of using 2-bit and 4-bit quantization in Figure 5(a). We find that the 2-bit approach leads to 16× ∼ 18× larger quantization errors and further results in 20× ∼ 25× larger accuracy loss. Thus, the key challenge is that even the state-of-the-art 2-bit methods still suffer from >3% accuracy loss for Llama2-7b [11, 12].

Motivations and Insights. Some previous methods like SpQR [8], SparseGPT [10] and AWQ [15] have analyzed that the quantization errors are large in specific groups, so we further depict the range of weights per group, as shown in Figure 5(b). The range of weights exhibits different distributions for different groups, and only a small fraction of groups show a large range while the others show a small range. Applying 4-bit quantization to these groups with a large range can reduce the quantization error. Thus, our key insight is, **only a small fraction of weights requires 4-bit**

quantization and average bit can be still reduced by applying 2-bit quantization on the other groups.

$$\overline{bit} = \alpha \times (2 + \frac{2+N_2}{g_1} + \frac{N_2+16}{g_1 g_2}) + (1-\alpha) \times 4 \tag{7}$$

Approach: Range-aware Quantization. In Figure 4, we depict the quantization flow of weights. Notably, weights are quantized by group to reduce accuracy loss while the groups with a large range still require larger bit-width to improve accuracy. Therefore, we first analyze the range distribution by group. The range of groups in the same input channels is merged to streamline the complexity of hardware computation. Concretely, weights are divided by the Hessian inverse matrix to get the calibrated weights, leveraging the Hessian inverse matrix's ability to discern weight importance. Subsequently, the calibrated weights are squared to magnify the range variance. The amplitude of each input channel is computed as follows:

$$Amp_i = \sum_{j=0}^{OC} \frac{W_{ij}^2}{H_i^2} \tag{8}$$

Thus, the range of weights by groups is equivalent to the amplitude variance in the grouped input channels. Then, we analyze the variance and obtain the information that the input channels with large variance require 4-bit quantization and others require 2-bit quantization. According to the analysis results, we apply the QAT finetuning approach to reduce the quantization error. In the forward process of QAT, the weights with a large range are quantized and then dequantized with 4-bit by group while others are with 2-bit. After finetuning, we use the same analysis results to quantize different grouped input channels with different bit-width. For small range input channels, we first do 2-bit 1-order quantization for the weights W_{float} by group g_1 in the direction of input channels to get the quantized weight W_{INT}, 1-order zeros $1^{st}z$ and scales s. To further reduce the average bit, the scales s are also quantized with 4-bit by group g_2 in the direction of output channels (2-order quantization) to get 1-order quantized scales $1^{st}s$, 2-order zeros $2^{nd}z$ and scales $2^{nd}s$. During 2-bit quantization, outliers are also detected and retained to reduce accuracy loss. Then, the weights in large range input channels are quantized with 4-bit.

Approach: Memory Alignment. As illustrated in Figure 4, the quantized weights are split into two quantized weight matrixes. The first matrix stores 3 groups of 2-bit quantized weights (the size of group is 16) and 8 4-bit quantized weights into continuous memory. The other matrix stores 8 remaining 4-bit quantized weights into independent memory. Thus, the memory access for each thread in GPU is memory-aligned. Then, 1-order zeros $1^{st}z$ and scales $1^{st}s$ are packed together and two of three scales are compressed into two 3-bit to fill 16-bit memory without paddings while only the 2-order zeros are stored with paddings. These memory alignment methods preserve accuracy loss and only leads to <0.02 average bit overheads.

4 ACCURACY-AWARE SPARSE OUTLIER

Challenge. The mixed-precision quantization method can efficiently reduce quantization errors of channels with large variations. However, there are still some outliers sparsely distributed in both 2-bit and 4-bit channels. Outliers in 2-bit channels lead to significant

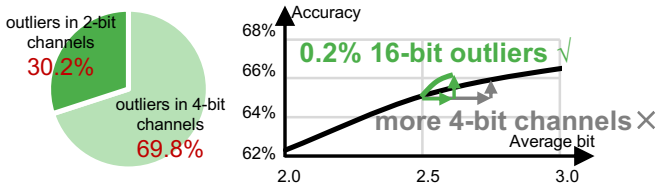

Figure 6: The distribution of outliers in Llama2-7b after quantization with 25% 4-bit channels.

quantization errors. In order to reduce the quantization error of these outliers, one straightforward approach is to continue adding 4-bit channels with the increased average bit. Nevertheless, the accuracy improvement of adding 4-bit channels is limited because these outliers are sparsely distributed. Figure 2 left middle shows that, increasing >10% extra bit only leads to <0.5% accuracy improvement when adding more 4-bit channels. Thus, the key challenge lies in reducing the quantization error of these sparse outliers with less increased average bit.

Motivation and Insights. Besides straightforwardly adding 4-bit channels, another way to reduce quantization errors is to use the 16-bit weights for these sparse outliers. Previous designs like SpQR [8] and SqueezeLLM [14] introduce ~1% 16-bit weights to represent these outliers with the corresponding sparse format(*e.g.,* CSR). Each sparse outlier requires at least one 16-bit for the weight representation and one 16-bit for the position. Thus, introducing 1% 16-bit sparse outliers leads to $(16 + 16 - 2) \times 1\% = 0.3$ extra average bit, which still increases >10% extra bit. We depict the sparse outlier distribution in 2-bit and 4-bit channels of Llama2-7b in Figure 6 left. Although only 25% channels are quantized using 4-bit in our practice, ~70% outliers are distributed in these 4-bit channels. Previous designs using 4-bit quantization [11, 15] lead to negligible accuracy loss, proving that 4-bit quantization is a promising way for accurate LLM inference. Thus, our key insight is, **only a small fraction (*e.g.,* ~30%) of sparse outliers are in the 2-bit channels and require 16-bit quantization to maintain the algorithm accuracy**.

Approach: Accuracy-aware Sparse Outlier. Based on the mixed 2-bit/4-bit quantization proposed in Sec. 3, we quantize a small fraction (*e.g.,* 25%) of weights using 4-bit by channels. We profile the accuracy improvement by continuing to add 4-bit channels. Figure 6 right shows that, the accuracy improvement is limited by adding >25% 4-bit weights. We only consider the average bit of weights, while the scales and zeros are not considered. The actual average bit is also related to the group size. The smaller group size leads to more scales and zeros for each group, resulting in a larger average bit [12]. In practice, we set the group size to 16, which is larger than Greenbit [12] and contributes to a smaller average bit. Instead of adding 4-bit channels when the average bit reaches 2.5, we represent sparse outliers only in 2-bit channels using 16-bit quantization. In practice, we finetune the model and restrict the ratio of sparse outliers in 2-bit channels to 0.2%. Thus, we increase $(16 + 16 - 2) \times 0.2\% = 0.06$ average bit, which is less than 3%. For Llama2-7b, the accuracy is increased by 0.6%.

Algorithm 1 GEMV with 2/4-bit dequantization algorithm

Input: W_{INT}: quantized weight, V: input vector
Output: O: output vector
1: load $1^{st}s$ to SharedMem
2: load $2^{nd}z, 2^{nd}s$ to SharedMem
3: syncthreads
4: load W_{INT} and $1^{st}z$ to SharedMem
5: $1^{st}s \leftarrow dequantize(1^{st}s, 2^{nd}z, 2^{nd}s)$
6: syncthreads
7: load v
8: $w_{float} \leftarrow dequantize(W_{INT}, 1^{st}z, 1^{st}s)$
9: syncthreads
10: $psum \leftarrow psum + w_{float} \times v$
11: $parallel\_reduce(\boldsymbol{O}, psum)$

Approach: Sparse Outlier Format and GPU Kernel. We utilize the CSR format used in previous designs [8, 14] for sparse outliers representation, and apply GPU kernels in the NVIDIA cuSPARSE library [2] to perform matrix multiplication on these outliers. In practice, only 0.2% weights are quantized using 16-bit, and the execution time on these sparse outliers is much faster (*e.g.,* >10× according to our profiling) than the computation on the dense 2-bit/4-bit weights.

5 ASYNCHRONOUS DEQUANTIZATION

Challenge. Because the weights are quantized by 2-bit and 4-bit, it requires the dequantization operation to restore the weights to 16-bit before performing the multiplication between the input and weights. We use 1-order and 2-order quantization to scale the weights and scales of each group. Previous designs (*e.g.*, SPQR[8], LLM-QAT[16]) use the synchronous dataflow (*i.e.*, performing dequantization after loading all weights), resulting in >50% overheads of end-to-end execution time. Thus, the key challenge is the time-consuming synchronous dequantization operation becomes the bottleneck in accelerating LLM inference after quantization.

Motivation and Insights. The 1-order and 2-order quantization requires two synchronous dequantizations in dataflow. However, the 2-order dequantization for calculating the scales of each group is independent of the weights of each group for the 1-order dequantization. Moreover, the 1-order dequantization for restoring the weights to 16-bit is independent of the input data of multiplication. Thus, our key insight is **calculating the scales of each group can be overlapped with loading weights of each group in GPU kernel.**

Approach: Asynchronous dequantization. Based on the insights above, we design the asynchronous dequantization dataflow as illustrated in Figure 7 on GPUs. With the help of the shared memory, we can overlap the calculating scales of each group and the loading weights of each group. Further, we use CUDA primitive *__shfl_down_sync()* to reduce the result inside a warp efficiently. From the perspective of memory access, we apply vector load (*e.g.* `double4`) technique to load quantized weights, activation, scales and zeros to minimize the numbers of memory access. Algorithm 1 shows the four main parts of our kernel. The preprocess (line 1 ~ 3) loads the zeros ($2^{nd}z$) and scales ($1^{st}s$ and $2^{nd}s$) for the 2-order dequantization, corresponding to ① in Figure 7. Then the 2-order dequantization and loading weight W_{INT} and $1^{st}z$ (line 4) provide the scales, zeros and weights for the 1-order dequantization, corresponding to operation ②. The third part is 1-order dequantization and loading input vector $V[\cdot]$ (line 7 ~ 8), corresponding to operation ③. After that, the multiplication (line 10), operation ④ in Figure 7 is performed. The final part (line 11) is used to reduce data from all threads.

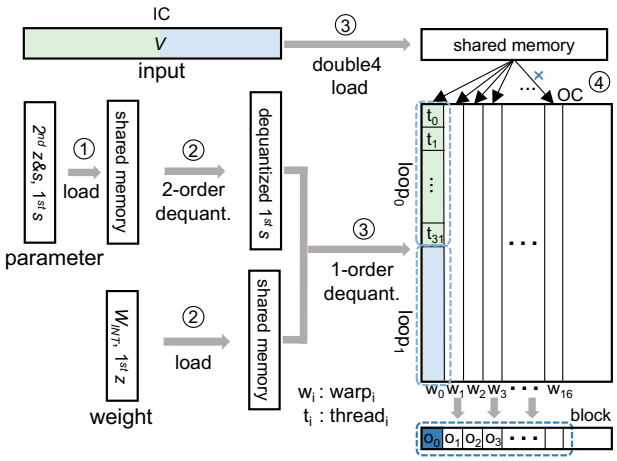

Figure 7: Asynchronous dequantization on GPUs.

Table 1: Algorithm accuracy with zero-shot performance

Method	$\overline{bit}$	PIQA/Hell./Wino./ARC-e	Average
Llama2-7b	16	77.2%/56.1%/68.7%/70.2%	68.0%
GPTQ	3	71.7%/48.2%/61.2%/56.1%	59.3%
Greenbit	2.91	77.2%/53.8%/65.8%/62.5%	64.8%
Ours (group=16)	2.85	75.9%/51.3%/66.4%/66.9%	65.1%
Ours (0.2%)	2.91	76.1%/52.0%/66.8%/67.2%	65.7%
Llama2-13b	16	78.8%/59.7%/69.6%/73.3%	70.4%
GPTQ	3	75.2%/56.1%/64.3%/64.2%	64.0%
Ours (group=16)	2.85	76.9%/55.6%/66.2%/69.2%	67.0%
Ours (0.2%)	2.91	77.3%/55.8%/66.7%/69.5%	67.6%
Llama2-70b	16	81.1%/64.0%/77.0%/77.7%	75.0%
Ours (group=8)	2.91	80.5%/62.1%/74.9%/76.5%	73.5%
ChatGLM3-6b	16	70.8%/49.4%/61.3%/51.1%	58.1%
Ours (group=8)	2.91	66.7%/44.7%/59.4%/48.5%	54.8%

6 EXPERIMENTAL RESULTS

6.1 Experimental Setup

Benchmarks. We conduct comprehensive experiments on the Llama2-7b model family [25] and the ChatGLM3-6b model [28]. The accuracy of the introduced quantized LLMs is assessed across four zero-shot benchmarks, namely PIQA [4], HellaSwag [27], WinoGrande [24], ARC-e [6]. We use 2-order weight-only quantization to evaluate the performance.

Table 2: Comparison of end-to-end speedup, runtime cost, and hardware cost.

Model	Method	GPU	Power(W)	Token/s	End-to-end speedup	Runtime cost reduction	Hardware cost reduction
Llama2-7b	PyTorch	3090×1	290/350	25.9	1×	1×	1×
	GPTQ	3090×1	320/350	37.5	1.45×	1.31×	1.39×
	Our	3090×1	350/350	45.2	**1.74**×	1.44×	1.60×
	Our	2080×1	150/215	34.0	1.31x	**2.53**×	**2.81**×
Llama2-13b	PyTorch	3090×2	240/350	22.0	1×	1×	1×
	GPTQ	3090×1	270/350	23.1	1.06×	1.88×	2.02×
	Our	3090×1	250/350	25.0	**1.14**×	**2.19**×	**2.24**×
Llama2-70b	PyTorch	A100×2	400/400	38.5	1×	1×	1×
	GPTQ	A100×1	400/400	46.5	1.21×	2.42×	**2.42**×
	Our	A100×1	400/400	50.5	**1.31**×	**2.62**×	**2.42**×
ChatGLM3-6b	PyTorch	3090×1	350/350	32.5	1×	1×	1×
	Our	3090×1	350/350	40.0	1.23×	1.23×	1.23×
	Our	2080×1	170/215	45.5	**1.40**×	**2.70**×	**2.75**×

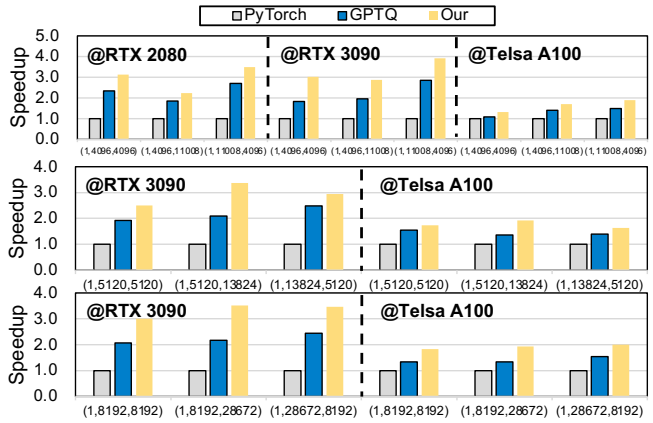

Figure 8: Kernel speedup. From the top to the bottom are Llama2-7b, Llama2-13b, and ChatGLM3-6b. The tuple in the x-axis represents the matrix shape of GEMV kernel.

Baselines. We compare our design with state-of-the-art quantization designs, including GPTQ [11] and Greenbit [12]. The original PyTorch implementation on HuggingFace [1] is used as the baseline.

Hardware Platforms. We implement our design and compare other baselines on NVIDIA RTX 2080, NVIDIA RTX 3090, and NVIDIA A100 (80G) GPUs with CUDA version 12.2. The runtime cost is evaluated considering the power consumption using the NVML library [3], and the hardware cost is evaluated considering the price ($12,500 for A100, $1499 for RTX 3090, and $699 for RTX 2080) with the 5-year service life period.

6.2 Accuracy Evaluations

Table 1 shows algorithm accuracy with zero-shot performance on Llama2 family and ChatGLM3-6b. We set the group size to 16 for Llama2-7b and Llama2-13b with 0.2% sparse outliers, and the group size to 8 for Llama2-70b and ChatGLM3-6b.

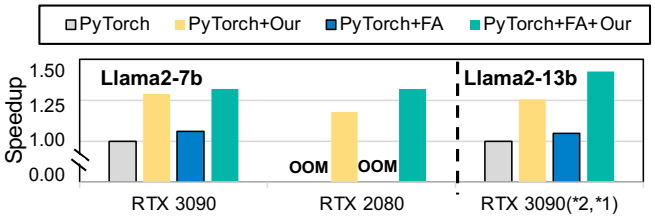

Figure 9: Comparison with FlashAttention (FA) kernels.

Compared with the state-of-the-art GPTQ [11] method, our design significantly reduces the accuracy loss from 8.7% to 2.3% for Llama2-7b, and from 6.4% to 2.8% Llama2-13b. Compared with Greenbit [12] on Llama2-7b, we achieve 0.9% higher accuracy with the same 2.91 average bit. On Llama2-70b and ChatGLM3-6b, the accuracy loss is also controllable from 1.5% to 3.3%.

6.3 Performance Evaluations

We compare the inference cost (end-to-end speedup, runtime cost, and hardware cost) among our method, PyTorch, and GPTQ in various model families and model sizes in Table 2. All results are normalized to PyTorch. In the Llama2 model family, our design achieves 1.14× ~ 1.74× end-to-end speedup and far outperforms other previous works (*e.g.*, GPTQ[11]) in terms of running cost reduction, reaching 2.19× ~ 2.53× over the original model. Moreover, the hardware requirement is dramatically lowered in our design for all models in Table 2, bringing up to 2.70× runtime cost reduction and 2.81× hardware cost reduction. The performance underscores the effectiveness of our design's optimizations and the practical utility of our approach tailored to optimize the inference and deployment of LLMs.

6.4 Detailed Discussions

6.4.1 Kernel Speedup. The detailed speedup of the GEMV kernel with dequantization is shown in Figure 8. We compare our kernel performance with some mainstream quantized kernels(*e.g.*,

GPTQ [11]) and the original FP16 models in common LLM GEMV cases. Our kernel achieves 1.33× ∼ 3.92× over FP16 on various NVIDIA GPUs.

6.4.2 Integrate with FlashAttention. FlashAttention [7] is widely used to accelerate LLM inference. We show that our method is also compatible with FlashAttention. Figure 9 shows that our method achieves higher speedup compared with adopting FlashAttention. Integrating our method with FlashAttention further accelerates LLM inference by up to 1.31×, and solves the limited memory problem on GPUs like NVIDIA RTX 2080.

7 CONCLUSIONS

We enable fast and low-cost LLM inference on GPUs in this paper with three novel techniques. We quantize 25% weights 4-bit quantization and 0.2% sparse outliers with 16-bit based on insights into weight distribution in LLMs. We further design the asynchronous GPU kernel to accelerate LLM dequantization. For Llama2-7b, our method achieves 2.85 average bits, and the end-to-end speedup is 1.74×. We reduce both runtime cost and hardware cost by up to 2.70× and 2.81× with less GPU requirements.

REFERENCES

[1] [n. d.]. https://huggingface.co/.
[2] [n. d.]. https://docs.nvidia.com/cuda/cusparse/.
[3] [n. d.]. NVIDIA Management Library (NVML) | NVIDIA Developer. https://developer.nvidia.com/nvidia-management-library-nvml.
[4] Yonatan Bisk, Rowan Zellers, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*.
[5] Christopher Clark, Kenton Lee, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044* (2019).
[6] Peter Clark, Isaac Cowhey, et al. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457* (2018).
[7] Tri Dao, Dan Fu, et al. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *NeurIPS* (2022).
[8] Tim Dettmers et al. 2023. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. *arXiv preprint arXiv:2306.03078* (2023).
[9] Mengnan Du et al. 2022. Shortcut learning of large language models in natural language understanding: A survey. *arXiv preprint arXiv:2208.11857* (2022).
[10] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. (2023).
[11] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).
[12] Nianhui Guo et al. 2023. Advanced Ultra-Low Bitrate Compression Techniques for the LLaMA Family of LLMs. *https://github.com/GreenBitAI/low_bit_llama* (2023).
[13] Ke Hong, Guohao Dai, et al. 2023. FlashDecoding++: Faster Large Language Model Inference on GPUs.
[14] Sehoon Kim, Coleman Hooper, et al. 2023. SqueezeLLM: Dense-and-Sparse Quantization. *arXiv preprint arXiv:2306.07629* (2023).
[15] Ji Lin, Jiaming Tang, et al. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *arXiv preprint arXiv:2306.00978* (2023).
[16] Zechun Liu, Barlas Oguz, et al. 2023. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models. *arXiv preprint arXiv:2305.17888* (2023).
[17] Stephen Merity, Caiming Xiong, et al. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).
[18] Todor Mihaylov, Peter Clark, et al. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789* (2018).
[19] Markus Nagel, Marios Fournarakis, et al. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295* (2021).
[20] Ansong Ni, Srini Iyer, et al. 2023. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*.
[21] Adam Paszke, Sam Gross, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[22] Dylan Patel and Afzal Ahmad. 2023. The Inference Cost Of Search Disruption - Large Language Model Cost Analysis. https://www.semianalysis.com/p/the-inference-cost-of-search-disruption.
[23] Baolin Peng, Chunyuan Li, et al. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277* (2023).
[24] Keisuke Sakaguchi, Ronan Le Bras, et al. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM* (2021).
[25] Hugo Touvron, Louis Martin, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[26] Guangxuan Xiao, Ji Lin, et al. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *ICML*.
[27] Rowan Zellers, Ari Holtzman, et al. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).
[28] Aohan Zeng, Xiao Liu, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414* (2022).