LLM-MQ: Mixed-precision Quantization for Efficient LLM Deployment

Shiyao Li^{1,2}, Xuefei Ning^{1}, Ke Hong^{1,2}, Tengxuan Liu^{1,2}, Luning Wang^{1,2}, Xiuhong Li^{2,3}, Kai Zhong^{1}, Guohao Dai^{2,4}, Huazhong Yang^{1}, Yu Wang^{1}

^{1}Dept. of EE, BNRist, Tsinghua University,
^{2}Infinigence-AI,
^{3}Peking University,
^{4}Shanghai Jiao Tong University

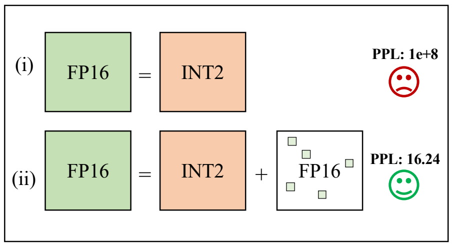
Abstract

Large Language Models (LLMs) have demonstrated impressive performance across various tasks. Nevertheless, deploying LLMs on edge devices presents significant challenges, primarily due to their substantial model size (e.g., over 10 billion parameters). Low-precision quantization is a promising way to reduce the memory requirement of LLMs. However, directly applying ultra-low-bit quantization to LLMs leads to significant performance degradation and fails to meet a specific weight memory budget. In this paper, we propose LLM-MQ, a Mixed-precision Quantization method, to address the above issues. Our method mainly contains three folds: (1) We propose a **sparse outlier protection** strategy for low-precision layers by protecting the outliers in FP16 format to maintain the performance. (2) We propose **sensitivity-based precision allocation** to assign the proper bit-width for each layer within the given budget for weight memory based on their first-order information and quantization error. (3) We develop **efficient CUDA core kernels** to accelerate mix-precision LLMs by fusing the dequantization and General Matrix-Vector Multiplication (GEMV). With comparable performance on various tasks, LLM-MQ can flexibly quantize LLMs that meet the given budget for weight memory. On NVIDIA T4 GPU, we achieve up to 1.6$\times$ end-to-end speedup compared to the pytorch FP16 baseline.

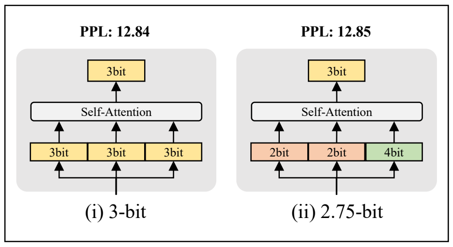
1 Introduction

Large language models (LLMs) have exhibited impressive performance across various language tasks. Recently, LLMs have led to the emergence of many interesting and helpful applications, such as ChatGPT [11], Copilot [2]. However, deploying LLMs on edge devices has been challenging due to the enormous model size. For example, the LLaMA-2-70b model [15] needs at least 140 GB of memory for deployment. Even the NVIDIA A100 GPU with 80 GB memory is not enough to deploy this model. It is especially not enough for devices with limited memory capacities.

To meet the demands of edge inference scenarios where model size constraints are paramount, the memory overhead of weights is the main bottleneck. Consequently, low-precision weight-only quantization can effectively reduce the memory requirement of LLMs. Furthermore, numerous methods [8; 6; 12; 14; 7] have been developed, all of which employ a consistent low-bit-width representation across all layers. They successfully quantize the FP16 LLMs into 4-bit or 3-bit LLMs with acceptable performance. However, for edge devices with more limited memory budgets, it is challenging to push the bit-width for each layer to 2-bit without significant accuracy loss.



(a) Sparse Outlier Protection



(b) Sensitivity-based Precision Allocation

Figure 1: (a) Sparse Outlier Protection. (i) Baseline methods directly quantize the original FP16 weights into 2-bit, with dramatically accuracy loss (ii) We keep 0.5% outlier at FP16 and quantize the rest normal weights int 2-bit. (b) Sensitivity-based Precision Allocation. (i) Baseline methods use the same bit-width for each layer. (ii) We assign lower bit-width to the less sensitive layers.

In this paper, we delve deeper into mixed-precision quantization to address the above issue. Given that different layers exhibit varying sensitivities to quantization, assigning the appropriate bit-width to each layer can provide advantages in meeting the specified memory budget while maintaining minimal accuracy loss. The main contributions of this paper are:

- We propose a sparse outlier protection strategy to only quantize the normal weight into ultra-low precision and keep the outliers in FP16 format. Experiments show that we can significantly reduce the accuracy loss using 2-bit quantization.
- We propose a sensitivity-based precision allocation method by modeling the precision allocation as an integer programming problem. Specifically, we model the given budget for weight memory as a constraint of the proposed integer programming problem. Experiments show that we can push the average bit-width into 2.8-bit.
- We design efficient CUDA kernels to accelerate the decoding stage by fusing the dequantization and GEMV operations and also design a memory arrangement strategy for odd-bit weights to improve the speed of memory access further. On NVIDIA T4 GPU, we achieve up to 1.6$\times$ end-to-end speedup compared to the pytorch FP16 baseline.

2 Method

2.1 Sparse Outlier Protection

As shown in Fig. 1 (a), while nearly 99.9% of the weights are concentrated around zero, the largest value is $\sim 10^7$ larger than the mean value. 2-bit uniform quantization results in a quantization error 2.3 $\times$ larger than 3-bit uniform quantization, leading to significant accuracy loss. To counteract this, we introduce sparse outlier protection, retaining 0.5% of outliers in FP16 while applying 2-bit quantization to the remaining values. This method decreases the error by 1.56 times, approximating 3-bit quantization outcomes. For efficient computation, FP16 outliers are stored using the CSR format, leveraging the sparse library ([1]), akin to the SqueezeLLM approach ([7])."

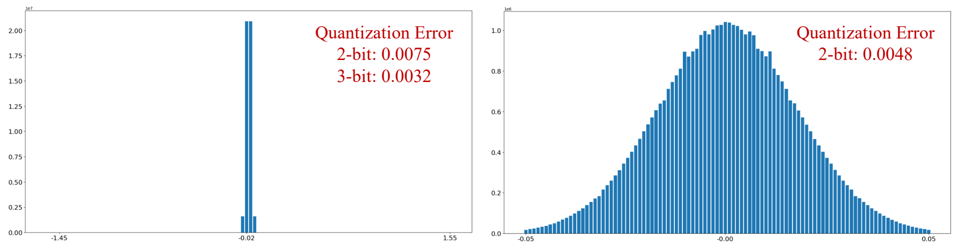
2.2 Sensitivity-based Precision Allocation

The primary goal of quantization is to express model weights using low-bit-width representation, ensuring the changes in the model output remain minimal ([5]). Since different layers have different sensitivities, we propose to assign a high bit-width to high-sensitivity layers and a low bit-width to low-sensitivity layers in order to minimize the change in model output. To pinpoint the more sensitive layers, we employ the first-order Taylor approximation in Eq. 1 to determine how the model output changes in response to weight perturbations.

$$\mathcal{L}(Q_b(\mathbf{W}_i)) \approx \mathcal{L}(\mathbf{W}) + \mathbf{g}_i^T(\mathbf{W}_i - Q_b(\mathbf{W}_i)), \tag{1}$$

where $\mathcal{L}$ is the loss function, $\mathbf{g}_i$ is the gradient of the loss function with respect to the weight of i-th layer $\mathbf{W}_i$, and $Q_b()$ is the b-bit quantization function. Note that in LLMs, the hessian matrix of $\sim$15% layers are not positive semi-definite, which means these layers have not converged to a local minimum. In this case, the second-order information of LLMs struggles to show the change of loss function accurately. We focus on the first-order information:

$$\min|\mathcal{L}(Q_b(\mathbf{W}_i))| \leq |\mathcal{L}(\mathbf{W}_i)| + \min|\mathbf{g}_i^T(\mathbf{W}_i - Q_b(\mathbf{W}_i))| = |\mathcal{L}(\mathbf{W}_i)| + \min s_{i,b}, \tag{2}$$



(a) Original Distribution (b) Distribution w/o 0.5% Outliers

Figure 2: (a) Directly apply 2-bit quantization to the original weight. (b) Only apply 2-bit quantization to the normal value while keeping 0.5% outliers in FP16 format.

where $s_{i,b}$ stand for $|\mathbf{g}_i^T(\mathbf{W}_i - Q_b(\mathbf{W}_i))|$. We minimize the sum of $s_{i,b}$ for all layers to minimize the change of loss function. In addition, we propose to use three different bit-width, that is 2, 3 and 4. We model the above bit-width allocation task as the following integer programming problem:

$$\arg\min_{c_{i,b}} \sum_{i}^{N}\sum_{b} c_{i,b} \cdot s_{i,b}, \tag{3}$$

$$s.t. \sum_{b} c_{i,b} = 1, \quad \sum_{i}^{N}\sum_{b} c_{i,b} \cdot \mathcal{M}(Q_b(\mathbf{W}_i)) \leq \mathcal{B}, \tag{4}$$

$$c_{i,b} \in \{0,1\}, b \in \{2,3,4\}, \tag{5}$$

where Q_b is the b-bit quantization function, $\mathcal{B}$ is the budget for weight memory of a certain device, $\mathcal{M}$ is the function to calculate memory usage of weights, N is the layer number of the target LLM, and c_i is the one-hot indicator of i-th layer to determine its proper bit-width. We use the budget for weight memory as a constraint of the optimization objective. With the help of the efficient integer programming solver, we can find a proper bit-width allocation scheme within a few seconds.

2.3 Efficient GPU Kernel Design

To maximize the utilization of the GPU's memory resources, we store two 4-bit weights in one byte. Eight 3-bit weights become three bytes, and four 2-bit weights are represented by one byte. To accelerate the inference of quantized models, we integrate weight dequantization with GEMV in our implementation, reducing memory access overhead. Additionally, we utilize the CUDA core instead of the Tensor core as employed in AWQ [8], primarily for two reasons: (1) The Tensor Core requires that each dimension of the inputs be at a minimum of eight, which is not required in the CUDA Core. In the GEMV operator, the minimal dimension of the inputs is one. Using zero-padding to expand the minimal dimension to 8 introduces inefficiencies. (2) The latency of GEMV latency is primarily dictated by model weight memory access, and the enhanced computational capacity of the Tensor Core does not alleviate this primary bottleneck.

3 Experiments

We focus on weight-only quantization with a group size of 128 (i.e., every 128 data points share a scaling factor and a zero point.). We also use the reparameterization strategy introduced in AWQ [8]. We conduct experiments of the proposed LLM-MQ on various benchmarks using LLaMA2 [15] and OPT [17] families. We assess the performance of quantized models across several benchmarks, The evaluation code is based on lm-harness-evaluation [1]. These include five zero-shot benchmarks: PIQA [3], HellaSwag [16], WinoGrande [13], ARC-e [4], and OpenBookQA [10]. Additionally, we evaluate the model on a language modeling task using the Wiki benchmark [9]. Tab. 1 presents the primary results for LLaMA2-13B, while the performance of other LLMs are detailed in the Appendix. Furthermore, in Tab. 1, we have not factored in the FP16 sparse weights when calculating the average bit-width, as they have a minimal impact, increasing it by a maximum of 0.08 bits, which is negligible.

The effect of the sparse outlier protection. As shown in Tab. 1, in the 3-bit and 4-bit quantization, RTN (Round-To-Nearest), AWQ and SqueezeLLM have no significant performance degradation.

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

Table 1: The performance of the LLaMA2-13B model on five zero-shot benchmarks by reporting their average accuracy and one language modeling task using perplexity.

Method	#Bit	PIQA	Hella.	Wino.	Arc-e	OpenBookQA	Avg. (↑)	Wiki (↓)
LLaMA2-13B	16.0	79.11	76.59	69.77	57.91	42.00	65.08	7.89
RTN	4.0	79.16	75.90	69.61	56.69	42.00	64.67	8.12
	3.0	77.15	73.68	67.64	56.90	40.80	63.23	9.26
	2.0	56.47	37.94	51.30	30.77	32.20	41.74	1056.33
AWQ [8]	4.0	79.00	76.14	70.32	57.49	42.20	65.03	8.08
	3.0	77.91	74.62	69.77	56.02	41.80	64.02	8.81
	2.0	50.76	25.69	50.28	26.98	31.60	37.06	5e6
SqueezeLLM [7]	4.0	78.94	76.05	69.14	57.70	42.80	64.93	8.44
	3.0	78.62	74.53	67.40	56.73	40.20	63.50	9.29
LLM-MQ (Ours)	4.0	79.49	76.31	69.30	58.50	41.20	64.96	8.03
	3.8	79.22	76.22	69.85	58.29	41.40	65.00	8.08
	3.6	79.05	75.88	69.77	58.59	42.20	65.10	8.23
	3.4	79.49	74.77	69.61	58.12	40.60	64.52	8.61
	3.2	79.33	75.12	67.96	57.87	41.60	64.38	8.43
	3.0	79.00	75.08	68.59	57.79	41.00	64.29	8.54
	2.8	78.73	74.32	67.96	57.95	41.20	64.03	8.83
	2.6	78.35	73.81	68.03	57.32	39.40	63.38	9.35
	2.4	77.31	72.93	68.59	54.63	40.00	62.69	10.03
	2.2	76.77	70.83	67.09	55.26	38.40	61.67	10.80
	2.0	75.84	68.32	65.51	54.29	37.20	60.23	12.17

However, in 2-bit quantization, the performance of RTN degrades significantly on all tasks. The performance of AWQ is even worse than RTN, especially on the Wiki benchmark. In LLM-MQ, we only protect the top 0.5% of the largest value in each layer in FP16 format and quantize the rest of the normal value to 2-bit. The average accuracy of LLM-MQ on zero-shot tasks is **23.17% and 19.16% better** than AWQ and RTN. On the Wiki benchmark, the perplexity of LLM-MQ is more than **two orders of magnitude lower than** that of other methods.

The effect of the sensitivity-based precision allocation. As depicted in Tab. 1, for 3-bit and 4-bit quantization, RTN (Round-To-Nearest), AWQ and SqueezeLLM do not have significant performance degradation. As the average bit-width decreases, the performance across all tasks decreases gradually. Furthermore, the performance loss of LLMs is more significant at lower average bit-widths. We observed that when the average bit-width remains above 2.8 bits, the performance decline is notably gradual, with sporadic instances of slight improvement. While quantizing the model to the average bit-width of 3.6bit and 2.8bit, performance degradation remains under 0.5% in comparison to AWQ's 4bit and 3bit models. In this case, if the target device is limited to deploying models with an average bit-width of 2.8 bits, the performance of the LLM-MQ 2.8-bit model significantly surpasses the 2-bit quantized models of AWQ and RTN.

The efficiency evaluation of the proposed CUDA kernels. In our implementation, the 2-bit and 3-bit kernels do not show significant speedup compared to the 4-bit kernel. The reason is that for every bit-width, we ensure each thread processes a fixed amount of data. Consequently, during 2-bit and 3-bit quantization, the utilization of GPU bandwidth is less efficient compared to 4-bit quantization. This hinders a higher acceleration ratio. We deploy our INT4 quantized LLaMA2-7B model on NVIDIA T4 GPU with 16 GB memory for evaluation. Compared to the original FP16 model, with the output token number of 32, 128, 512, 1024, and 2048, our quantized model with the proposed CUDA kernels demonstrate end-to-end acceleration factors of 1.27, 1.42, 1.65, 1.64, and 1.45, respectively. For the proposed GEMV Kernel, the 4096×4096 and 4096×11008 GEMV kernels are $2.33\times$ and $2.48\times$ faster than CUBLAS FP16 GEMV kernel.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (No. U19B2019, 61832007, 62104128, U21B2031), Tsinghua EE Xilinx AI Research Fund, Tsinghua-Meituan Joint Institute for Digital Life, and Beijing National Research Center for Information Science and Technology (BNRist).

References

[1] https://developer.nvidia.com/cusparse.

[2] https://github.com/features/copilot.

[3] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[4] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[5] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

[6] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[7] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

[8] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2023.

[9] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[10] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[11] OpenAI. Gpt-4 technical report, 2023.

[12] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models, 2023.

[13] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[14] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu, 2023.

[15] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[16] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[17] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.