Post-training Quantization of Large Language Models with Microscaling Formats

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have distinguished themselves with outstanding performance in complex language modeling tasks, yet they come with significant computational and storage challenges. This paper explores the potential of quantization to mitigate these challenges. We systematically study the combined application of three well-known post-training techniques, SmoothQuant, AWQ, and GPTQ, and provide a comprehensive analysis of their interactions and implications for advancing LLM quantization. We enhance the versatility of these techniques by enabling quantization to microscaling (MX) formats, expanding their applicability beyond their initial fixed-point format targets. We show that combining different PTQ methods enables us to quantize models to 4-bit weights and 8-bit activations using the MXINT format with negligible accuracy loss compared to the uncompressed baseline.

1 Introduction

Large Language Models (LLMs) have emerged as extremely powerful tools to comprehend and generate natural language. However, their intensive computational demand and energy consumption make widespread adoption of these models in everyday tasks to be challenging. One way to address these challenges is post-training quantization, a technique that involves reducing the precision of model parameters and/or activations from the original bit-width to formats with fewer bits. Quantization can significantly reduce the memory footprint and computational requirements of these models, making them more accessible and deployable on a wider range of hardware, including mobile and edge devices. However, previous work has shown that the activations of LLMs with more than 3B parameters are difficult to quantize due to the emergence of outliers with large magnitude, which leads to significant accuracy degradation (Dettmers et al., 2022). To address this issue, Xiao et al. proposed SmoothQuant, a quantization method that smooths out the activation outliers by migrating the quantization difficulty from activations to weights with a mathematically equivalent transformation (Xiao et al., 2023). Lin et al., proposed AWQ, a weight only quantization algorithm that mitigates the quantization error by channel-wise scaling of the salient weights (Lin et al., 2023). Similarly, Frantar et al. proposed GPTQ, a scalable one-shot quantization method that utilizes approximate second-order information to quantize weights (Frantar et al., 2022). In this work, we systematically study the combined application of these three algorithms and provide a comprehensive analysis of their interactions and implications for advancing LLM quantization to various fixed-point and microscaling (MX) formats.

Microscaling format. The microscaling (MX) format for neural net computation was proposed by prior work, first as MSFP (Rouhani et al., 2020) and later subsumed by an emerging industry standard *microscaling formats* (Rouhani et al., 2023b). Specifically, MXINT8 is a microscaling format that enables high-accuracy inference using half the memory footprint and twice the throughput of FP16. It is an emerging industry standard endorsed by Microsoft, AMD, Arm, Intel, Meta, and NVIDIA (Rouhani et al., 2023b) and is already seeing adoption in today's hardware products, such as the Qualcomm cloud AI100 Accelerator (Qualcomm, 2024).

The MX format, as outlined in this paper, is characterized by three key components: 1) the scale factor data type, 2) the data type and precision of individual elements, and 3) the scaling block size. The scale factor is applied uniformly across a block of individual elements. This paper specifically focuses on MX formats employing the *INT* data type for individual elements, thus termed *MXINT*. See Section A of the appendix for more details on the microscaling format.

Notation. Throughout the paper we denote a microscaling (MX) format with scaling block size of b, *8-bit* shared scaling factor, and d bits per element by ***MXINTd-b***. For example, *MXINT6-64* represents an MX format with 6 bits per element, 8 bits shared exponent across 64 values within a block. Similarly, a fixed-point value with i integer bits and no fractional bits is denoted by ***INTi***. For instance, *INT4* specifies a fixed-point value with 4 integer bits and no fractional bits.

Contributions.

- We adopt SmoothQuant, AWQ, and GPTQ to support quantization to microscaling (MX) data formats, extending their compatibility beyond the originally targeted fixed-point formats in the proposed methods.
- We study the interaction of SmoothQuant, AWQ, and GPTQ to quantize state-of-the-art models like Llama2 and Llama3, offering a comprehensive analysis of their impact on advancing LLM quantization. Our findings demonstrate that SmoothQuant and GPTQ, as well as AWQ and GPTQ, are synergistic, especially at more restrictive bit-widths.

2 Quantization algorithms adaptation methodology

Various Post-Training Quantization (PTQ) techniques have emerged to reduce memory bandwidth requirements during LLM inference by quantizing weights and/or activations to lower precisions while maintaining accuracy. In this work, we examine the interaction of three well-known PTQ algorithms for LLMs: GPTQ (Frantar et al., 2022), SmoothQuant (Xiao et al., 2023), and AWQ (Lin et al., 2023). GPTQ is a weight-only quantization technique that reduces quantization error by quantizing the weight matrix column-wise and sequentially updating the unquantized weights using second-order activation Hessians to mitigate the error. SmoothQuant scales both activations and weights to smooth the activation's dynamic range, transferring some of the quantization challenges from activations to weights. AWQ scales weights according to activation magnitudes for improved quantization. For further details on these three algorithms, please refer to Section B of the appendix. The remainder of this section details the generalization of GPTQ, AWQ, and SmoothQuant to support microscaling (MX) quantization, extending their compatibility beyond the originally targeted fixed-point formats in the initially proposed methods.

Algorithm 1 Enhanced GPTQ: Quantize $\mathbf{W}$ given inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}$, block size b_1, and micro-block size b_2.

1: Input: $\mathbf{W}$ ▷ Weight matrix
2: Input: d_{row} ▷ Row dimension of W
3: Input: d_{col} ▷ Column dimension of W
4: Input: b_1 ▷ Block size
5: Input: b_2 ▷ Micro-block size
6: Input: $\mathbf{H}^{-1}$ ▷ Hessian inverse information
7: Variable: $\mathbf{E}$ ▷ Quantization error matrix
8: Output: $\mathbf{Q}$ ▷ Quantized weight matrix
9: Initialize: $\mathbf{Q} \leftarrow 0_{d_{row} \times d_{col}}$
10: Initialize: $\mathbf{E} \leftarrow 0_{d_{row} \times d_{col}}$
11: Initialize: $\mathbf{H}^{-1} \leftarrow Cholesky(\mathbf{H}^{-1})^T$
12: **for** $i = 0, b_1, 2b_1, \ldots$ **do**
13: **for** $j = i, i + b_2, i + 2b_2, \ldots, i + b_1 - 1$ **do**
14: $k \leftarrow j + b_2$ ▷ Helper index
15: $\mathbf{Q}_{:,j:k} \leftarrow quant(\mathbf{W}_{:,j:k})$
16: $\mathbf{E}_{:,j:k} \leftarrow (\mathbf{W}_{:,j:k} - \mathbf{Q}_{:,j:k})([\mathbf{H}^{-1}]_{j:k,j:k})^{-1}$
17: $\mathbf{W}_{:,k:} \leftarrow \mathbf{W}_{:,k:} - \mathbf{E}_{:,j:k}[\mathbf{H}^{-1}]_{j:k,k:}$
18: **end for**
19: $\mathbf{W}_{:,i+b_1:} \leftarrow \mathbf{W}_{:,i+b_1:} - \mathbf{E}_{:,i:i+b_1}[\mathbf{H}^{-1}]_{i:i+b_1,i+b_1:}$
20: **end for**
21: Return: $\mathbf{Q}$

2.1 GPTQ adaptation to MX format

To make GPTQ compatible with the MX format, we modify the algorithm to quantize and update weight values block-wise instead of the originally proposed column-wise updates. Algorithm 1 illustrates the quantization procedure: The weight matrix is divided into blocks (Line 4: b_1), which are further subdivided into micro-blocks (Line 5: b_2). Blocks of consecutive micro-blocks are quantized at each step using inverse Hessian information stored in the Cholesky decomposition (Lines 13-18), and the remaining weights are updated at the end of the step (Line 19). This quantization process is applied recursively to different weight blocks until the entire weight matrix is quantized (Line 12). Note that for quantizing weight matrix to a specific MX format, the micro-block size in the algorithm, b_2, should be a multiple of the block size of the MX format. For more details on the GPTQ algorithm please refer to Section B.1 of the appendix.

2.2 SmoothQuant and AWQ adaptation to MX format

For quantization to the MX format using SmoothQuant and AWQ, we directly calculate per-channel scaling factors to mitigate outliers in activations and/or weights, similar to the approaches proposed in the original paper, and skip the addi-

Act - Wgt bit-width	Format	Method	Llama2-7B	Llama2-13B	Llama3-8B
16-16	FP16, FP16	N/A	**5.12**	**4.57**	**5.54**
8-8	MXINT8-128, MXINT8-128	RTN	5.13	**4.58**	**5.55**
		GPTQ	5.13	**4.58**	**5.55**
		SmoothQuant	**5.12**	**4.58**	**5.55**
		AWQ	**5.12**	**4.58**	**5.55**
		SmoothQuant+	**5.12**	**4.58**	**5.55**
		AWQ+	**5.12**	**4.58**	**5.55**
	INT8, INT8	RTN	**5.15**	**4.60**	**5.62**
		GPTQ	**5.15**	**4.60**	**5.62**
		SmoothQuant	**5.15**	**4.60**	**5.62**
		AWQ	5.17	4.62	5.85
		SmoothQuant+	**5.15**	**4.60**	**5.62**
		AWQ+	5.17	4.62	5.84
8-4	MXINT8-128, MXINT4-128	RTN	5.55	4.82	7.13
		GPTQ	5.45	4.76	6.98
		SmoothQuant	5.60	4.93	7.05
		AWQ	5.43	4.77	6.37
		SmoothQuant+	5.48	4.84	6.51
		AWQ+	**5.37**	**4.73**	**6.16**
	INT8, INT4	RTN	5.91	4.97	8.44
		GPTQ	5.67	4.85	18.64
		SmoothQuant	6.34	5.56	9.13
		AWQ	5.61	4.85	7.33
		SmoothQuant+	5.78	5.12	7.32
		AWQ+	**5.53**	**4.80**	**7.06**

Table 1: Perplexity score on *WikiText-2-test* for the Llama2-7B, Llama2-13B, and Llama3-8B models, when quantized to fixed-point and MX formats using different post-training quantization techniques. Act, Wgt, and RTN denote activation, weight, and round to nearest, respectively. +: GPTQ weight quantization is used. We used *per-channel affine* quantization for the fixed-point formats.

tional calibration phase required for quantization to fixed-point formats (Xiao et al., 2023; Lin et al., 2023). Sections B.2 and B.3 of the appendix provide more details on the SmoothQuant and AWQ algorithms, respectively.

3 Challenges in Studying PTQ Algorithms Interactions

This section highlights the challenges encountered when applying the post-training quantization algorithms studied. We found that some algorithms are incompatible, and for those that are compatible, the order of application is crucial. For instance, both AWQ and SmoothQuant aim to moderate the dynamic range of weight values by calculating scaling factors based on activation and weight tensors. However, despite using different formulas to calculate these scaling factors, we did not observe any benefit from combining the two algorithms. In contrast, GPTQ paired with either AWQ or SmoothQuant proved to be synergistic. When combining GPTQ with SmoothQuant or AWQ, it is essential to first smooth the weight range using SmoothQuant or AWQ, then apply GPTQ to the smoothed weights. Reversing this order results in a significant performance degradation. Section 4 provides more details on the quantization results using different combinations of these PTQ algorithms.

4 Experiments

Setup. We evaluate the impact of the SmoothQuant, AWQ, and GPTQ techniques on quantization of Llama2 and Llama3 models. We employ various fixed-point and MX formats with different bit-widths for our assessment and report the perplexity of the quantized models on *WikiText-2* (Merity et al., 2016). Moreover, we study the impact of applying GPTQ, SmoothQuant, and AWQ individually, as well as the combined effects of GPTQ with AWQ and GPTQ with SmoothQuant. For more details on experiment setup refer to Section C.

Results. Table 1 illustrates perplexity of the quantized *Llama* models (Touvron et al., 2023; Meta, 2024) with three different sizes on WikiText-2-test using various MX and fixed-point formats. For all three models, aggressive quantization to small bit-widths penalizes the model performance, while quantizing to higher bit-widths has negligible effect on perplexity. For example, quantizing *Llama3-8B* to *MXINT8* preserves the baseline perplexity while quantizing to *MXINT4* increases perplexity by 29% to 7.13. Moreover, quantization results using different MX format delivers better perplexity com-

Weight Format	Weight Memory (GB)	Perplexity
FP16	12.35	5.12
INT8	6.18	5.15
MXINT8	6.22	5.12
INT4	3.10	5.55
MXINT4	3.13	5.91

Table 2: Weight Memory and Perplexity score on *WikiText-2-test* for Llama2-7B when quantized to 8-bit and 4-bit fixed-point and microscaling formats.

pared to the fixed-point formats with the same bit-width. For instance, quantizing *Llama2-7B* to *INT4* increases perplexity to 5.91. Enabling AWQ, and GPTQ jointly, reduces it to 5.53, while using *MXINT4* and enabling AWQ and GPTQ we can achieve perplexity of 5.37. Additionally, we found that in all cases except for the quantization of both activations and weights to INT8, AWQ shows superior results compared to SmoothQuant. For the studied models and quantization formats, both SmoothQuant and GPTQ, as well as AWQ and GPTQ, are synergistic, an effect most prominent in more aggressive quantizations.

Similarly, we assess the impact of GPTQ, SmoothQuant, and AWQ on the quantization of the *Llama2, and Llama3* models (Touvron et al., 2023) using MX formats with the block size of *16*. We observe similar trends to those identified in this section. Detailed results of the experiment can be found in the Table 3 of the appendix.

Weight memory footprint study. The objective of a quantization method is to reduce the model size while preserving its accuracy. In this experiment, we quantize *Llama2-7B* to 4-bit and 8-bit data widths, measuring both the weight memory footprint and model perplexity on the WikiText-2-test dataset (Table 2). When quantizing Llama2-7B to *MXINT8*, we achieved a perplexity of 5.12, matching the baseline, while reducing the memory footprint by approximately 2×, from 12.35 GB to 6.22 GB. *INT8* quantization closely follows, achieving a perplexity of 5.15 and memory footprint of 6.18 GB. With more aggressive quantization to 4-bit, both *MXINT4* and *INT4* formats reduced the memory footprint by around 4×. However, the performance gap between these two formats increases to 6.5%, with *MXINT4* showing superior performance.

5 Related Work

Model quantization methods. There are two primary categories of quantization techniques: Quantization-Aware Training (QAT), which leverages backpropagation to update quantized weights (Bengio et al., 2013; Choi et al., 2018; Nagel et al., 2021; Gholami et al., 2022; Liu et al., 2024), and Post-Training Quantization (PTQ), which typically requires no additional training. Quantization-aware training methods cannot easily scale up to quantize giant LLMs. Consequently, PTQ methods are commonly employed for quantizing LLMs (Jacob et al., 2018; Nagel et al., 2020; Wang et al., 2020; Hubara et al., 2021; Li et al., 2021; Deng et al., 2023).

Large Language Model quantization. With the recent open-source releases of language models like Llama (Touvron et al., 2023), researchers are developing cost-effective quantization methods to compress these models for inference: LLM.int8() proposes to preserve activation outliers in higher precision using a mixed INT8/FP16 decomposition (Dettmers et al., 2022). Similarly, SpQR (Dettmers et al., 2023) and OWQ (Lee et al., 2024) propose to retain outlier features that are difficult to quantize in full-precision, while AWQ (Lin et al., 2023) mitigates the quantization error for the outliers using grid-searched channel-wise scaling. QuaRot utilizes Hadamard matrices to effectively rotate LLMs and eliminate outliers in the activations and KV cache (Ashkboos et al., 2024). Lee et al., explored the combined use of AWQ, SmoothQuant, and GPTQ for quantizing LLMs, focusing solely on fixed-point data types in their study (Lee et al., 2023).

6 Conclusion

To summarize, we demonstrated that for the studied models, quantizations using different MX formats deliver better perplexity compared to fixed-point formats with the same bit-width when the per-channel affine quantization scheme is employed. Particularly, for quantization to MXINT8, none of GPTQ, AWQ, or SmoothQuant is necessary to preserve the baseline accuracy. Notably, we found that for Llama2 and Llama3, when quantized to MX formats, AWQ is superior to SmoothQuant. Moreover, AWQ and GPTQ are synergistic, especially, with more aggressive quantization to 4-bit.

Throughout the paper, we have shown that by utilizing AWQ, and GPTQ and applying MX formats we can quantize the Llama2 and Llama3 models to 4-bit weights and 8-bit activations, with minimal perplexity degradation.

7 Limitations

With quantization of LLMs, we make the models accessible to more people, which generally comes with security risks, such as potential misuse for generating harmful content. This highlights the need for further investigation into responsible AI practices. On the technical side, due to space and computational resource constraints, we have only reported results for text generation with Llama2 and Llama3 models up to 13B parameters on the WikiText-2 dataset. Further investigation of broader models, datasets, and tasks remains for future work.

References

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. QuaRot: Outlier-free 4-bit inference in rotated llms. *arXiv:2404.00456*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90% chatgpt quality.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: Parameterized clipping activation for quantized neural networks. *arXiv:1805.06085*.

Zihao Deng, Xin Wang, Sayeh Sharify, and Michael Orshansky. 2023. Mixed-precision quantization with cross-layer dependencies. *arXiv:2307.05657*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv:2208.07339*.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. SpQR: A sparse-quantized representation for near-lossless llm weight compression. *arXiv:2306.03078*.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv:2210.17323*.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC.

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. 2021. Accurate post training quantization with small calibration sets. In *International Conference on Machine Learning*, pages 4466–4475.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. OWQ: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13355–13364.

Janghwan Lee, Minsoo Kim, Seungcheol Baek, Seok Hwang, Wonyong Sung, and Jungwook Choi. 2023. Enhancing computation efficiency in large language models through weight and activation quantization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14726–14739.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. 2021. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv:2102.05426*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. AWQ: Activation-aware weight quantization for llm compression and acceleration. *arXiv:2306.00978*.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024. SpinQuant: Llm quantization with learned rotations. *arXiv:2405.16406*.

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

Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv:2106.08295*.

Qualcomm. 2024. Qualcomm Cloud AI 100 Accelerates Large Language Model Inference by 2x Using Microscaling (Mx) Formats.

Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Jasmine Klar, Eric Chung, Yuan Yu, Michael Schulte, Ralph Wittig, Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Summer Deng, Maxim Naumov, Paulius Micikevicius, Michael Siu, and Colin Verrilli. 2023a. Ocp microscaling formats (mx) specification. *Open Compute Project*.

Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, et al. 2020. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. *Advances in neural information processing systems*, 33:10271–10281.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. 2023b. Microscaling data formats for deep learning. *arXiv:2310.10537*.

Stability AI. 2023. Stable Beluga.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A strong, replicable instruction-following model.

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

Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. 2020. Towards accurate post-training network quantization via bit-split and stitching. In *International Conference on Machine Learning*, pages 9847–9856.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

A Microscaling data format

The Microscaling (MX) data format, initially introduced in 2020 as Microsoft Floating Point (MSFP, Rouhani et al. 2020), has since evolved and gained widespread adoption among leading industry players, including Microsoft, AMD, Intel, Meta, Nvidia, and Qualcomm (Rouhani et al., 2023b).

The core concept of the MX format is centered around the MX block, where a vector of k numbers share a single scale (X) while retaining individual elements $\{P_i\}_{i=1}^{k}$, as shown in Figure 1. The actual value for each of the k numbers in the block can be represented as $v_i = XP_i$ (Rouhani et al., 2023b). The data format for the single scale and the data format for individual elements can be independent of each other, while the data format for individual elements needs to be consistent across the k elements in the block (Rouhani et al., 2023a). An MX block can be represented in $(w + kd)$ bits, where w is the number of bits for shared scale X and d is the number of bits for each individual element. Consequently, the MX format is characterized by three main components:

1. Data type of scale X
2. Data type of elements P_i
3. Scaling block size k

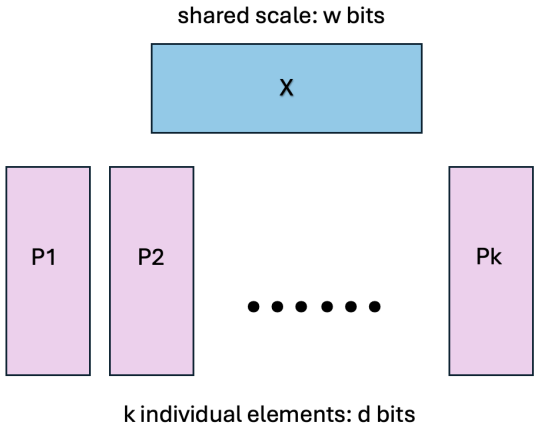

Figure 1: Illustration of an MX block.

The MX format has proven to be highly effective in addressing the challenges of balancing hardware efficiency, model accuracy, and user experience in machine learning applications. According to empirical results, 8-bit MX formats can perform inference directly on FP32 pretrained models with minimal accuracy loss, eliminating the need for additional calibration or finetuning

(Rouhani et al., 2023b). Furthermore, when using 6-bit MX formats, the inference accuracy remains close to that of FP32 models, especially after applying quantization-aware fine-tuning or post-training quantization methods (Rouhani et al., 2023b). Remarkably, the MX format also enables the training of large transformer models using sub-8-bit precision for weights, activations, and gradients, achieving accuracy comparable to FP32 without requiring changes to the training process (Rouhani et al., 2023b).

B Post training quantization algorithms

B.1 GPTQ

GPTQ is a post-training quantization (PTQ) method that uses second-order Hessian information for weight quantization in LLMs (Frantar et al., 2022). It employs layer-wise quantization for each layer l in the network, seeking quantized weights $\hat{\mathbf{W}}_l$ that make the outputs $(\hat{\mathbf{W}}_l\mathbf{X}_l)$ closely approximate those of the original weights $(\mathbf{W}_l\mathbf{X}_l)$. In other words, GPTQ aims to find (Frantar et al., 2022):

$$\texttt{argmin}_{\hat{\mathbf{W}}_l} ||\mathbf{W}_l\mathbf{X}_l - \hat{\mathbf{W}}_l\mathbf{X}_l||_2^2 \tag{1}$$

To solve equation 1, GPTQ quantizes each row of the weight matrix, $\mathbf{W}$, independently, focusing on a single weight per row at a time. It consistently updates all not-yet-quantized weights to offset the error introduced by quantizing a single weight. Since the objective function in equation 1 is quadratic, its Hessian $\mathbf{H}$ can be calculated using the following formula, where F denotes the set of remaining full-precision weights:

$$\mathbf{H}_F = 2\mathbf{X}_F\mathbf{X}_F^T \tag{2}$$

Given $\mathbf{H}$, the next to be quantized weight, w_q, and the corresponding update of all remaining weights in F, δ_F, are given by the following formulas, where $\texttt{quant}(w)$ rounds w to the nearest quantized value (Frantar et al., 2022):

$$\begin{aligned} w_q &= \texttt{argmin}_{w_q} \frac{(w_q - \texttt{quant}(w_q))^2}{[\mathbf{H}_F^{-1}]_{qq}} \\ \delta_q &= -\frac{w_q - \texttt{quant}(w_q)}{[\mathbf{H}_F^{-1}]_{qq}} \cdot (\mathbf{H}_F^{-1})_{:,q} \end{aligned} \tag{3}$$

For all rows of $\mathbf{W}$, GPTQ quantizes weights in the same order. This accelerates the process, as certain computations need to be performed only once for each column rather than once for each weight. Additionally, the vectorized implementation of GPTQ enables processing multiple rows of $\mathbf{W}$ simultaneously. For more details on the GPTQ algorithm refer to Frantar et al.'s work (Frantar et al., 2022).

B.2 SmoothQuant

SmoothQuant (SQ) is a quantization method that targets both activations and weights of a model (Xiao et al., 2023). In this approach, the activation of a linear layer is scaled by a per-channel smoothing factor $s \in R^{C_i}$ to minimize quantization errors. Simultaneously, the weight of the layer is adjusted in the opposite direction to maintain the mathematical equivalence of the linear layer:

$$\mathbf{Y} = (\mathbf{X}\texttt{diag}(s)^{-1}) \cdot (\texttt{diag}(s)\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}} \tag{4}$$

In Equation 4, $\mathbf{X}$ is the original input activation with outliers, and $\hat{\mathbf{X}} = \mathbf{X}\texttt{diag}(s)^{-1}$ is the smoothed activation. To minimize the quantization error of the input activation, the smoothing factor is selected such that all channels of the smoothed input activation have the same maximum magnitude. Accordingly, s is set to:

$$s_j = \texttt{max}(|\mathbf{X}_j|), \quad j = 1, 2, ..., C_i \tag{5}$$

Where C_i is the number of input channels in the input activation and j corresponds to j^{th} input channel. Note that since the range of activations varies for different input samples, the maximum value of each channel is estimated using 128 calibration samples from the calibration dataset (see Section C for more details). By dividing the input activation by the the scaling factor of Equation 5, all channels of the scaled input activation would have the same range, making quantization of the scaled tensor to be very easy. However, this will migrate the difficulty of the quantization completely to the weight side of a linear layer. To address this issue, Xiao et al. proposed a scaling formula that balances the quantization difficulty of activations and weights:

$$s_j = \texttt{max}(|\mathbf{X}_j|)^{\alpha} / \texttt{max}(|\mathbf{W}_j|)^{1-\alpha}, \quad j = 1, 2, ..., C_i \tag{6}$$

Where α is a hyper-parameter that controls how much quantization difficulty we want to migrate from activations to weights. For more details on the SmoothQuant algorithm refer to Xiao et al.'s work (Xiao et al., 2023).

B.3 AWQ

Activation-aware Weight Quantization (AWQ), is a weight-only quantization method for LLMs (Lin et al., 2023). In this algorithm, a small fraction (i.e.,

Format	Method	Llama2-7B	Llama2-13B	Llama3-8B
A:FP16, W:FP16	N/A	**5.12**	**4.57**	**5.54**
A:MXINT8-16 W:MXINT8-16	RTN	**5.12**	4.58	**5.54**
	GPTQ	**5.12**	4.58	**5.54**
	SmoothQuant	**5.12**	**4.57**	**5.54**
	AWQ	**5.12**	4.58	**5.54**
	SmoothQuant+	**5.12**	**4.57**	**5.54**
	AWQ+	**5.12**	4.58	**5.54**
A:MXINT8-16 W:MXINT4-16	RTN	5.40	4.72	6.18
	GPTQ	5.41	**4.68**	5.93
	SmoothQuant	5.33	4.74	6.14
	AWQ	5.30	4.70	6.03
	SmoothQuant+	5.28	4.69	5.95
	AWQ+	**5.27**	**4.68**	**5.90**

Table 3: Perplexity score on *WikiText-2-test* for the Llama models, when quantized to MX formats with the block size of *16* using different post-training quantization techniques. *A*, *W*, and RTN denote activation, weight, and round to nearest, respectively. +: GPTQ weight quantization is used.

0.1%-1%) of salient weight channels are scaled up to reduce their relative quantization error:

$$\mathbf{Y} = \mathbf{X}\mathbf{W} \approx \mathbf{X}\hat{\mathbf{W}} \approx (\mathbf{X}/s)(s\hat{\mathbf{W}}) \tag{7}$$

In Equation 7, s is a per-channel scaling factor for the salient weights. To determine the salient weights, AWQ refers to the activation distribution instead of the weight distribution, as weight channels corresponding to the outlier activations are more salient than other weights. The per-channel scaling factor is calculated using the following formula:

$$s = s_{\mathbf{X}}^{\alpha}, \quad \alpha \in [0, 1] \tag{8}$$

Where $s_{\mathbf{X}}$ is the average magnitude of activation (per-channel), and α is a hyper-parameter which balances the protection of salient and non-salient channels. For more details on AWQ refer to Lin's et al. work (Lin et al., 2023)

C Experiment Setup

Models. We evaluated various quantization methods using the Llama2, and Llama3 families (Touvron et al., 2023; Meta, 2024). These LLMs are widely accepted in the machine learning community for their superior performance compared to other open-source LLMs (Dettmers et al., 2022; Frantar et al., 2022; Xiao et al., 2023; Lin et al., 2023). Llama also serves as the foundation for many popular open-source models such as Alpaca (Taori et al., 2023), Vicuna (Chiang et al., 2023), and Stable Beluga (Stability AI, 2023).

Datasets. Following previous work (Dettmers et al., 2022; Xiao et al., 2023; Frantar et al., 2022; Lin et al., 2023; Dettmers and Zettlemoyer, 2023; Yao et al., 2022), we measured the perplexity of quantized language models on *WikiText-2* (Merity et al., 2016) as perplexity can stably reflect the performance of LLMs (Dettmers and Zettlemoyer, 2023; Lin et al., 2023). Unless otherwise stated, the *test* split of the dataset is used to evaluate the models.

Quantization formats. We evaluated models using different microscaling and fixed-point quantization formats. For the fixed-point quantization, we calibrated the models using 128 random input sentences from *WikiText-2-train* to estimate the dynamic range of activations. We utilized *MinMaxObserver* to find the range of activations, and calculated the zero-point and the scale parameters for the activations and weights in per-channel granularity levels. For the MXINT format, unless otherwise specified, the blocking dimension of a given tensor is the last dimension.

Activation smoothing. We calculated the per-channel scaling factor for activations and weights using the formula stated in Equation 4. As in the previous work, we consistently use a migration strength (α) value of 0.5 across all models throughout the paper. To calculate the scaling factors, we gathered the statistics of activations using 128 random sentences from the *WikiText-2-train* dataset. Once we calculated the scaling factors, we used the same values to evaluate the models with different quantization formats.

Targeted layers. Similar to the previous work (Xiao et al., 2023), we apply smoothing on the input activation of the self-attention and the feed-forward layers of LLMs. Unless stated otherwise, we transform all *Linear* layers to the specified quantization format while keeping the activation/weight in the original format for other layers including *GELU*, *Softmax*, and *LayerNorm*.