AFPQ: Asymmetric Floating Point Quantization for LLMs

Yijia Zhang[†,*], **Sicheng Zhang**[†,*], **Shijie Cao**[‡], **Dayou Du**[§], **Jianyu Wei**[¶], **Ting Cao**[‡], **Ningyi Xu**[†,†]
[†]Shanghai Jiao Tong University [‡]Microsoft Research Asia
[§]The Hong Kong University of Science and Technology (Guangzhou)
[¶]University of Science and Technology of China
{zhangyijia, zhangsicheng, xuningyi}@sjtu.edu.cn, {shijiecao, ting.cao}@microsoft.com,
ddu487@connect.hkust-gz.edu.cn, noob@mail.ustc.edu.cn

Abstract

Large language models (LLMs) show great performance in various tasks, but face deployment challenges from limited memory capacity and bandwidth. Low-bit weight quantization can save memory and accelerate inference. Although floating-point (FP) formats show good performance in LLM quantization, they tend to perform poorly with small group sizes or sub-4 bits. We find the reason is that the absence of asymmetry in previous FP quantization makes it unsuitable for handling asymmetric value distribution of LLM weight tensors. In this work, we propose asymmetric FP quantization (AFPQ), which sets separate scales for positive and negative values. Our method leads to large accuracy improvements and can be easily plugged into other quantization methods, including GPTQ and AWQ, for better performance. Besides, no additional storage is needed compared with asymmetric integer (INT) quantization. The code is available at https://github.com/zhangsichengsjtu/AFPQ.

1 Introduction

LLMs have significantly advanced language understanding, generation, and reasoning (Touvron et al., 2023; Rozière et al., 2023; Zhang et al., 2022). However, the increasing size of LLMs poses great pressure on memory capacity and bandwidth during deployment. Low-bit quantization is a widely used solution to decrease both memory capacity and bandwidth requirements. To effectively accommodate LLMs, new quantization methods have been proposed, such as GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2023). These methods quantize LLMs with low-bit INT formats.

Recent studies suggest utilizing low-bit FP formats, such as FP4 and NF4 (Dettmers et al., 2021), in place of INT can lead to improved quantization accuracy of LLMs (Dettmers and Zettlemoyer, 2023; Zhang et al., 2023; Wu et al., 2023). This improvement is attributed to the non-uniform distribution of low-bit FP formats, which more effectively align with LLM weights, characterized by mostly smaller values and a long tail of larger, significant ones. Although generally superior, FP formats tend to be worse than INT when quantization with small group sizes or sub-4 bits.

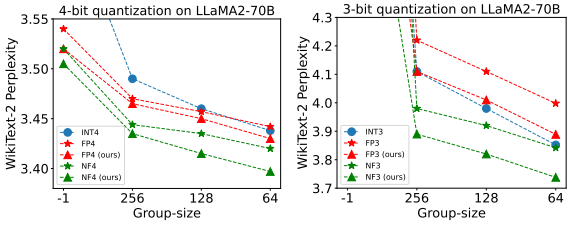

Figure 1: On LLaMA2-70B (Touvron et al., 2023), our asymmetric FP quantization reduces the WikiText-2 perplexity (the lower the better) in both 3-bit and 4-bit FP quantization (NF, short for NormalFloat, is an advanced type of FP formats). We use group-size '-1' to represent per-channel quantization.

We identify this is caused by the absence of asymmetry in FP quantization. Given that most weight tensors naturally exhibit asymmetric distributions, it is not suitable to quantize them with standard low-bit FP values, which have a symmetric distribution. Furthermore, we find the conventional methods used in asymmetric INT quantization, such as scale and zero-point adjustments, do not perform well in the context of FP quantization.

In this work, we propose asymmetric floating point quantization (AFPQ), a simple yet effective approach to fit the weight asymmetry in LLMs. Unlike previous symmetric FP quantization, which uses a uniform scale for positive and negative values within a weight group, AFPQ sets seperate scales for positive and negative values. AFPQ ensures that the rescaled FP values can better match

[*]Equally contributed.
[†]Corresponding author.

Findings of the Association for Computational Linguistics ACL 2024, pages 28–36
August 11-16, 2024

the original weight values, thereby enhancing quantization accuracy in LLMs. In Figure 1, our AFPQ with FP and NP formats show better results in both 3-bit and 4-bit round-to-neare (RTN) quantization. Moreover, AFPQ requires no additional storage compared with asymmetric INT quantization.

Our contributions can be summarized as follows:

1. We identify that the subpar quantization accuracy of FP for LLMs is caused by the asymmetry of weights within the quantization group.

2. We introduce the asymmetric FP quantization, which can enhance FP quantization performance significantly.

3. AFPQ can work as a plugin to other quantization methods, such as GPTQ and AWQ. We integrate AFPQ with these methods.

2 Background and Motivation

Post Training Quantization (PTQ) for LLMs. There are two PTQ methods for LLMs: 1) Quantizing both weights (W) and activations (A), for example, W8A8 quantization (Dettmers et al., 2022; Xiao et al., 2023); 2) W-only quantization, for example, W4A16 one (Dettmers and Zettlemoyer, 2023). This article focuses on the W-only method. The naive W-only method is RTN. The advanced methods include GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2023). GPTQ uses second-order information to compensate for quantization error, while AWQ scales salient weights before quantization. Both methods use INT for quantization.

Lack of asymmetry for FP quantization In the weight tensors of LLMs, outliers often appear (Lin et al., 2023; Dettmers et al., 2023). Due to the randomness of outliers, many weight tensors exhibit an asymmetric distribution of maximum and minimum values. In Figure 2, we have randomly selected some LLaMA2 weight groups. It can be observed that more than 50% of the groups exhibit an asymmetric value distribution.

For INT, asymmetric quantization with one zero-point (for range translation) and one scale (for scaling) for each weight group can fit the asymmetric tensor distribution well. However, zero-point scale based asymmetric quantization is not suitable for FP because it may move the dense value area far from zero, making FP lose advantages over INT format. For example, if we apply asymmetric INT quantization to asymmetric weights in Figure 3,

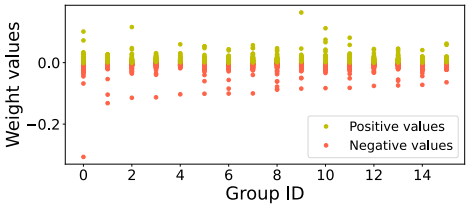

Figure 2: Randomly selected weight groups (group-size is 128) from LLaMA2-7B. The maximum and minimum values in many groups are not symmetric about zero.

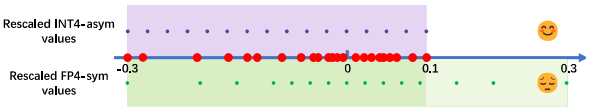

Figure 3: Red points are original asymmetric weight values. Recaled INT4-asym covers the weight values well, but the coverage of rescaled FP4-sym exceeds the range of weights, thus wasting values in FP formats.

the original weights will be fully covered by the rescaled asymmetric INT (INT-asym) values. However, when applying previous FP quantization (only one scale for scaling)[1][2], the range of rescaled symmetric FP (FP-sym) values exceeds the range of original weights, leading to a waste of the expressive ability of some FP values. Therefore, asymmetric FP quantization should be introduced.

3 Asymmetric Floating Point Quantization

Quantization methods. To make FP quantization applicable to the asymmetric distribution of LLM weights, an intuitive approach is to apply the method with one scale and zero-point used in asymmetric INT quantization to FP quantization. However, this approach would shift the dense number area of FP from zero to the left of zero, eliminating the advantages of using FP formats, shown in the purple block of Figure 4.

To preserve the advantage of FP formats, we propose asymmetric FP quantization with two separate scales, one for positive numbers and another for negative numbers in each weight group. In Algorithm 1, we show our FP quantization process of a weight group. Each FP16 weight will be divided by scale_pos or scale_neg depending on its sign, then it can be rounded to the nearest FP4 values.

In this way, the rescaled FP-asym values can better fit the distribution of original weights (Fig-

[1] https://github.com/TimDettmers/bitsandbytes

[2] https://github.com/openppl-public/ppq

Algorithm 1 Quantization methods from FP16 weight groups to **FP4-asym** ones with two scales

input : Wgrp_16bit
output : Wgrp_4bit
constant : range # The range of FP4 formats
# Get the positive scale and negative one for this weight group
W_max = max(Wgrp_16bit)
$scale\_pos = \frac{W\_max}{range/2}$
W_min = min(Wgrp_16bit)
$scale\_neg = \frac{-W\_min}{range/2}$
# Wgrp_4bit is an empty tensor with the same shape of Wgrp_16bit
for *index in shape(Wgrp_16bit)* **do**
  if *Wgrp_16bit[index] > 0* **then**
    $W\_tmp = \frac{Wgrp\_16bit[index]}{scale\_pos}$
  end
  else
    $W\_tmp = \frac{Wgrp\_16bit[index]}{scale\_neg}$
  end
  # Round W_tmp to the nearest FP4 values
  Wgrp_4bit[index] = Round(W_tmp, 'FP4')
end
return *Wgrp_4bit*

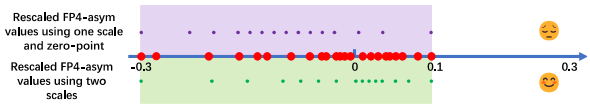

Figure 4: Red points are original asymmetric weight values. Recaled FP4-asym using two scales gathers more values near zero than the FP4-asym using one scale and zero-point, which aligns with the distribution of LLMs weights more.

ure 4 green block). Besides, no storage overhead is incurred compared with asymmetric INT quantization (both need two parameters for one group).

Application to SOTA methods. As AFPQ operates on each individual sub-tensor or group, it can work as a plugin to other high-level quantization algorithms such GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2023). To demonstrate the applicability, we integrate AFPQ with SOTA methods GPTQ and AWQ for better quantization accuracy.

Inference system implementation. In W-only low-bit inference systems, the low-bit weights should be de-quantized to FP16 ones before their MatMul with FP16 activation tensors. Therefore, the system overhead can come from different de-quantization process. We've implemented two version of inference system for our AFPQ method, the naive one and the optimized one.

Instead of the naive de-quantization method of multiplying each weight with the positive scale or negative one depending on its sign (detailed in Appendix E), we generate conversion LUT for each weight group to largely reduce the number of sign judgments and scale multiplications. In Algorithm 2, we take FP4-asym as an example to show the optimized de-quantization process of a weight group. For each weight group, we first obtain its exclusive conversion LUT lut_grp, then we can map Wgrp_4bit to Wgrp_16bit using lut_grp. After all 4-bit weight groups in a weight tensor have been converted to the 16-bit ones, it can be matmuled with the activation tensor. This approach reduces the number of sign judgments during inference to be equal to the size of lut_grp. If the quantization group size is 128, the overhead could be reduced by 8x because the number of sign judgments in a quantization group decreases from naive 128 (the number of weights in a group) to 16 (the size of lut_grp for 4-bit quantization). Moreover, the system overhead can be reduced more if the group-size gets larger or the bit-width gets smaller.

Algorithm 2 Optimized de-quantization of weight groups from FP4-asym (two scales) to FP16 ones

input : Wgrp_4bit, scale_pos, scale_neg
output : Wgrp_16bit
constant : FP4toFP16_lut # Standard LUT of mapping FP4 values to IEEE FP16 ones
lut_grp = { } # Conversion LUT for each weight group
for *k, v* ***in*** *FP4toFP16_lut* **do**
  # Generating lut_grp depending on the value signs
  lut_grp[k] = v * scale_pos **if** v > 0 **else** v * scale_neg
end
# Mapping Wgrp_4bit to Wgrp_16bit using lut_grp
Wgrp_16bit = [lut_grp[w_4bit] **for** w_4bit **in** Wgrp_4bit]
return Wgrp_16bit

4 Experiments

Experimental setup. We focus on 4-/3-bit PTQ since they can mostly preserve the performance of LLMs (Dettmers and Zettlemoyer, 2023). The formats we use are shown in Appendix B. We select LLaMA2 (Touvron et al., 2023) models for basic evaluation because of their superior performance among open-sourced LLMs (Zhang et al., 2022; Scao et al., 2022). We also include WizardCoder (Luo et al., 2023) and MetaMath (Yu et al., 2023) models for further evaluation. We conduct quantization experiments on AutoGPTQ project[3]. We use 'g-1' to represent per-channel quantization in this section.

Comparisons between AFPQ with two scales and the one with scale + zero-point. We evaluate LLaMA2-70B with these two methods using the

[3] https://github.com/PanQiWei/AutoGPTQ

Table 1: WikiText-2 perplexity and MMLU average accuracy on LLaMA2 models after **4-bit** RTN quantization.

		LLaMA2-7B				LLaMA2-13B				LLaMA2-70B			
		g-1	g256	g128	g64	g-1	g256	g128	g64	g-1	g256	g128	g64
WikiText-2 ↓	FP16	5.47				4.88				3.32			
	INT4	6.12	5.75	5.72	5.67	5.20	5.02	4.98	4.97	3.67	3.49	3.46	3.44
	NF4-sym	5.87	5.68	5.66	5.65	5.09	5.01	4.99	4.98	3.52	3.44	3.44	3.42
	NF4-asym	**5.77**	**5.67**	**5.66**	**5.64**	**5.07**	**5.00**	**4.98**	**4.97**	**3.51**	**3.44**	**3.42**	**3.40**
MMLU(%) ↑	FP16	46.58				55.38				69.58			
	INT4	40.31	43.67	45.28	45.59	52.92	54.09	54.33	54.44	67.82	68.43	68.32	68.53
	NF4-sym	43.04	**43.94**	45.09	45.70	53.59	54.37	54.58	54.84	**67.96**	68.41	68.66	**69.18**
	NF4-asym	**45.05**	43.53	**45.42**	**46.12**	**54.10**	**54.93**	**54.71**	**55.03**	67.78	**68.64**	**68.81**	68.93

Table 2: WikiText-2 perplexity and MMLU average accuracy on LLaMA2 models after **3-bit** RTN quantization.

		LLaMA2-7B				LLaMA2-13B				LLaMA2-70B			
		g-1	g256	g128	g64	g-1	g256	g128	g64	g-1	g256	g128	g64
WikiText-2 ↓	FP16	5.47				4.88				3.32			
	INT3	542.80	7.10	6.66	6.40	10.68	5.67	5.52	5.39	7.53	4.11	3.98	3.85
	NF3-sym	74.27	6.74	6.45	6.26	7.73	5.53	5.43	5.35	8.38	3.98	3.92	3.85
	NF3-asym	**9.85**	**6.42**	**6.29**	**6.15**	**6.53**	**5.46**	**5.35**	**5.27**	**5.42**	**3.89**	**3.82**	**3.74**
MMLU(%) ↑	FP16	46.58				55.38				69.58			
	INT3	25.22	37.46	38.50	40.06	27.79	48.91	51.23	50.77	34.39	64.77	65.05	66.16
	NF3-sym	26.20	36.85	38.61	38.47	38.96	49.84	50.97	51.72	40.63	**66.40**	65.90	**66.92**
	NF3-asym	**30.31**	**38.58**	**41.61**	**41.11**	**42.74**	**52.31**	**52.60**	**53.3**	**56.07**	66.23	**66.78**	66.43

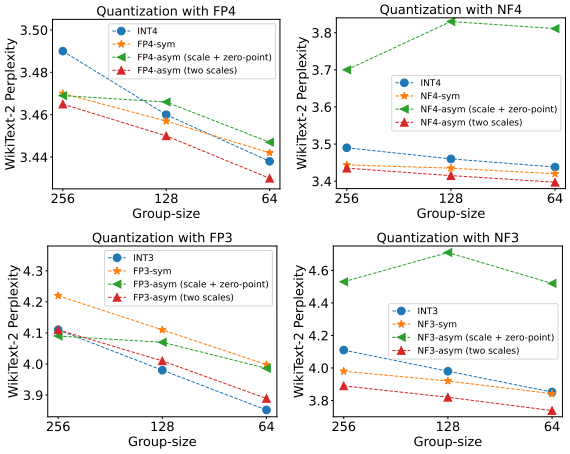

Figure 5: When quantizing LLaMA2-70B, FP-asym and NF-asym quantization with two scales shows lower perplexity (ppl) on WikiText-2 (the lower the better).

RTN quantization on WikiText-2 perplexity following Frantar et al. (2022). As shown in Figure 5, quantization using FP-asym with two scales brings better quantization accuracy in both 4-bit and 3-bit grouped quantization for FP and NF formats. For simplicity, asymmetric FP quantization mentioned below is the one using two scales. Note that the performance of the FP3 formats is still worse than INT3, this is because FP3 can only represent 7 values for quantization, whereas INT3 and NF3 can represent 8. To ensure a fair comparison, the remaining quantization experiments in this section are conducted using INT and NF formats.

Results across various group-sizes and bit-widths using RTN quantization. To demonstrate the generality of our method, we evaluate our AFPQ using RTN on LLaMA2 models with different bit-widths and group-sizes. The evaluation focuses on WikiText-2 and MMLU benchmark with in-context learning (5-shot) following Lin et al. (2023). We provide the 4-bit and 3-bit results in Table 1 and Table 2, respectively. For both bit-widths, quantization with NF-asym achieves better or on-par results in all settings. Detailedly, NF3-asym with group-size 128 can lead to 3% MMLU accuracy improvement for LLaMA2-7B (a model size well-suited for edge deployments (Dettmers et al., 2023)) compared with INT3 and NF3-sym quantization. The conclusions of FP4 and FP3 are similar to NF formats, which are shown in Appendix D.

Table 3: WikiText-2 perplexity and MMLU average accuracy on LLaMA2-70B after we integrate asymmetric FP quantization with **GPTQ**.

		LLaMA2-70B			
		g-1	g256	g128	g64
WikiText-2 ↓ FP16: 3.32	INT3	4.57	3.88	3.77	3.67
	NF3-sym	4.16	3.77	3.72	3.67
	NF3-asym	**4.07**	**3.73**	**3.66**	**3.61**
MMLU(%) ↑ FP16: 69.58	INT3	60.10	66.65	67.25	67.75
	NF3-sym	64.45	67.03	67.42	67.72
	NF3-asym	**64.95**	**67.33**	**68.05**	**68.03**

Results of applying AFPQ to GPTQ. Although being effective PTQ methods, there is still an accuracy gap between FP16 LLMs and quantized ones using GPTQ. In Table 3, We try to improve GPTQ by replacing the INT3 quantization with NF3-asym

Table 4: WikiText-2 perplexity and MMLU average accuracy on LLaMA2-70B after we integrate asymmetric FP quantization with **AWQ**.

		LLaMA2-70B			
		g-1	g256	g128	g64
WikiText-2 ↓ FP16: 3.32	INT3	4.91	4.10	3.87	3.72
	NF3-sym	4.26	4.03	3.83	3.71
	NF3-asym	**4.18**	**3.87**	**3.74**	**3.65**
MMLU(%) ↑ FP16: 69.58	INT3	59.08	65.15	66.45	67.40
	NF3-sym	62.60	65.02	65.88	**67.66**
	NF3-asym	**63.57**	**66.56**	**67.00**	67.41

ones. GPTQ with NF3-asym shows improved accuracy in all group-sizes. For group-size 128, the commonly used setting in Frantar et al. (2022); Lin et al. (2023), our method can reduce WikiText-2 ppl by 0.11 from GPTQ-INT3, which should be considered significant.

Results of applying AFPQ to AWQ. In Table 4, we also improve AWQ by replacing the INT3 quantization with NF3-asym ones. Note that the INT3 or NF3 baseline is already strong, our NF3-asym one can still raise the performance to a higher level.

Table 5: Evaluation results on WizardCoder-7B and MetaMath-7B after 3-bit AWQ with group-size of 64. For WizardCoder-7B, we show the percentage of pass rates on the HumanEval. For MetaMath-7B, we show the testing accuracy on gsm8k.

	FP16	INT3	NF3-sym	NF3-asym
WizardCoder-7B ↑	57.31	47.56	45.12	**52.43**
MetaMath-7B ↑	66.41	63.52	60.86	**64.53**

As quantization may hurt LLMs' performance in difficult downstream tasks, such as coding and mathematical ones, we evaluate AWQ with NF3-asym on WizardCoder-7B and MetaMath-7B in Table 5. NF3-asym helps reach the highest quantization accuracy in both tasks. Notably, the accuracy of quantized WizardCoder-7B is enhanced by 4.87% compared with AWQ-INT3, which strongly proves the effectiveness of our method.

Table 6: Inference latency (ms) of LLaMA2-7B and LLaMA2-13B under different formats. The group-size is 128.

	FP16	INT4	NF4-sym	NF4-asym (naive)	NF4-asym (optimized)
LLaMA2-7B	415.06	174.29	187.23	265.54	215.42
LLaMA2-13B	788.01	309.87	317.15	485.42	352.00

Efficiency evaluation. We have implemented an inference system prototype to verify the overhead. Since low-bit NF-based kernels have not been proposed in previous work, we develop these kernels and integrate them into FasterTransformer[4] framework. We measure the end-to-end latency of LLaMA2 models on a single A6000 GPU. We keep the batch size to be 1, the input sequence length to be 128, and a uniform output token count of 20. In Table 6, our AFPQ method with NF4-asym (optimized) reduces the latency of NF4-asym (naive) largely and achieves up to 2.24x speedup compared with FP16 baseline.

5 Conclusion

In this study, we identify that the lack of asymmetry in previous FP quantization can lead to poor quantization for LLM weight tensors with asymmetric distribution. To solve the problem, we propose asymmetric FP quantization which sets separate scales for positive and negative values. Besides, our method can be easily plugged into other effective methods, including GPTQ and AWQ, for performance improvements. In the future, we plan to push towards other promising formats such as Posit (Gustafson and Yonemoto, 2017) for better LLM quantization results.

6 Limitations

Although we pay efforts to design an efficient FP-asym-based inference system in Section 3, it still incurs inference overhead compared with INT4-/NF4-sym-based system shown in Section 4. We believe the gap can be narrowed with advanced kernel optimizations or specific hardware design, which we leave it as a future work.

References

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

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

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2021. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos,

[4] https://github.com/NVIDIA/FasterTransformer

Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint 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. PMLR.

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*.

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.

John L Gustafson and Isaac T Yonemoto. 2017. Beating floating point at its own game: Posit arithmetic. *Supercomputing frontiers and innovations*, 4(2):71–86.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

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.

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 preprint arXiv:2306.00978*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Xiaoxia Wu, Zhewei Yao, and Yuxiong He. 2023. Zeroquant-fp: A leap forward in llms post-training w4a8 quantization using floating-point formats. *arXiv preprint arXiv:2307.09782*.

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. PMLR.

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.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

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

Yijia Zhang, Lingran Zhao, Shijie Cao, Wenqiang Wang, Ting Cao, Fan Yang, Mao Yang, Shanghang Zhang, and Ningyi Xu. 2023. Integer or floating point? new outlooks for low-bit quantization on large language models. *arXiv preprint arXiv:2305.12356*.

Appendix

A Model quantization methods

Quantization is a process that reduces the precision of Deep Neural Network (DNN) weights to decrease model size and accelerate model inference (Han et al., 2015; Jacob et al., 2018). Existing quantization methods can be broadly categorized into two types: Post Training Quantization (PTQ) and Quantization Aware Training (QAT) (Bengio et al., 2013; Gholami et al., 2022). QAT necessitates model training, which can be expensive, whereas PTQ does not. We focus on PTQ in this work.

Besides, the group-wise quantization we use in this work is one kind of fine-grained quantization methods of a weight tensor. It can lead to better quantization accuracy compared with coarse-grained one such as per-weight or per-channel quantization methods.

B Low-bit formats used in this work

The current mainstream quantization formats include low-bit INT and FP (Yao et al., 2022; Wu et al., 2023). INT is uniformly distributed, while FP, with its exponent and mantissa design, has a distribution that is dense near zero and sparse far from it. In addition, some new formats have also emerged, such as NF (Dettmers et al., 2021), a new type of FP formats designed based on normal number distribution.

In this work, we use FP4 E2M1 and FP3 E1M1 formats. Both excludes NaN and Inf following Zhang et al. (2023). For NF formats, we use the values from Bitsandbytes[5]. The exact values of the INT, FP and NF formats used in our experiments are as follows:

INT4: [-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

FP4: [-6, -4, -3, -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, 3, 4, 6]

NF4: [-1, -0.6961928009986877, -0.5250730514526367, -0.39491748809814453, -0.28444138169288635, -0.18477343022823334, -0.09105003625154495, 0, 0.07958029955625534, 0.16093020141124725, 0.24611230194568634, 0.33791524171829224, 0.44070982933044434, 0.5626170039176941, 0.7229568362236023, 1]

INT3: [-4, -3, -2, -1, 0, 1, 2, 3]

FP3: [-4, -2, -1, 0, 1, 2, 4]

NF3: [-1, -0.5350227355957031, -0.2469314038753510, 0, 0.1833375245332718, 0.3819939494132996, 0.6229856610298157, 1]

[5] https://github.com/TimDettmers/bitsandbytes

Algorithm 3 Quantization methods from FP16 weight groups to **INT4-asym** ones

input : Wgrp_16bit
output : Wgrp_4bit
constant : range # The range of INT4 formats
W_max = max(Wgrp_16bit)
W_min = min(Wgrp_16bit)
$scale = \frac{W\_max - W\_min}{range}$
$zeropoint = [\frac{-W\_min}{scale}]$
// Wgrp_4bit is an empty tensor with the same shape of Wgrp_16bit
for *index in shape(Wgrp_16bit)* **do**
  $W\_tmp = \frac{Wgrp\_16bit[index]}{scale} + zeropoint$
  // Round W_tmp to the nearest INT4 values
  Wgrp_4bit[index] = Round(W_tmp, 'INT4')
end
return *Wgrp_4bit*

Algorithm 4 Quantization methods from FP16 weight groups to **FP4-sym** ones

input : Wgrp_16bit
output : Wgrp_4bit
constant : range # The range of FP4 formats
W_max = max(Wgrp_16bit)
W_min = min(Wgrp_16bit)
$scale = \frac{max(W\_max, |W\_min|)}{range/2}$
// Wgrp_4bit is an empty tensor with the same shape of Wgrp_16bit
for *index in shape(Wgrp_16bit)* **do**
  $W\_tmp = \frac{Wgrp\_16bit[index]}{scale}$
  // Round W_tmp to the nearest FP4 values
  Wgrp_4bit[index] = Round(W_tmp, 'FP4')
end
return *Wgrp_4bit*

C Detailed quantization methods

In this section, we present the pseudocode of other quantization methods used in our experiments, including INT4-asym, FP4-sym, and FP4-asym (one scale and one zero-point). Detailedly, INT4-asym is shown in Algorithm 3, FP4-sym in Algorithm 4, and FP4-asym (one scale and one zero-point) in Algorithm 5. As 3-bit quantization methods are similar to 4-bit ones, we won't go into further detail in this section.

D Results of AFPQ with FP formats

Additional results of RTN quantization using FP4/3 formats are shown in Table 7 and Table 8, respectively. We use 'g-1' to represent per-channel quantization in this section.

Table 7: WikiText-2 perplexity and MMLU average accuracy on LLaMA2 models after FP4 RTN quantization

		LLaMA2-7B				LLaMA2-13B				LLaMA2-70B			
		g-1	g256	g128	g64	g-1	g256	g128	g64	g-1	g256	g128	g64
WikiText-2 ↓	FP16	5.47				4.88				3.32			
	INT4	6.12	5.75	5.72	5.67	5.20	5.02	4.98	4.97	3.67	3.49	3.46	3.44
	FP4-sym	5.89	5.73	5.70	5.67	5.11	5.03	5.02	5.01	3.54	3.47	3.46	3.44
	FP4-asym	5.82	5.71	5.70	5.67	5.09	5.02	5.01	4.99	3.52	3.47	3.45	3.43
MMLU(%) ↑	FP16	46.58				55.38				69.58			
	INT4	40.31	43.67	45.28	45.59	52.92	54.09	54.33	54.44	67.82	68.43	68.32	68.53
	FP4-sym	44.14	44.25	43.74	44.04	53.77	54.17	54.83	54.62	68.14	68.72	68.71	68.90
	FP4-asym	45.25	44.61	45.15	44.55	54.23	54.47	54.70	54.99	68.74	68.65	68.86	69.06

Table 8: WikiText-2 perplexity and MMLU average accuracy on LLaMA2 models after FP3 RTN quantization

		LLaMA2-7B				LLaMA2-13B				LLaMA2-70B			
		g-1	g256	g128	g64	g-1	g256	g128	g64	g-1	g256	g128	g64
WikiText-2 ↓	FP16	5.47				4.88				3.32			
	INT3	542.80	7.10	6.66	6.40	10.68	5.67	5.52	5.39	7.53	4.11	3.98	3.85
	FP3-sym	1621.90	7.16	6.89	6.64	12.76	5.82	5.66	5.54	8.43	4.22	4.11	4.00
	FP3-asym	18.72	6.89	6.63	6.48	7.72	5.69	5.57	5.41	5.93	4.11	4.01	3.89
MMLU(%) ↑	FP16	46.58				55.38				69.58			
	INT3	25.22	37.46	38.50	40.06	27.79	48.91	51.23	50.77	34.39	64.77	65.05	66.16
	FP3-sym	23.73	31.75	36.55	33.08	27.13	48.66	49.76	49.89	32.32	64.65	65.17	65.91
	FP3-asym	27.32	35.42	40.33	40.24	36.15	50.09	50.72	51.60	49.74	64.62	66.14	66.41

Algorithm 5 Quantization methods from FP16 weight groups to **FP4-asym** ones with one scale and one zeropoint

input : Wgrp_16bit
output : Wgrp_4bit
constant : range # The range of FP4 formats
W_max = max(Wgrp_16bit)
W_min = min(Wgrp_16bit)
$scale = \frac{W\_max - W\_min}{range}$
$zeropoint = [\frac{-W\_min}{scale}]$
// Wgrp_4bit is an empty tensor with the same shape of Wgrp_16bit
for *index in shape(Wgrp_16bit)* **do**
  $W\_tmp = \frac{Wgrp\_16bit[index]}{scale} + zeropoint$
  // Round W_tmp to the nearest FP4 values
  Wgrp_4bit[index] = Round(W_tmp, 'FP4')
end
return *Wgrp_4bit*

E More details about system implementation

Currently, W-only quantization requires low-bit weights to be dequantized to FP16 during inference, and then calculations are performed with the FP16 activations. In our system implementation, we store two 4-bit quantized weights using one byte. During de-quantization, we load the byte and recover it to two 4-bit weights.

For INT formats, the de-quantization from 4-bit to FP16 values can be completed by algebraic computation. For FP/NP formats, we realize the de-quantization process by using look-up tables (LUTs).

Algorithm 6 Naive de-quantization process of weight groups from FP4-asym to FP16 ones

input : Wgrp_4bit, scale_pos, scale_neg
output : Wgrp_16bit
constant : FP4toFP16_lut # Standard LUT of mapping FP4 values to IEEE FP16 ones
# Mapping Wgrp_4bit to Wgrp_16bit using FP4toFP16_lut
Wgrp_16bit = [FP4toFP16_lut[w_4bit] **for** w_4bit **in** Wgrp_4bit]
for *w_16bit* ***in*** *Wgrp_16bit* **do**
  # Multiplying w_16bit with scale_pos or scale_neg depending on the sign of w_16bit
  w_16bit = w_16bit * scale_pos **if** w_16bit > 0 **else** w_16bit * scale_neg
end
return Wgrp_16bit

In the main text, we say the optimized de-quantization process can be faster than the naive version by reducing the number of sign judgements and scale multiplications, but we only shows the optimized de-quantization process because of text length limitation. Here we show the naive version in Algorithm 6, taking FP4 as an example. In this version, each quantized 4-bit weight is converted to an FP16 value using look-up tables (LUT), referred to FP4toFP16_lut. The number of look-up values is 16 (2^4) for 4-bit quantization and 8 (2^3) for 3-bit quantization. These values are then multiplied by either scale_pos or scale_neg based on the weight's sign. This process results in a num-

ber of sign judgments during inference equal to the number of weights, leading to system overhead. For example, in the naive NF4-asym system with group-size 128, the number of sign judgements and scale multiplications is 128 because each weight should be multiplied with the positive scale or negative scale depending on its sign.