Towards Efficient Pre-training: Exploring FP4 Precision in Large Language Models

Anonymous ACL submission

Abstract

The burgeoning computational demands for training large language models (LLMs) necessitate efficient methods, including quantized training, which leverages low-bit arithmetic operations to reduce costs. While FP8 precision has shown potential, leveraging FP4 remains challenging due to inherent quantization errors and limited representation capability. Based on the Transformer architecture, we present an FP4 training scheme for LLMs, overcoming these obstacles through mixed-precision quantization strategies tailed for different modules and training stages. This allows us to apply the precision level suitable to distinct components within the model, ensuring that multi-head attention and linear layers are handled appropriately. Our pretraining recipe ensures stability in backpropagation by incorporating fine-grained quantization methods with a target precision training schedule. Experimental results demonstrate that our FP4 training scheme achieves accuracy comparable to BF16 and FP8, with smaller theoretical computational cost. With the advent of next-generation hardware supporting FP4, our method sets the foundation for efficient ultra-low precision training.

1 Introduction

Recent advancements in large language models, including GPT (Radford, 2018; Floridi and Chiriatti, 2020; Achiam et al., 2023), DeepSeek (Liu et al., 2024), Llama (Touvron et al., 2023) and OPT (Zhang et al., 2023), have demonstrated strong generalization capabilities across various tasks (Stiennon et al., 2020; Alexey, 2020). Among these advancements, pretraining on large-scale unlabeled data has proven to be critical for ensuring model performance (Radford, 2018; Dong et al., 2019). Increasing the model size and the dataset scale can enhance performance (Kaplan et al., 2020; Hoffmann et al., 2022), but this improvement comes with significant computational costs. To address this, numerous methods have been proposed to accelerate the pretraining process (Duan et al., 2024).Particularly, low-precision computation serves as an efficient acceleration technique. This approach quantizes the inputs of computationally intensive operators to a specified low-bit width, leveraging low-bit arithmetic units to speed up training.

Previous research on low-precision training has primarily focused on deep learning models. However, these methods do not fully consider the characteristics of large language model pretraining, which has unique training methods and model architectures. The good news is that next-generation hardware will support FP4 and FP8 format (Nvidia). Studies like (Peng et al., 2023; Micikevicius et al., 2022a; NVIDIA; Fishman et al., 2024; Xi et al.) have demonstrated the capability of 8-bit computation for LLM pretraining. However, the application of FP4 tensor cores in LLM pretraining remains unexplored. Compared to INT4, FP4 offers a larger numerical representation space, making it possible to further reduce the bit width in large-scale model pretraining. However, the limited number of bits in FP4 format introduces significant quantization errors, making the application of FP4 to pretraining highly challenging.

From the perspective of LLM structure, it has been observed that different modules and computational components exhibit varying levels of sensitivity. Given the critical role of the multi-head attention (MHA) mechanism in the Transformer architecture, it is imperative to implement specific strategies to ensure the accuracy of attention modules. As the volume of data continues to grow, the gradient values tend to decrease, making FP4 quantization more prone to underflow, thus hindering parameter updates. Based on these observations, we propose a novel FP4 mixed-precision large language model pretraining recipe. Specifically, we leverage a per-block quantization strategy and employ different quantization approaches across mod-

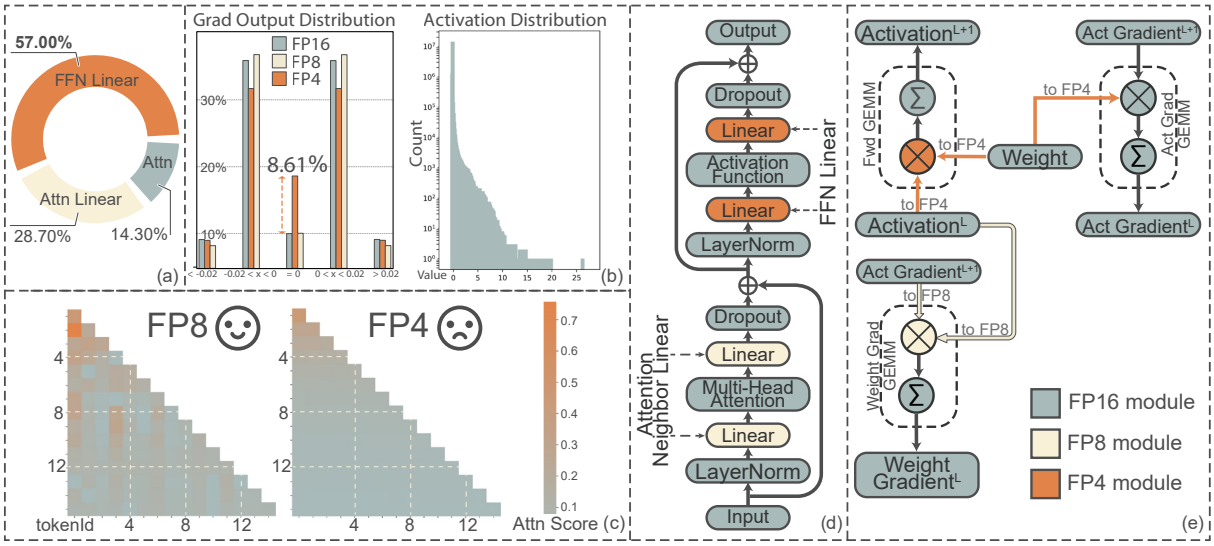

Figure 1: (a) shows the proportion of computational overhead for the main computation components of a transformer block when using the LLaMA 7B configuration with a sequence length of 4K. (b) shows the distribution of activations and gradients after the GPT-large model has been trained to approximately 10B tokens. (c) shows the heatmap of attention scores when using different training strategies. (d) and (e) illustrate our training scheme, which will be detailed in Section 3.

ules and training stages to enable FP4 model training. The approaches enable better exploitation of the computational improvements brought by future hardware advancements.

In this paper, we explored the use of FP4 precision in language model pretraining and, for the very first time, proposed an effective mixed precious pretraining strategy. First, considering the distinct requirements of different modules, we applied tailored quantization strategies to preserve the precision of MHA execution. Second, as backpropagation has shown high sensitivity to precision, we employed finer-grained quantization methods to ensure accurate parameter updates during the backward pass. Finally, we adopted a 2-stage target precision training schedule to eliminate the impact of quantization noise on the model.

2 Related Work

Low-precision training enhances deep learning efficiency by reducing computational costs. Many existing studies focus on the training of deep neural networks (DNNs) (Wang et al., 2018; Chmiel et al., 2023; Sun et al., 2019; Xi et al., 2023; Fu et al., 2021), whose architecture and performance differ from LLM pre-training. In the context of low-precision training for large model pre-training, some progress has been made in FP8. For example, (Micikevicius et al., 2022a) introduced new FP8 floating-point formats (E4M3 and E5M2), and (Fishman et al., 2024) extends FP8 to trillion-token large-scale model pretraining. In terms of FP4 training, (Wang et al., 2025) improved FP4 computational precision using a differentiable quantization estimator and outlier clamping and compensation strategy. However, most existing methods fail to fully account for the varying sensitivity to precision across different model modules.

3 Methods

Our objective is to maximize the efficiency of low-precision computations based on the characteristics of LLMs. As shown in Fig.1(a), the computational cost of three key components in a transformer model is analyzed, with FFN accounting for 57%. Considering both the computational cost and impact on performance, we meticulously design three corresponding training schemes: 3.1 Attention-protected Neighbor Linear, 3.2 Gradient-sensitive Linear, and 3.3 Target Precious Training Schedule. These schemes fully leverage hardware acceleration while keeping precision loss within an acceptable range during training.

3.1 Attention-protected Neighbor Linear

As the core component of the Transformer model, the attention mechanism is highly sensitive to precision. Quantization errors introduced by low-

precision training can accumulate over time, eventually disrupting the function of the attention mechanism. As shown in Fig. 1(c), an undisturbed attention mechanism identifies tokens 0, 3, 6, and 9 as more important. However, under FP4 training, the result becomes nearly uniformly distributed, preventing the model from distinguishing which tokens are significant. This makes it difficult for the model to differentiate the importance of tokens, thereby affecting the convergence speed.

To ensure the proper functioning of the attention mechanism and enable the model to correctly evaluate the importance of each token, we employ FP8 precision for the computation of QKV and the output projection to "protect" the accurate execution of the attention mechanism, as shown in Fig. 1(d).

3.2 Gradient-sensitive FFN Linear

Weight gradient computation is more sensitive to errors compared to forward computation, due to the fact that both gradients and activations contribute to the error. For gradients, since many values are around 0.02, especially as training progresses and gradient magnitudes decrease, underflow is likely to occur. As can be seen in the left of Fig.1(b), there is an 8.6% difference between FP4 and FP8/FP16, thus requiring a more accurate representation. For activations, we observe that underflow occurs approximately 18% of the time between FP4 and FP8/FP16. This is largely due to the relatively large range of values, as can be seen in the right of Fig. 1(b). Therefore, a more accurate representation is also needed. Additionally, optimizers use the gradients to update model parameters. Based on the above discussion, for the weight gradient computation of model weights, we adopt FP8 precious computation, as shown in the bottom left corner of Fig.1(e).

Furthermore, for the activation gradient computation (the top right corner of Fig.1(e)), we find that quantizing gradients significantly impacts the convergence of model training. There is always a nonlinear operation between the linear layers, which requires more precise numerical representations. Furthermore, quantization errors accumulate iteratively through the chain rule during backpropagation, ultimately hindering the convergence of model training.

Lastly, in our experiments, we observed that quantization noise increases as the model size and the amount of data grow (a detailed explanation can be found in Appendix B). This occurs because, when the model reaches a certain level of accuracy, coarse-grained low-precision tensors can no longer currently represent the parameter space and input information. Therefore, we adopt a more conservative quantization approach to maintain stable training in forward computation. As shown in the top left corner of Fig.1(e). To ensure efficient hardware implementation, we use per-block quantization strategies where the block size is set to 128.

3.3 Target Precious Training Schedule

When using low-precision training throughout the entire process, there tends to be a performance gap between the low-precision model and the FP16 model, as shown in Fig.2. The validation loss curves exhibit a parallel trend. Although the gap between the two curves is very small, the difference in downstream tasks, such as wikitext perplexity (PPL), can be more pronounced, reaching up to approximately 6.3 compared to the model trained with FP16. This is likely due to compromises the model makes to adapt to the noise introduced by quantization during low-precision training. To address this issue, we employ a Target Precious Training Schedule, which involves two stages: continuing the FP4 pretraining process with FP16 for a short period. This accounts for only 5% to 10% of the total training steps, allowing the model to return to an ideal state.

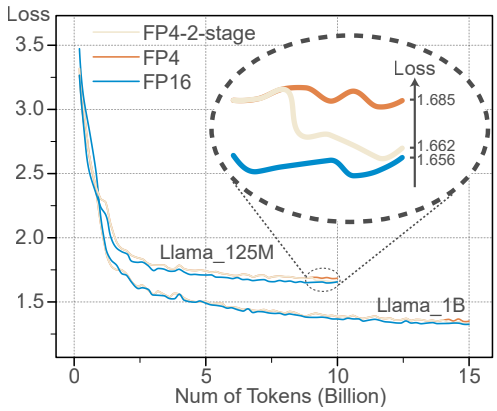

Figure 2: Loss curve for the Target Precious Training Schedule.

4 Experiment

In this section, we evaluate the proposed FP4 training method across language models of various sizes. The detailed model training configurations and hyperparameter settings are provided in Appendix B. Section 4.1 presents the main results, showcasing the model's performance on downstream tasks.

Table 1: Comparison of FP4 and FP16 Training Results

Model	Method	Val Loss	Val PPL	Text Gen	Natural Language Understanding (GLUE)								
				WikiText	cola	sst2	mrpc	stsb	rte	wnli	qnli	mnli	qqp
GPT2 125M	Ours	1.706	5.507	50.98	0.2663	0.8704	0.7549 / 0.8322 / 0.7936	0.7681 / 0.7658	0.5704	0.3380	0.8473	0.7554 / 0.7705	0.8777 / 0.8403
	FP16-baseline	1.705	5.503	50.14	0.2290	0.8796	0.7647 / 0.8395 / 0.8021	0.7798 / 0.7808	0.5884	0.3099	0.8548	0.7613 / 0.7695	0.8799 / 0.8437
GPT2 335M	Ours	1.549	4.705	37.62	0.2565	0.8899	0.7647 / 0.8362 / 0.8004	0.8159 / 0.8122	0.6029	0.2535	0.8611	0.7798 / 0.7874	0.892 / 0.8572
	FP16-baseline	1.556	4.739	38.39	0.3002	0.8819	0.7745 / 0.8419 / 0.8082	0.8266 / 0.8298	0.6209	0.1831	0.8726	0.7799 / 0.7889	0.8929 / 0.8508
GPT2 774M	Ours	1.431	4.181	30.01	0.3473	0.9002	0.7745 / 0.8472 / 0.8108	0.8305 / 0.8336	0.6498	0.2676	0.8742	0.7995 / 0.808	0.8984 / 0.8656
	FP16-baseline	1.430	4.178	28.36	0.3708	0.8922	0.7794 / 0.8454 / 0.8124	0.8347 / 0.8353	0.6498	0.2254	0.8911	0.8078 / 0.813	0.9012 / 0.8683

Section 4.2 presents the ablation study to demonstrate the effectiveness of our training method.

4.1 Main Result

We validate the proposed FP4 pretraining method on two large language models, using the widely adopted GPT-2 and LLaMA architectures. The GPT-2 and LLaMA models are pretrained on the RedPajama-WikiText (Weber et al., 2025) dataset within the Megatron framework and evaluate their text generation capabilities on wikiText(Merity et al., 2016). Additionally, we assess their natural language understanding abilities on the GLUE(Wang, 2018) benchmark.

We train approximately 10B tokens on GPT-2-small and GPT-2-mid and around 25B tokens on GPT-2-large. The final validation loss and validation perplexity (PPL) are presented in Table 1, showing that the pretraining results obtained with our method exhibit almost no performance difference compared to models trained using FP16. In addition to training loss, the downstream task performance of the same pretrained models demonstrates that the average accuracy of FP4-trained models is comparable to that of FP16-trained models.

4.2 Ablation Study

We aim to investigate the effect of the module-wise pretraining method introduced in Section 3. For this ablation study, we train the LLaMA2-125M model on approximately 5B tokens. The results in the table indicate that different modules exhibit varying levels of robustness to low precision. Additionally, we compute the theoretical computation cost for these methods and observe that our approach achieves a lower theoretical computation cost (see Appendix B for details) while maintaining higher performance.

Attention Linear	FFN Linear	FP4 Linear' Backward	Training loss	Val loss	Val ppl	Computation cost
FP4	FP4	FP4	2.2659	1.7828	5.9467	57.1%
FP4	FP8	FP8	2.2211	1.7543	5.7798	69.6%
FP8	FP4	FP4	2.2562	1.7549	5.7831	60.7%
FP8	**FP4**	**FP8**	**2.2225**	**1.7415**	**5.7062**	**66.1%**
FP16	FP16	FP16	2.1998	1.7097	**5.5273**	100%

Table 2: Ablation studies about different precious on different modules

	Attention	FFN	FFN Backward	Target Precious	Val loss	Val ppl	Computation Cost
Llama 1B	FP8	FP4	FP8	no	1.3505	3.8596	67.5%
	FP8	FP4	FP8	yes	1.3311	**3.7855**	69.7%
	FP16	FP16	FP16	-	1.3296	**3.7797**	100%
Llama 125M	FP8	FP4	FP8	no	1.6851	5.3933	68.2%
	FP8	FP4	FP8	yes	1.6622	**5.2670**	71.4%
	FP16	FP16	FP16	-	1.6567	**5.2424**	100%

Table 3: Ablation studies about target precious training schedule

Furthermore, to demonstrate the effectiveness of the 2-stage training schedule, we conducted the following ablation experiment, using the same experimental setup as in Section 4.1. The results in the table highlight the importance of the 2-stage training schedule for large-scale model pretraining.

5 Conclusion

We propose an FP4 pre-training scheme for modern large language models. Based on the sensitivity analysis of computational modules and training stages, we adopt a tailored training recipe according to the position of linear modules, together with a Target Precious Training schedule, to ensure stable convergence. Experimental results show that FP4-based training achieves comparable validation loss and downstream task accuracy to traditional FP16 training while reducing computational costs by 30%. Additionally, our ablation studies confirm the importance of adaptive quantization strategies across different model modules and training stages, providing new insights for the further development of low-precision training techniques and efficient training of large language models on next-generation hardware.

6 Limitation

First, due to computational resource limitations, our method has not been validated on larger models and larger datasets to demonstrate its effectiveness. Investigating such scalability remains a critical direction for future research. In addition, since the model adopts a simulated FP4 approach, it is unable to obtain an accurate increase in training efficiency. Lastly, for the sensitive computational components, we employed a simple strategy to ensure numerical precision. In future work, we will explore more customized approaches to enable a broader range of computations to utilize FP4.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Dosovitskiy Alexey. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929*.

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

Brian Chmiel, Ron Banner, Elad Hoffer, Hilla Ben-Yaacov, and Daniel Soudry. 2023. Accurate neural training with 4-bit matrix multiplications at standard formats. In *The Eleventh International Conference on Learning Representations*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *Advances in neural information processing systems*, 32.

Jiangfei Duan, Shuo Zhang, Zerui Wang, Lijuan Jiang, Wenwen Qu, Qinghao Hu, Guoteng Wang, Qizhen Weng, Hang Yan, Xingcheng Zhang, et al. 2024. Efficient training of large language models on distributed infrastructures: A survey. *arXiv preprint arXiv:2407.20018*.

Maxim Fishman, Brian Chmiel, Ron Banner, and Daniel Soudry. 2024. Scaling fp8 training to trillion-token llms. *arXiv preprint arXiv:2409.12517*.

Luciano Floridi and Massimo Chiriatti. 2020. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694.

Yonggan Fu, Han Guo, Meng Li, Xin Yang, Yining Ding, Vikas Chandra, and Yingyan Lin. 2021. Cpt: Efficient deep neural network training via cyclic precision. *arXiv preprint arXiv:2101.09868*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. arxiv. *arXiv preprint arXiv:2203.15556*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Tanishq Kumar, Zachary Ankner, Benjamin F Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. 2024. Scaling laws for precision. *arXiv preprint arXiv:2411.04330*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. 2023. Llm-fp4: 4-bit floating-point quantized transformers. *arXiv preprint arXiv:2310.16836*.

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

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. 2022a. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. 2022b. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*.

Nvidia. Nvidia blackwell architecture technical brief. https://resources.nvidia.com/en-us-blackwell-architecture.

NVIDIA. Transformer engine. https://github.com/NVIDIA/TransformerEngine.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. 2023. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*.

Alec Radford. 2018. Improving language understanding by generative pre-training.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. 2019. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. *Advances in neural information processing systems*, 32.

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*.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31.

Ruizhe Wang, Yeyun Gong, Xiao Liu, Guoshuai Zhao, Ziyue Yang, Baining Guo, Zhengjun Zha, and Peng Cheng. 2025. Optimizing large language model training using fp4 quantization. *arXiv preprint arXiv:2501.17116*.

Maurice Weber, Dan Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, et al. 2025. Redpajama: an open dataset for training large language models. *Advances in Neural Information Processing Systems*, 37:116462–116492.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Haocheng Xi, Yuxiang Chen, Kang Zhao, KAI JUN TEH, Jianfei Chen, and Jun Zhu. Jetfire: Efficient and accurate transformer pretraining with int8 data flow and per-block quantization. In *Forty-first International Conference on Machine Learning*.

Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. 2023. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36:49146–49168.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2023. Opt: Open pre-trained transformer language models, 2022. *URL https://arxiv. org/abs/2205.01068*, 3:19–0.

A FP4 Quantization

Quantization is the process of converting a data type with more bits (e.g., 32- or 16-bit floating points) into another data type with fewer bits (e.g., 4-bit floating points). In integer quantization, the real-valued variable X_R is quantized to an integer X_{INT} with the following formula:

$$X_{\text{INT}} = \alpha \left\lfloor \text{Clip}\left(\frac{X_R}{\alpha}, Q_{\text{min}}, Q_{\text{max}}\right)\right\rceil \quad (1)$$

Similar to integer quantization, in float point quantization, scaling and clipping of the values are required before quantization, as follows.

$$Q_{\text{max}} = -Q_{\text{min}} = (2 - 2^{-m})2^{2^e - b - 1} \quad (2)$$

$$\tilde{Q}_{\text{max}} = \alpha Q_{\text{max}} \quad (3)$$

$$X'_R = \text{Clip}\left(X_R, \tilde{Q}_{\text{min}}, \tilde{Q}_{\text{max}}\right) \quad (4)$$

Where the min/max value range of signed floating-point quantization can be calculated from Eq. (2), and the scaling factor α determines the quantization granularity. Thus, we can determine the upper and lower bounds of floating-point quantization and perform the clip operation accordingly.

After scaling and clipping, we can quantize the real value into a specific data format. Unlike INT quantization, floating-point numbers have different quantization levels for different values. Therefore, we first need to determine the quantization step size:

$$\tilde{\alpha} = 2^{-\tilde{b}} = 2^{-b} \cdot \alpha \quad (5)$$

$$v = \begin{cases} 2^{\lfloor \log_2 |X'_R|/\tilde{\alpha} \rfloor - m} & \text{if } \lfloor \log_2 |X'_R|/\tilde{\alpha} \rfloor \geq 1 \\ 2^{1-m} & \text{otherwise} \end{cases} \quad (6)$$

Here, the quantization level v is determined by $x'_R/\tilde{\alpha}$, after which the floating-point number can be quantized following the format of Eq. (8). A detailed explanation can be found in (Micikevicius et al., 2022b; Liu et al., 2023). Finally, the quantization formula can be expressed as follows:

$$X_{\text{FP}} = \tilde{\alpha} \cdot v \cdot \left\lfloor \frac{X'_R}{\tilde{\alpha} \cdot v} \right\rceil \quad (7)$$

B Training Detail

We conducted our experiments on the Megatron (Shoeybi et al., 2019) framework and conducted downstream evaluations using the transformers (Wolf et al., 2020) and lm-evaluation-harness (Gao et al., 2024), ensuring standardized and reproducible benchmarking. Hyperparameters remain consistent across precision settings for fair comparison. The learning rate follows a warm-up and cosine decay schedule, with the warm-up phase spanning 0.15% of total steps and the learning rate gradually decreasing to 10% of its peak over the remaining 90%. The peak learning rate is 1×10^{-4}, with a weight decay of 0.1 for Llama models. For GPT models, the peak learning rate is set to 6×10^{-4}, with a weight decay of 0.1 For the Adam optimizer, we use $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 1 \times 10^{-8}$. For llama models, the input sequences are fixed at 2048 tokens, and the batch size is 512, comprising approximately 1M tokens. For GPT models, the input sequences are fixed at 1024 tokens, and the batch size is 480, comprising approximately 0.5M tokens, detail model config can be found in Table 4.

Parameter	GPT-125M	GPT-335M	GPT-774M	LLaMA-125M	LLaMA-1B
Layers	12	24	36	12	48
Hidden Size	768	1024	1280	768	1280
Activation Function	GELU	GELU	GELU	SwiGLU	SwiGLU
Normalization	LayerNorm	LayerNorm	LayerNorm	RMSNorm	RMSNorm
FFN Hidden Size	3072	4096	5120	3072	3392
Sequence Length	1024	1024	1024	2048	2048
Attention Heads	12	16	20	12	20

Table 4: GPT and LLaMA Model Configurations

We quantize all of the linear layers in the MLP and attention module to target precious, and leave multi-head attention and activation function in FP16 by employing FlashAttention(Dao et al., 2022). The master copy of the weights is kept in FP32. We quantize linear layers' inputs to target precious prior to each matmul, but leave layernorm's weight and bias to floating-point since they are relatively small.

For the calculation of theoretical computation cost, we first separately count the forward and backward computation amounts for each part of a Transformer block (considering only computations related to matrix multiplications, as these account for more than 95% of the total computation). Then, based on the assumptions that FP8 achieves twice the computation speed of FP16 and FP4 achieves four times the computation speed of FP16, we compute the theoretical time required for each matrix multiplication. Finally, we obtain the theoretical computation cost for each method.

Since the quantized weight $\tilde{w}$ is an estimate of w, We directly use a straight-through estimator(Bengio et al., 2013) directly passes the gradient

of $\tilde{w}$ to w:

$$\nabla_{\mathbf{w}}\mathcal{L}(\tilde{\mathbf{w}}) \leftarrow \nabla_{\tilde{\mathbf{w}}}\mathcal{L}(\tilde{\mathbf{w}}).$$

In our experiments, we found that different models have varying precision requirements. For the GPT-125M model, applying a per-token and per-channel FP4 quantization strategy for both forward computation and weight gradient computation is feasible. The final results are shown in Table 1. The use of per-token and per-channel quantization is designed to better align with matrix multiplication rules, allowing for efficient implementation on accelerators. However, for the GPT-335M model, the per-token and per-channel FP4 quantization strategy becomes ineffective as the data volume increases. In this case, switching to per-block FP4 quantization for weight gradient computation enables training to proceed, with the final results also presented in Table 1. For the GPT-770M model, the quantization strategy used for GPT-335M becomes ineffective as training progresses. At this point, modifying the forward computation to use per-block FP4 quantization while increasing the precision of weight gradient computation to FP8 ensures stable training. Additionally, we validated the feasibility of the GPT-770M quantization strategy(As we discuss in Section 3) on the LLaMA-125M and LLaMA-1B models. It can be anticipated that as model size and data volume continue to grow, the precision requirements for model training will become increasingly stringent(Kumar et al., 2024). Ensuring that FP4 can support long-term, large-scale pretraining of large models remains a key direction for our future work.