

Quantized Pre-training for Small Mixture-of-Experts

Stanford CS224N Custom Project

Pranith Koppula

Department of Computer Science
Stanford University
pranith@stanford.edu

Abstract

Pretraining takes up the majority of compute and training cost in the development of modern large language models (LLMs). The paper studies the effects of quantized pre-training for mixture-of-experts (MoE) models. This paper looks foremost at the cutting-edge of low-bit quantization and pretraining for MoE models. We perform various experiments with different model precisions and features, and seek to replicate the methodology present in NVFP4 paper. We propose an analysis of quantization techniques, loss patterns, and showcase a quantized W4A16 MoE model performing close to the baseline BF16 model.

1 Key Information to include

- Mentor: Julie Kallini
- External Collaborators (if you have any): N/A
- Sharing project: N/A
- Late Days (if you are using any):

Team member	Late days remaining
<Name 1>	Pranith Koppula
<Name 2>	<>
<Name 3>	<>
Total pooled	2
Team size	1
Late days applied for the report	2

2 Introduction

Large language models (LLMs) have continued to grow larger and more complex, and this has increased the compute required to achieve these goals. Recent advancements in low precision or fully quantized training (FQT) have shown that promising results that push the boundary of extremely efficient training.

Mixture-of-Experts (MoEs) have emerged as frontier architecture for LLMs, scaling to trillions of parameters [1, 2, 3]. These models at an abstracted level are sparse large neural networks where subset of the networks are activated based on the tokens received.

The current literature focuses on FP4 quantization on transformer architectures with dense networks. There are not much studies on how quantized training can effect sparse networks such as MoEs and work is focused on post training quantization and tuning [4, 5, 6] Low-precision quantization can unlock significant compute efficiency for MoEs, especially as MoEs are scaling to the trillions of parameters.

This work showcases a functional training method that serves as the first look into training fully quantized MoE models. We showcase that router sensitivity is extremely important in MoE models, and quantizing router weights leads to poor performance. MoE models also are attuned to parameter outliers and removing activation gradients and noise also leads to worse performance. We find that randomized Hadamard transform (RHT) potentially leads to more instability in MoE models. Stochastic rounding (SR) and keeping the last projection layer in high precision lead to much higher stabilized training. Finally we discuss the tradeoffs between weight, activation, and gradient quantization, the importance of the router to MoE models and future work to robustly scale and push the boundary for low-precision training.

3 Related Work

Recent work by NVIDIA showcased large scale pre-training using the NVFP4 format [7]. They used fully quantized training (FQT) to train a hybrid Mamba-Transformer model with 12 billion parameters on 10 trillion tokens with the NVFP4 format. Using Random Hadamard transforms (RHT) to bound block-level outliers, two-dimensional quantization, stochastic rounding (SR), and selected high-precision layers they found that the model achieved accuracies and performance equal to an FP8 baseline. The work pushed the boundary of low-precision training, and focused on training stability and convergence for large scale selective-state space architectures. This work focuses on smaller MoE models, however we apply some of the methodology used to analyze the similarities and differences of quantization. They use NVFP4 which is a custom extended microscaling format introduced in the Blackwell architecture [8]. We focus on FP4 and standard data types available in general GPU architectures.

BitNet [9] introduced the most extreme version of quantization aware training (QAT) by limiting the weight gradients to 1 bit and keeping the activation parameters to 8 bits. They find that 1-bit training has significant compute savings through reductions in memory bandwidth and energy consumption. They stabilize training through straight-through estimator (STE), mixed precision training by keeping gradients and activations in high-precision, and using large learning rates to update latent weights when precision is so small with 1-bit weights. However, this methodology is not well supported in hardware architectures due to the custom ternary math operations and thus is hard to scale to large scale models on real hardware.

More works have focused on the boundary from FP8 training to FP4 [10, 11]. The FP4 All the Way paper uses FP4 for the weights, gradients, and activations and introduce a novel theoretical threshold for the effectiveness of quantized training. They find that when the gradient norm reaches $\sqrt{3}$ times the quantization noise training becomes less effective. They use stochastic rounding (SR) in the backward pass with round-to-nearest in the forward pass to improve stability. [11] use MXFP4 to train GPT 345M, 1.3B and 6.7B models. They find that using MXFP4 GEMMs with simulations and using SR and RHT to stabilize training produces models that are near-lossless.

Many of the works showcase similar themes of using standardized techniques to soften quantization noise and precision loss. Thus we apply these techniques to a Sparse MoE model below.

4 Approach

4.1 Architecture

We train a small transformer decoder model [12] and replace the feed-forward networks (FFN) with Sparse MoE blocks [1]. We use flash attention [13] for the scaled dot product speedup.

The MoE blocks are sparse FFN connected through a router mechanism. We first compute an expert score h for each expert i given input token x :

$$H_i(x) = (x \cdot W_g)_i + \text{Softplus}((x \cdot W_{\text{noise}})_i) \quad (1)$$

For our quantized version we simply apply a quantization strategy to the weight matrices:

$$H_i(x) = (x \cdot \text{quantize}(W_g))_i + \text{Softplus}((x \cdot \text{quantize}(W_{\text{noise}}))_i) \quad (2)$$

Then we get the top k router indices that is used to do the actual forward pass for each token.

$$G(x) = \text{TopK}(\text{Softmax}(H(x))) \quad (3)$$

Because this routing is dynamic, each expert get dynamic input sizes (for example expert A can get 20 tokens, B get 10 tokens). To solve this problem we use **expert capacity** [1].

$$\text{Expert Capacity} = \frac{K \cdot B \cdot C}{N} \cdot \text{Capacity Factor} \quad (4)$$

where K is the number of active experts, $B \cdot C$ is the batch size times sequence length representing the total independent tokens, and N is the total number of experts. The capacity factor is a hyperparameter that dictates the routing uniformity with a value of one representing uniform routing, and greater than one provides extra buffer to handle token imbalance.

Router Load Balancing Initially the distribution of most of the routers weights did not change from mean zero gaussian. This indicated router collapse, and that the model did not learn to balance the various routers. Thus we can use load balancing loss [14].

$$\begin{aligned} \mathcal{L}_{\text{Balance}} &= \alpha \sum_{i=1}^N f_i P_i, \\ f_i &= \frac{N}{KT} \sum_{t=1}^T \mathbb{1}(\text{Token } t \text{ selects Expert } i), \\ P_i &= \frac{1}{T} \sum_{t=1}^T s_{i,t}, \end{aligned} \quad (2)$$

Stride We do not replace every FFN block with an MoE block. We use stride to interleave a standard FFN network with a Sparse MoE network.

Loss function The loss function comprises of cross entropy loss regularized with load balancing loss with α parameter acting as the weight / amount of loss to apply.

4.2 Quantization

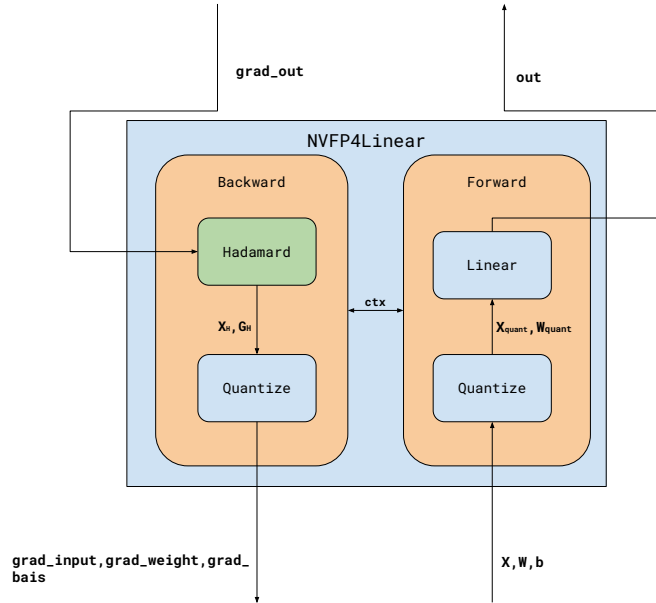


Figure 1: NVFP4 Linear Layer with Hadamard Transform. SR is present inside the Quantize utility and both the quantization and de-quantization happen before the outputs are transferred.

In modern LLM training, quantization involves converting parameters of model into low-precision numbers, performing highly optimized GEMMs that are built through fused kernels, and then de-quantizing the output of the GEMM and passing this along to either the forward or backward passes.

Quantization is the method of converting high-precision number types to low-precision representations.

$$\mathbf{S} = \frac{\max(|\mathbf{X}|)}{6.0} \quad (5)$$

$$\mathbf{X}_{\text{quant}} = \text{clamp}(\text{round}(\mathbf{S} \cdot \mathbf{X})) \quad (6)$$

$$\mathbf{X}_{\text{dequant}} = \mathbf{S}\mathbf{X}_{\text{quant}} \quad (7)$$

We use 6.0 for the scaling factor as its the maximum value to represent the E2M1 FP4 representation.

The process of rounding and clamping introduce significant precision errors. Outliers present in the gradients and activations are not preserved, which causes poor model performance.

Random Hadamard Transforms (RHT) To preserve outliers we use Random Hadamard Transforms. Hadamard matrix is a special type of orthogonal matrices that satisfy the requirement

$$\mathbf{H}_0 = [1] \quad (8)$$

$$\mathbf{H}_k = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_{k-1} & \mathbf{H}_{k-1} \\ \mathbf{H}_{k-1} & -\mathbf{H}_{k-1} \end{bmatrix} \quad (9)$$

And the matrix satisfies the constraint that

$$\mathbf{H}_k \mathbf{H}_k^\top = \mathbf{I} \quad (10)$$

k is a hyperparameter where a larger value leads to more diffusion of outliers across a linear subspace. This allows us to apply RHT to GEMM operations, and we do so the following way:

$$\mathbf{G} = \frac{\partial L}{\partial Y} \quad (11)$$

$$\begin{aligned} \text{grad_output} &= (\mathbf{H}_k \mathbf{G})^\top \mathbf{H}_k \mathbf{X} \\ &= (\mathbf{G}^\top \mathbf{H}_k^\top)(\mathbf{H}_k \mathbf{X}) \\ &= \mathbf{G}^\top \mathbf{X} \end{aligned}$$

RHT rotates the matrix by multiply by the random diagonal matrix and the Hadamard matrix. This has the effect of smearing high-value outliers uniformly across other elements reducing distortion.

Stochastic Rounding (SR) This is the principle of adding noise to the activation gradients after quantizing / rounding. We apply this technique as nearest-based rounding can lead to bias. We add uniform noise ϵ and floor to match to the closest bin.

$$\epsilon \sim \mathcal{U}(0, 1) \quad (12)$$

$$\text{SR}(x) = \lfloor x \rfloor + \epsilon \quad (13)$$

We replace the `nn.Linear` layer with a custom implementation that performs RHT and SR on the weights and tokens, performs the linear GEMM $\mathbf{W}^\top \mathbf{X}$, and returns the dequantized output. The output projection head and normalization layers were not quantized and kept in original precision (BF16) to preserve numerical stability.

5 Experiments

We run multiple experiment arms testing the various stabilities and model training for the methods mentioned above. We run simulated quantization through quantization aware training (QAT) that quantizes and de-quantizes the inputs on the fly.

5.1 Data

We used the OpenWeb Text dataset for pretraining [15]. The dataset contains 9B tokens, and is a derived from reddit post urls, and out-links. There are about 8M total documents in the dataset. We do not do any processing or filtering. We tokenize the dataset using tiktoken, which is the tokenizer used in GPT-2.

5.2 Evaluation method

We train a baseline 190M parameter model with the same model architecture hyperparameters as the experimental models. We evaluate the models with HellaSwag [16] which tests the models common sense sentence prediction. We also benchmarked the models with the final checkpoint using COPA [17]. This is similar to HellaSwag and checks for basic causal understanding in text and also serves as a test set with unseen text.

We also evaluate training stability and validation loss. We also look into the potential issues with FP4 format and an ablation study to understand the methodology.

5.3 Experimental details

We trained multiple 190M parameter models, each running on 4xH200 GPUs using Distributed Data Parallel training [18]. The baseline model was trained with the architecture mentioned above for 7000 steps, which was also replicated for all experiment arms. See table 3 for the full hyperparameters used.

We implemented a custom `NVFP4Linear` layer that replicated the methodology from [7]. We did try to implement custom triton kernels for the various quantization strategies such as `w4a4` or `w4a16` but found that without fused operations, buffers, and other optimized features our model runs slower and the results performed similarly to the simulated versions. Furthermore, FP4 operations are not supported natively in the Ampere and Hopper architectures, and thus limited what insights we could get without the more modern Blackwell architecture that supports FP4 operations.

The implementation was derived from `nanoMoE` [19] and `nanoGPT` [20]. We contributed by adding quantization utilities and simulations, and modifying some of the existing implementations such as the routers and `SparseMoe` layer to fit our needs.

Below we describe the various experiments we conducted. Not we use the notation `W#A#` to represent the bit representations for the weights (W) and the activations and gradients (A).

5.3.1 FP4 W4A16 for all Linear Layers

This experiment involved quantizing all linear layers weights to FP4. We inherit the `nn.Linear` layer’s parameters, specifically the weights. The weights are then quantized and combined linearly with `X`, the input tokens. This was done in the forward pass only.

The backward pass contained no other logic and simply passed through the `grad_output` from the layer ahead which is simply the STE method.

5.3.2 FP4 W4A16 for all Linear Layers except Router

This experiment involved quantizing all linear layers weights to FP4 except the router. This arm was added as MoE models quantization has been shown to not work as well due to the inherent sparsity and heterogeneity of the architecture [21].

5.3.3 INT4 W4A16 for all Linear Layers except Router

This experiment involved quantizing all linear layers weights to INT4 except the router. This arm was added as a ablation / baseline to see if we get any benefits from floating point. The hypothesis is that INT4 has lower dynamic range and lack of spread due to integer cutoff.

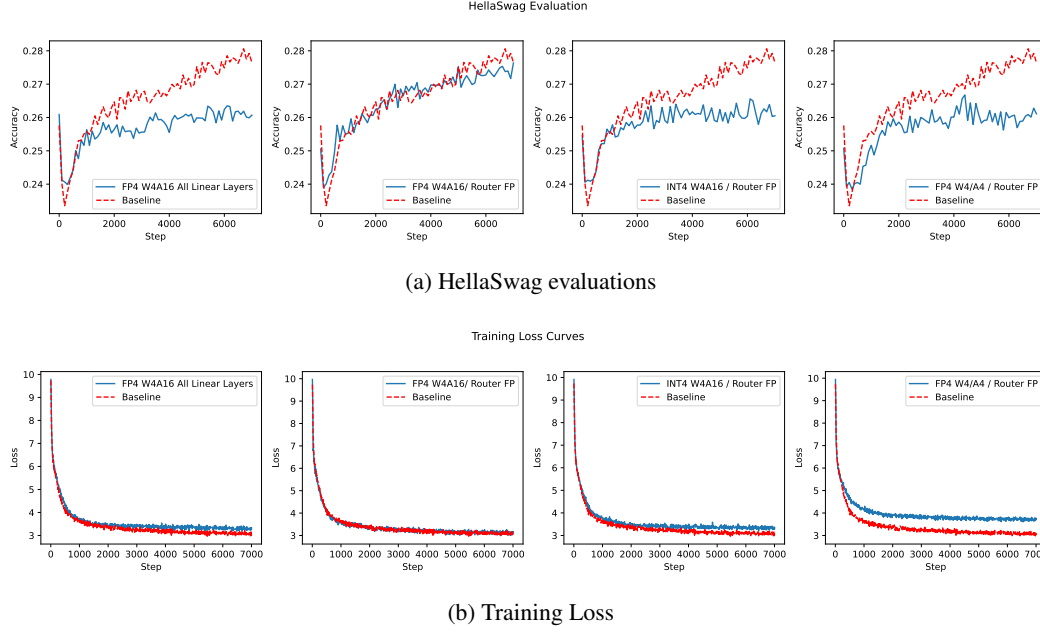


Figure 2: HellaSwag Evaluation and Training Loss

5.3.4 FP4 W4A4 for all Linear Layers except Router

This experiment involved quantizing all linear layers weights to FP4 W4A4 except the router. This arm contained the core methodology from the NVFP4 pre-training paper, which included implementing RHT, SR to mitigate the activation quantization.

5.4 Results

Table 1: Model arms, parameters, and train and validation losses

Params.	Toks.	Exp Arm.	Train. Loss	Val. Loss
190M	9B	BF16	3.09	3.09
190M	9B	FP4 Quantized All Linear	3.34	3.30
190M	9B	INT4 Quantized Except Router	3.35	3.35
190M	9B	FP4 W4A4 RHT + SR	3.74	3.72
190M	9B	FP4 W4A16 SR	3.14	3.15

Table 1 showcases the various models and their parameters and the loss they achieved for train and validation split.

Figure 2 shows the difference in HellaSwag performance and training curves respective to the 4 experiment arms.

5.4.1 NVFP4 Methodology

Figure 3 is the NVFP4 methodology but with RHT ablated and only keeping SR.

6 Analysis

We are able to successfully quantize weights to 4-bit FP4 values that can nearly match the performance of the baseline model, with a small difference of 8% when comparing validation loss in table 1. There are tradeoffs to the various decisions we made. Below we present the key takeaways.

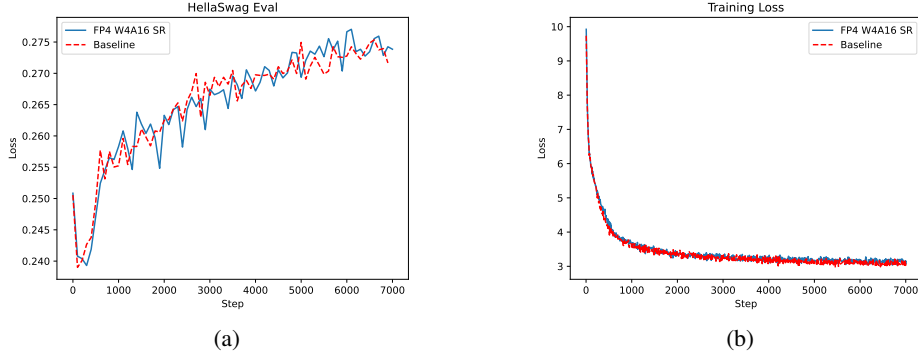


Figure 3: NVFP4 Evaluation and Training Loss

Table 2: COPA Benchmark

Model	COPA (acc.)
Base Model	.279
INT4 Quantized Except Router	.218
FP4 W4A4 RHT + SR	.236
FP4 W4A16 SR	.272

Representation and quantization strategy are very important The hypothesis that INT4 values have lower dynamic range which leads to inefficiency as the extreme buckets are not used seem to have disturbed model quality as well. Compared to FP4 quantization, INT4 performs around 11% worse as it scored around 0.26 for HellaSwag eval vs the baseline score of 0.279. This highlights that the representation and discretization of the parameters matters. Thus training in INT4 would require increasing dynamic range while managing to keep noise to a minimum.

W4A16 to W4A4 did not perform with NVFP4 methodology It is clear that dropping precision or quantizing the activations / gradients leads to immediate decrease in performance. To mitigate the effect of this we tried many different arms to add more precision, through keeping attention layers high-precision, however the hindrance always seemed to be the precision of the activation gradients (See Figure 5 for the gap).

Low-bit representations require micro-scaling datatypes We believe the above issues are due to the naive FP4 representation. Firstly, FP4 is inherently low precision and without newer micro-scaling formats such as NVFP4 or MXFP4 this task requires that we store more information otherwise we will have some sort of lossy compression. MXFP4 stores higher precision 8-bit scale factors and use a power-of-two scale that is precisely chosen to reduce quantization noise and preserve dynamic range.

Ablation We conducted a small ablation study with the NVFP4 methodology show in Figure 4. We find that RHT did not improve outlier smearing and potentially had no impact in the W4A4 quantized experiments. When only using RHT we find that the training stability became much worse. We determine that SR was crucial to improve training stability in W4A4 quantization. Note, this could potentially be due to implementation error, however we did not have the time to accurately measure the reasons for this and leave it to future work.

7 Conclusion

This work presents quantized training in low-bit precisions for small MoE models. We showcase the different tradeoffs that occur when quantizing weights, activations, and gradients. We discuss the potential issues with RHT that showcased incompatibility with MoE models due to their heterogeneity

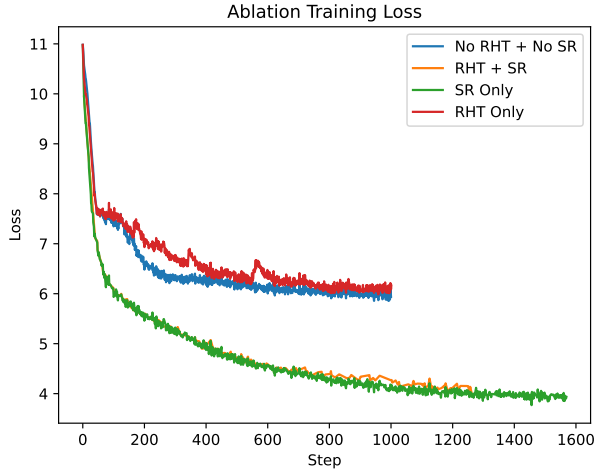


Figure 4: Ablations for NVFP4 methodology in MoE training. Early stopping if results showed non-convergence and instability to save compute.

and sparse nature that require better outlier smearing methods. We find that router sensitivity is important to MoE models and quantizing routers leads to worse performance.

Low-precision training can actively help to decrease training costs, improve compute utilization, and help to make LLMs more accessible for standard consumer grade hardware. We present the initial look into this challenging problem, and find that quantizing weights does not lead to a loss in model performance.

Limitations This work only presented the high level algorithmic simulations for the various FP4 methodologies, and did not test on real hardware that supports FP4 operations. Moreover, we tested on small scale MoE models and future work can test what happens when we scale. Further analysis of outlier smearing and router precision are needed to know with more certainty that quantized pretraining can work.

References

- [1] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.
- [2] Google. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025.
- [3] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojuan Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shutong Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang,

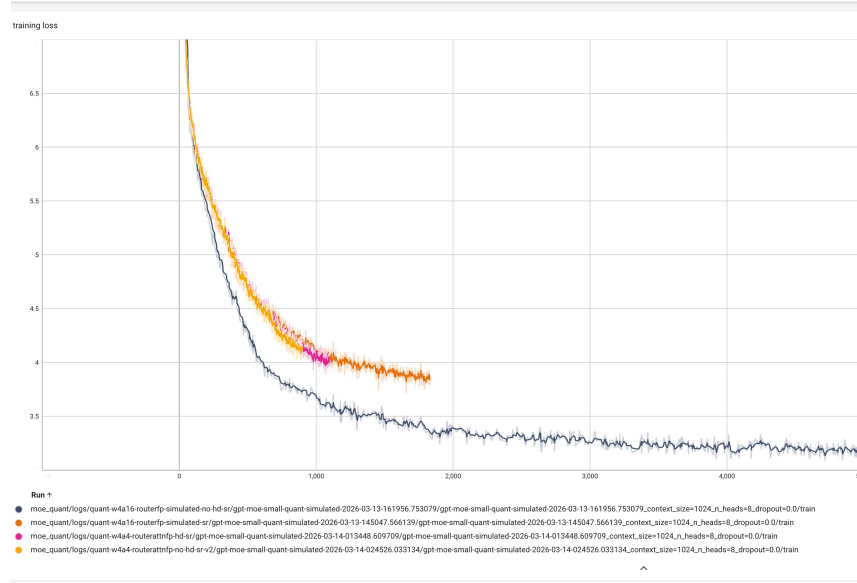
- X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025.
- [4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.
 - [5] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.
 - [6] Zhongqian Fu, Tianyi Zhao, Ning Ding, Xianzhi Yu, Xiaosong Li, Yehui Tang, and Yunhe Wang. Eaquant: Enhancing post-training quantization for moe models via expert-aware optimization, 2026.
 - [7] NVIDIA, Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Alborghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blakeman, Evan Briones, Ian Buck, Bryan Catanzaro, Muya Chang, Jinhang Choi, Mike Chrzanowski, Eric Chung, Victor Cui, Steve Dai, Bit Darvish Rouhani, Carlo del Mundo, Deena Donia, Burc Eryilmaz, Henry Estela, Abhinav Goel, Oleg Goncharov, Yugi Guvvala, Robert Hesse, Russell Hewett, Herbert Hum, Ujval Kapasi, Bruce Khailany, Mikail Khona, Nick Knight, Alex Kondratenko, Ronny Krashinsky, Ben Lanir, Simon Layton, Michael Lightstone, Daniel Lo, Paulius Micikevicius, Asit Mishra, Tim Moon, Deepak Narayanan, Chao Ni, Abhijit Paithankar, Satish Pasumarthi, Ankit Patel, Mostofa Patwary, Ashwin Poojary, Gargi Prasad, Sweta Priyadarshi, Yigong Qin, Xiaowei Ren, Oleg Rybakov, Charbel Sakr, Sanjeev Satheesh, Stas Sergienko, Pasha Shamis, Kirithi Shankar, Nishant Sharma, Mohammad Shoeby, Michael Siu, Misha Smelyanskiy, Darko Stosic, Dusan Stosic, Bor-Yiing Su, Frank Sun, Nima Tajbakhsh, Shelby Thomas, Przemek Tredak, Evgeny Tsykunov, Gandhi Vaithilingam, Aditya Vavre, Rangharajan Venkatesan, Roger Waleffe, Qiyu Wan, Hexin Wang, Mengdi Wang, Lizzie Wei, Hao Wu, Evan Wu, Keith Wyss, Ning Xu, Jinze Xue, Charlene Yang, Yujia Zhai, Ruoxi Zhang, Jingyang Zhu, and Zhongbo Zhu. Pretraining large language models with nvfp4, 2026.
 - [8] NVIDIA. Nvidia blackwell architecture technical brief, 2024.
 - [9] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruijing Wang, Yi Wu, and Furu Wei. Bitnet: Scaling 1-bit transformers for large language models, 2023.
 - [10] Brian Chmiel, Maxim Fishman, Ron Banner, and Daniel Soudry. Fp4 all the way: Fully quantized training of llms, 2025.
 - [11] Albert Tseng, Tao Yu, and Youngsuk Park. Training llms with mxfp4, 2025.
 - [12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
 - [13] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
 - [14] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

- [15] Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. <http://Skylion007.github.io/OpenWebTextCorpus>, 2019.
- [16] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019.
- [17] Pride Kavumba, Naoya Inoue, Benjamin Heinzerling, Keshav Singh, Paul Reisert, and Kentaro Inui. When choosing plausible alternatives, clever hans can be clever. In Simon Ostermann, Sheng Zhang, Michael Roth, and Peter Clark, editors, *Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing*, pages 33–42, Hong Kong, China, November 2019. Association for Computational Linguistics.
- [18] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.
- [19] Cameron R. Wolfe. nanomoe: Mixture-of-experts (moe) llms from scratch in pytorch. 2025.
- [20] Andrej Karpathy. nanogpt. GitHub, 2024.
- [21] Xing Hu, Zhixuan Chen, Dawei Yang, Zukang Xu, Chen Xu, Zhihang Yuan, Sifan Zhou, and Jiangyong Yu. Moequant: Enhancing quantization for mixture-of-experts large language models via expert-balanced sampling and affinity guidance, 2025.

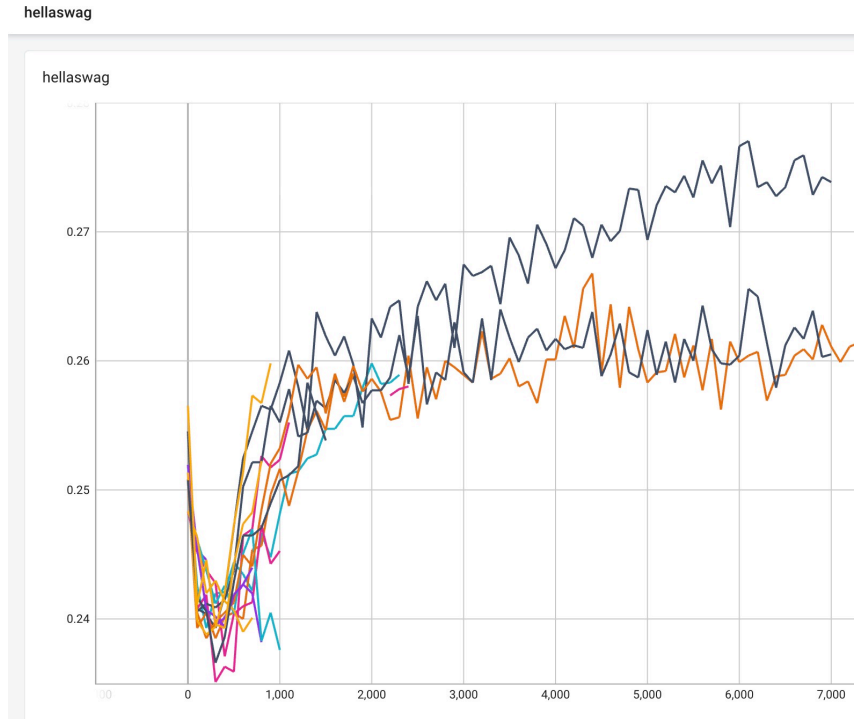
A Appendix

Table 3: Hyperparameter Values for MoE Models

Hyperparameter	Selected Value
Learning Rate	6e-4
Batch Size (Simulated)	2.6M
Micro Batch Size	64
Optimizer	AdamW
Weight Decay λ	1e-1
Betas β_1, β_2	0.9, 0.95
Activation Function	GELU
Capacity Factor	1.25
Number of Experts	8
Top K	2
Stride	2
Embedding Size	768
Block Size	1024
Number of Layers	12
RHT Size	16



(a) Training



(b) HellaSwag

Figure 5: Experiments showcased moving from A4W16 to A4W4 had a threshold that could not be broken with FP4 and simply quantize/not quantizing different parts of the model

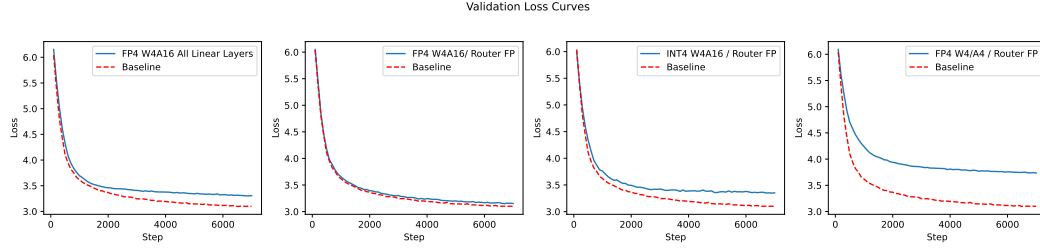


Figure 6: Validation Loss for the experiments

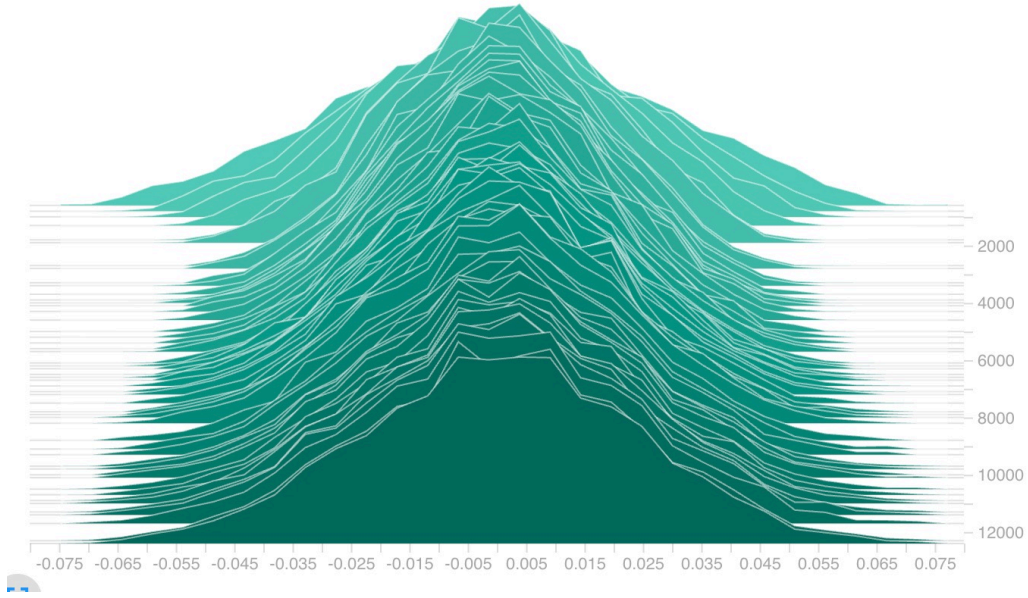


Figure 7: Regular router weights

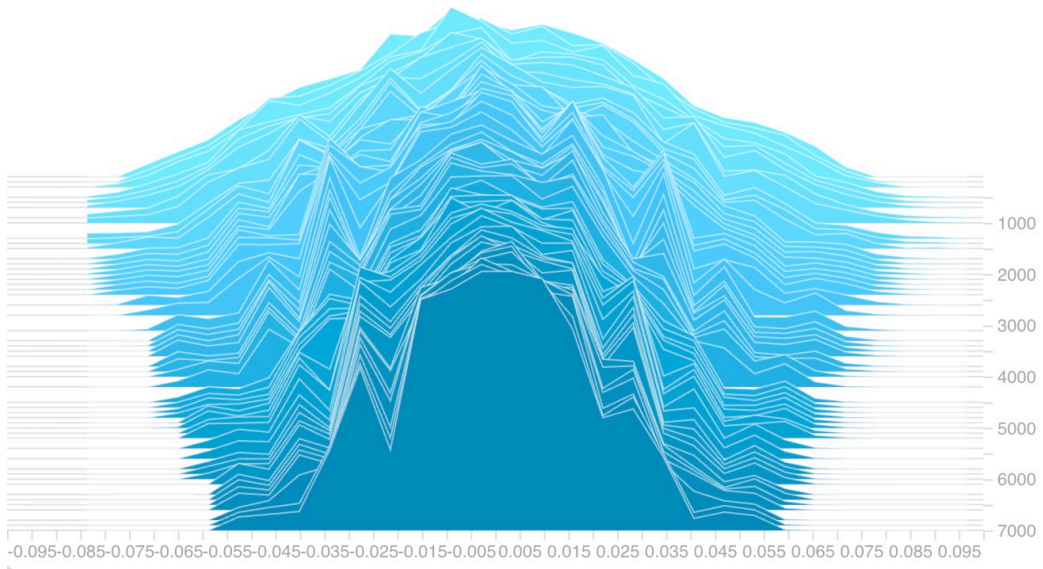


Figure 8: Quantized router weights