
Recipes for Pre-training LLMs with MXFP8

Asit Mishra, Dusan Stosic, Simon Layton, Paulius Micikevicius

NVIDIA

Corresponding authors: asitm@nvidia.com, dstosic@nvidia.com

Abstract

Using fewer bits to represent model parameters and related tensors during pre-training has become a required technique for improving GPU efficiency without sacrificing accuracy. Microscaling (MX) formats [1] introduced in NVIDIA Blackwell generation of GPUs [2] represent a major advancement of this technique, making it practical to combine narrow floating-point data types with finer granularity per-block scaling factors. In turn, this enables both quantization of more tensors than previous approaches and more efficient execution of operations on those tensors.

Effective use of MX-formats requires careful choices of various parameters. In this paper we review these choices and show how MXFP8-E4M3 datatype and a specific number conversion algorithm result in training sessions that match those carried out in BF16. We present results using models with up to 8B parameters, trained on high-quality datasets of up to 15T tokens.

1 Introduction

Scaling the number of parameters of deep learning models, especially large generative models, has intensified the need for more efficient compute and memory solutions. Reducing the number of bits to represent a data type (sometimes referred to as reducing precision) is a successful strategy to improve performance while lowering memory requirements. However, maintaining model accuracy while reducing precision remains a key challenge.

Microscaling (MX) formats, specified by an Open Compute Project working group (OCP) [1], combine narrow floating-point types with fine-grained scaling factors to deliver a balance of dynamic range, precision, and hardware efficiency. OCP v1.0 specification [1] and research works like [3, 4, 5] have shown a fine-grained scaling approach to be an effective strategy compared to granular (e.g., per-tensor) scaling approaches, especially in sub-8-bit pre-training regimes.

NVIDIA’s latest GPU generation, Blackwell [2], adds native MX data type support to Tensor Cores. Blackwell supports MX data for 8-bit (MXFP8), 6-bit (MXFP6) and 4-bit (MXFP4) floating-point types. This paper focuses on MXFP8 pre-training of large language models (LLMs), demonstrating two key factors needed for successful pre-training (no accuracy degradation compared to BF16 sessions):

- First, MXFP8-E4M3 datatype should be used for all tensor types, including activation gradients.
- Second, careful consideration of rounding is needed when converting from high-precision formats to MXFP8, resulting in an algorithm different from the one suggested in OCP v1.0 specification.

For each design choice, we present ablation studies on small models (e.g. 100s of millions of parameters) pretrained on short (e.g. 100s of billions) token horizons. We then apply the findings to large multi-billion-parameter models trained on trillion token horizons.

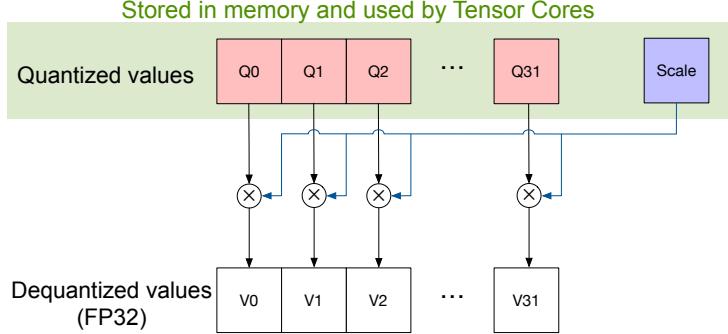


Figure 1: A single MXFP block (in green box) and interpretation of MXFP format.

2 Microscaling format support in NVIDIA Blackwell

Background: An MX-format is specified by the block size K , a shared scaling factor per block, X , and the data-type of elements in the block. A block is a contiguous sequence of K elements. $K = 32$ for all MX types. The data-type of X is UE8M0 and X encodes either NaN or any power-of-two¹ value in the range 2^{-127} to 2^{127} .

Given K input elements in a source-format, V_i (typically FP32); $1 \leq i \leq K$, conversion to MX-format consists of computing X and Q_i such that $Q_i \times X$ is the decoded value corresponding to V_i . X and Q_i are stored in memory instead of V_i . Thus, X serves as the decoding scale factor to decode quantized values back to their high-precision counterparts. This is depicted in Figure 1.

A tensor in source-format is sub-divided into blocks of K elements and converted into MX-format to be stored in memory and/or processed by math units in hardware. Tensor Cores in Blackwell consume X and Q_i to compute the dot product of two MX-formatted blocks. If the accumulated output of a dot-product operation in Tensor Cores is FP32, then this output is thereafter quantized to MX-format if a subsequent operation consuming the output needs it in that format. The conversion process is, $Q_i = \text{Quantize_to_fp8}(V_i/X)$. Section 3 describes the conversion details. Fine-grained scaling helps each MX block independently align to the needed range of input values before quantization.

Table 1: MX-format support in Blackwell

Format	Data type	Max. normal	Min. subnorm.	Binades	Relative speed vs BF16 (Tensor Core math)
MXFP8	E4M3	$1.75 * 2^8$	2^{-9}	17.8	2x
	E5M2	$1.75 * 2^{15}$	2^{-16}	31.8	2x
MXFP6	E2M3	$1.875 * 2^2$	2^{-3}	5.9	2x
	E3M2	$1.75 * 2^4$	2^{-3}	8.8	2x
MXFP4	E2M1	$1.5 * 2^2$	2^{-1}	3.6	4x

Table 1 shows the MX-formats supported in Blackwell. The data type column uses the convention ExMy to denote x-bit for floating-point exponent and y-bit for mantissa. For a fixed bit-width, floating-point numbers trade-off exponent width with mantissa width. Thus, MXFP8 E4M3 is an 8-bit data type with 1 sign bit, 4-bit for FP exponent and 3-bit of mantissa. Similarly, E5M2 is an 8-bit type with 5-bit for exponent and 2-bit for mantissa. Compared to E4M3, E5M2 can represent a larger dynamic range, but with less precision. E5M2 follows IEEE 754 conventions [6] for representation of special values, whereas the remaining data types extend the dynamic range by not representing Infinity and NaN (E4M3 has only one bit-pattern for NaN [7]). More bits in the exponent field translates to a larger range while more bits in the mantissa field translates to more precision within a given range. Every floating-point type has a dynamic range that it can represent — we denote this range in terms of *binades* which is the \log_2 -ratio of the maximum to the minimum finite representable in that format.

¹Power-of-two is a number of the form 2^n where n is a positive integer, negative integer, or zero.

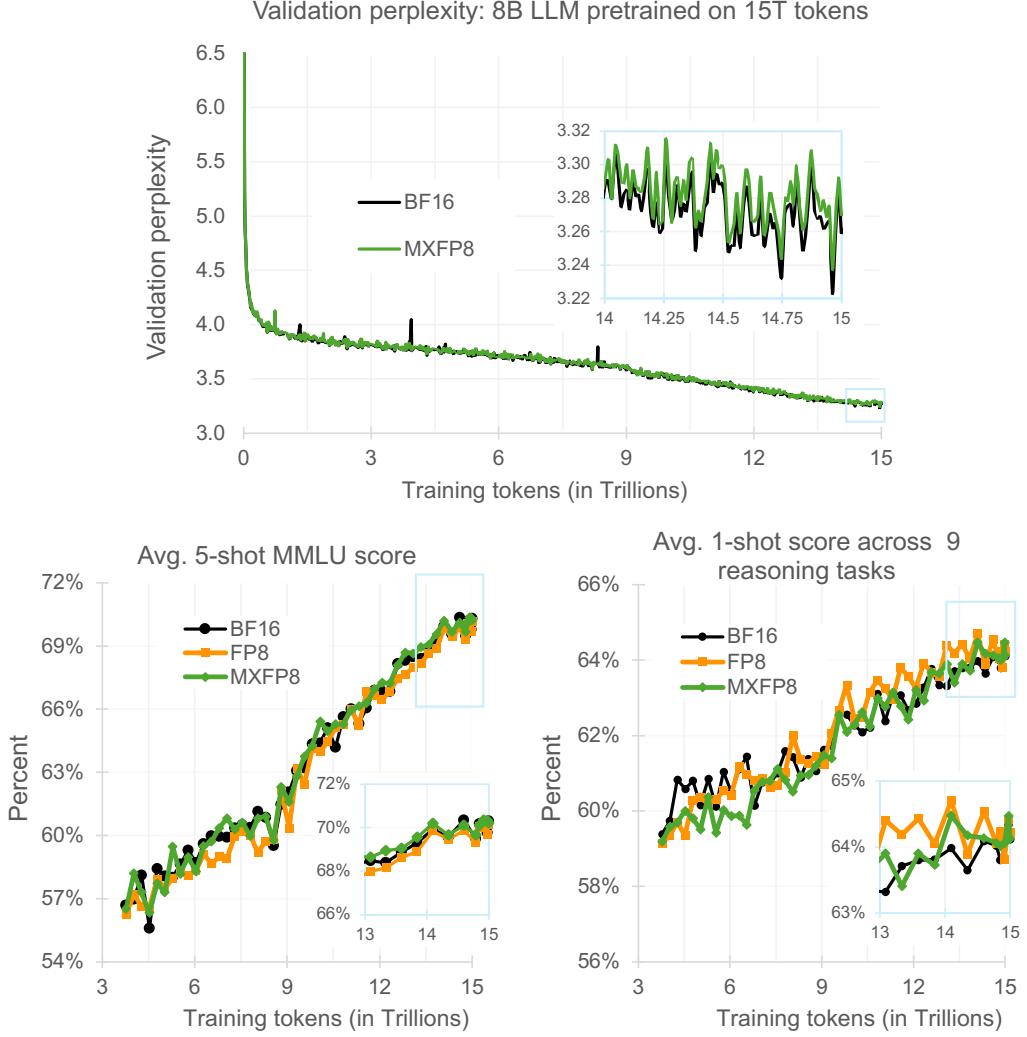


Figure 2: Pre-training a 8B LLM on 15T tokens. **Top:** Training behavior of BF16 vs MXFP8. **Bottom:** Comparing BF16, FP8 and MXFP8’s downstream task scores on MMLU and a set of 9 reasoning tasks. MXFP8 numerics use our proposed rounding method and E4M3 for all quantized tensors.

3 Pre-training with MXFP8

In this section we review MXFP8 training results, training recipe, and the algorithm for converting to MXFP8 used for training. Ablation studies explain the recipe and conversion algorithm choices.

3.1 Training Results

To establish the equivalence of MXFP8 and BF16 for model accuracy we pretrain an 8B parameter model on 15T tokens. We observe:

- Validation perplexity when using MXFP8 matches that of BF16 (top plot in Figure 2). There is less than 0.50% difference between MXFP8 and BF16 validation perplexity values throughout the pre-training run.
- Downstream task evaluation scores match between MXFP8 and BF16 sessions (bottom plots in Figure 2).

The same equivalence is also observed for smaller models or smaller datasets, making MXFP8 a preferred option to efficiently train LLMs. The 8B parameter model [8] consists of 32 transformer

blocks, 32 attention heads. Hidden size is 4096, GQA group size is 8, KV-channels count is 128, sequence length during pre-training is 8192. Initial learning rate is 6e-4 that cosine decays to 6e-6. A phased data-blending approach is used to train the model: in the first phase, a data mixture that promotes diversity in data is used and in the second phase high-quality datasets (e.g., Wikipedia) are used. We switch to the second phase at the 60% point of training. This blend style has also been used in other large scale pre-training setups [9].

Training was carried out using Megatron-LM software [10] on 3072 Hopper GPUs using batch size 768. MX operations were simulated by converting BF16 inputs to MXFP8 and back to BF16 prior to GEMM operations.

Evaluation scores are measured on two sets of downstream tasks: (1) 5-shot score on MMLU [11] and (2) Averaged 1-shot score across 9 general reasoning benchmarks: ARC-Challenge and ARC-Easy [12], Race [13], PIQA [14], Winogrande [15], Hellaswag [16], OpenBookQA [17], Social IQA [18] and Commonsense QA [19].

In summary, we find MXFP8 maintains accuracy compared to BF16 or FP8 pre-trained models. On Blackwell GPU-based systems, MXFP8 has $2\times$ higher throughput than BF16 making end-to-end MXFP8 pre-training faster than BF16 pre-training. We also find the MXFP8 recipe to be simpler to use when compared to FP8 (all layers can be quantized and scaling is handled in the hardware) while allowing for equal or better throughput.

3.2 MXFP8 Training Recipe

The recipe is defined by two choices: which tensors to quantize to MXFP8 and which FP8 binary encoding to use for different tensors.

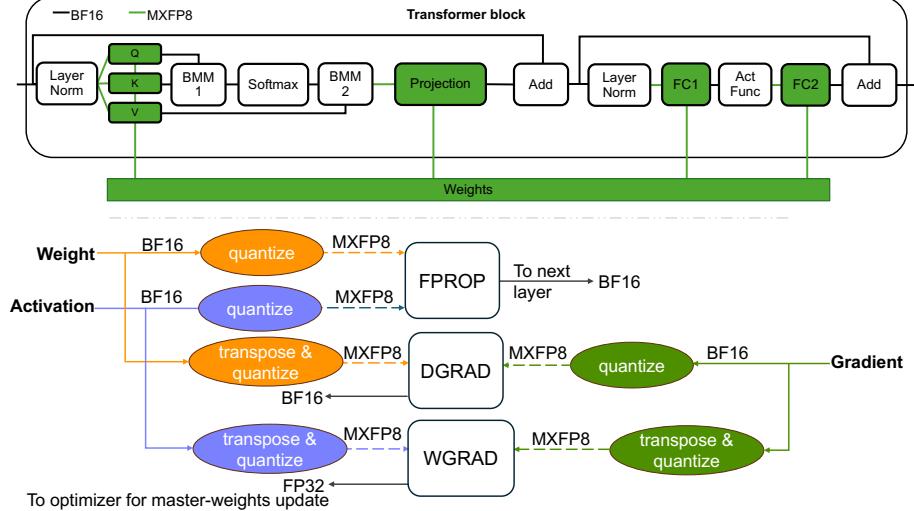


Figure 3: **Top:** Transformer layers quantized to MXFP8 inside a single transformer block. **Bottom:** Training workflow for a single layer during forward-pass (FPROP), error-gradient pass (DGRAD) and weight-gradient pass (WGRAD).

3.2.1 Tensors to Quantize

We execute GEMMs with weights (i.e. QKV and Proj in attention and FFN, as shown in Figure 3) in MXFP8. Therefore, activation, weight, as well as activation gradient input tensors for these GEMMs are converted to MXFP8. The Batch-Matrix Multiplications (BMM1, the query-key dot product and BMM2, the attention score-value product) in the self-attention layer, along with operations like Softmax, Act-func and residual-add are in high-precision. The input embedding layer and the final output-projection layer are also in BF16 or FP16. We leave it to future work to explore reducing the precision of these operations.

The aforementioned tensors are quantized to MXFP8 for all transformer blocks of a model, in contrast to per-tensor [9] or per-row quantization [20] for FP8, where some transformer layers were left in BF16. As a result, MXFP8 due to its fine-granularity scaling factors enables accelerating more of an LLM’s layers.

During training, with MXFP quantization, the training framework keeps two copies of each tensor, each copy is quantized along the axes of dot-product reduction (row and column). Figure 3 shows how each tensor is used in forward (FPROP), weight-gradient (WGRAD) and activation-gradient (DGRAD) computation during the training loop. Since each tensor is used in non-transposed and transposed form, quantization needs to occur along two separate axes (row and column).

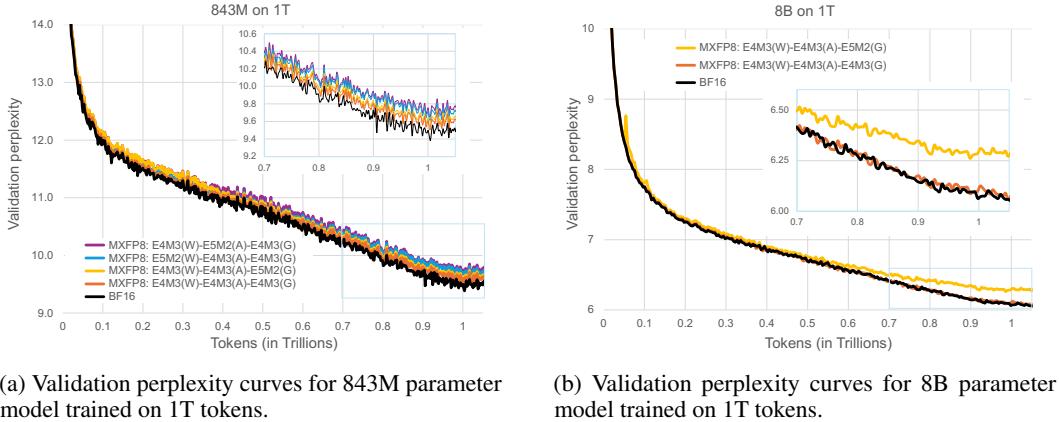


Figure 4: Pre-training loss curves comparing E4M3 and E5M2 when used across different tensor types: weights (W), activation (A) and gradients (G). The inset shows a zoomed-in view of the loss at the end of training.

3.2.2 FP8 Encoding Choice

Our MXFP8 recipe uses E4M3 encoding for all tensor types - weights, activations, and activation gradients. This is in contrast to coarser-scaled FP8 training [9, 7, 21, 22], which used E4M3 for weights and activations but E5M2 for gradients. With fine-grained scaling of MXFP8, the dynamic range requirement at a 32-element block size is sufficiently captured by 17.8 binades of E4M3 type. Once the range requirements are met, precision (or sampling) becomes important. We demonstrate this recipe choice with an empirical study on an 843M and 8B parameter models. Figure 4a shows that E4M3 is the preferred choice for weights and activations, as using E5M2 for these tensor types (the blue and purple curves, respectively) results in a worse perplexity. While either E5M2 or E4M3 encodings are comparably good for the 843M parameter model, in Figure 4b we see that the larger 8B parameter model requires the higher precision of E4M3 for activation gradients (orange curve), as E5M2 gradients (yellow curve) lead to a substantial degradation in perplexity.

3.3 Conversion from FP32 to MXFP8

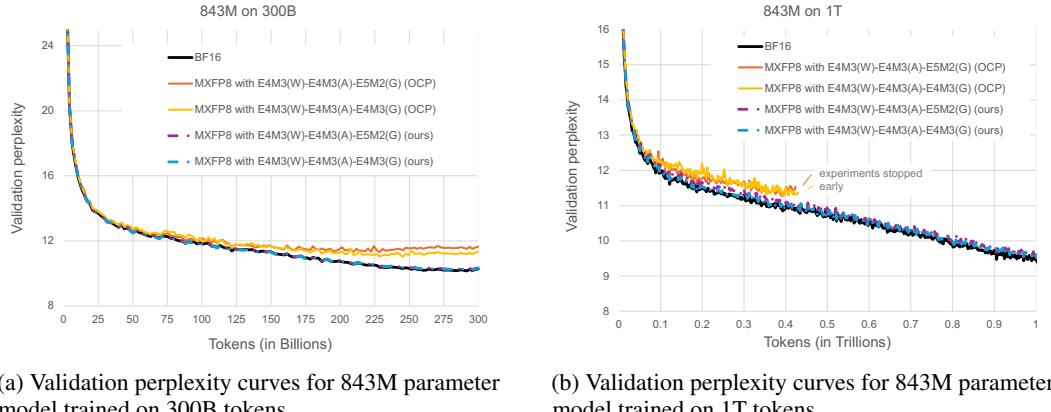
Often the original high precision (for example, FP32 or BF16) values within a block are outside the representable range for the target MX format, both underflowing the minimum and overflowing the maximum representable magnitude. To address this, prior to conversion all values are multiplied with a scale factor so that no value overflows the MX format range, while minimizing flushes to 0. The shared per-block scale factor, X , is computed based on the maximum absolute value (amax) among the 32 high-precision input values, i.e. $\text{amax} = \max \|V_i\|; 1 \leq i \leq 32$. The goal is to map this amax in the input-format to be the largest representable value in the desired MX-format. Once X is computed, V_i is divided by X and the resulting value is quantized to a FP8-representable number using Round-to-nearest-ties-to-even (RN) rounding. Furthermore, saturating conversion mode is used (as described in OCP FP8 specification [23]) i.e. magnitudes exceeding FP8-maximum are clamped to respective maximum representative magnitude in quantized format. Special care is taken if some or all elements in the input are Infinity and/or not-a-number (NaN).

Algorithm 1 Our proposal for computing scale factor, X (simplified method)

$X_{\text{float}} \leftarrow \text{amax}/\text{destmax}$	▷ destmax is max representable in MX-format
$\text{exp}X_{\text{float}} \leftarrow \log_2(X_{\text{float}})$	▷ extract the exponent of X (de-biased form)
$\text{exp}X_{\text{int}} \leftarrow \text{ceil}(\text{exp}X_{\text{float}})$	▷ round-up
$X \leftarrow \text{clamp}(\text{exp}X_{\text{int}}, -127, 127)$	▷ clamp to min/max E8M0 representable
$X \leftarrow X + 127$	▷ add bias
return 2^X	▷ store X

Algorithm 1 outlines the scale-factor computation used by training sessions in Section 3.1. Step 3 is the key part, *rounding-up the scale factor towards positive infinity*, when converting X to a power-of-2 value.. This is different from the scheme suggested by OCP v1 specification, which effectively rounds down the scale factor. Since during quantization to MXFP, high precision values are divided by X, rounding up X ensures that after scaling no V/X value exceeds the maximum magnitude representable in the destination format. Conversely, rounding down, as in OCP v1 suggestion, will lead to scaled values overflowing the representable range, introducing more quantization noise.

We hypothesize this aspect leading to MXFP8 training issues when using OCP v1 conversion scheme as shown in Figure 5. As can be seen in the figure, these issues are eliminated when using our proposal in Algorithm1. A more detailed description of conversion workflow is provided in Appendix Sec. A.1.



(a) Validation perplexity curves for 843M parameter model trained on 300B tokens.

(b) Validation perplexity curves for 843M parameter model trained on 1T tokens.

Figure 5: Comparing scale factor rounding modes suggested in the OCP v1.0 specification and our work. E4M3 and E5M2 are MXFP8 formats; notation E4M3(W) denotes E4M3 data type for Weights. Similarly, E4M3(A) and E4M3 (G) refer to activations and gradients, respectively.

4 Related work

Low-precision training and inference is a widely studied topic in deep learning literature. While significant progress has been made in low-precision inference [24, 25, 26, 27], there are relatively fewer studies demonstrating low-precision techniques for pre-training LLMs, especially large-scale pre-training on a large token horizon. Our work primarily focuses on 8-bit pre-training and prior work on related low-precision LLM pre-training can be grouped into the following two categories:

- **LLM pre-training using FP8 formats:** [7] proposes an FP8 binary interchange format, consisting of E4M3 and E5M2 encodings, and a per-tensor scaling approach — an entire tensor is scaled to capture the maximum number of tensor values in the representable range in FP8. [28] discusses FP8 pre-training challenges and proposes model-level modifications to train a 7B parameter model. Recently, per-tensor scaled FP8 was used to train the Nemotron-H family of LLMs [9] and the Llama-4 family of models also used FP8 [29]. Both of these approaches mention the need to keep a few transformer blocks in high-precision. Instead of per-tensor scaling, DeepSeek-V3 family of models [22] use block-scaled FP8, this helps to better capture outliers and minimize quantization errors. With block-scaling setup, certain tensors require vector 1x128

scaling and certain tensors require per-block (e.g. 128x128) software scaling. Native support for MXFP8 simplifies this — fine-grained scaling provides better numerical robustness and hardware support for scaling avoids any tradeoff between smaller block sizes and hardware speed.

- **Pre-training using MXFP formats:** [3] presents empirical data on pre-training models with MXFP formats. They show cast-only inference results for MXFP8 and pre-training results for MXFP6 and MXFP4-weights. [30] studies MXFP4 backward-pass quantization and [31] investigates MXFP4 weight quantization on relatively small token horizon.

5 Conclusions and future work

We present studies on several aspects of MXFP8-formats and their usage in pre-training models, e.g. scale computation, data format of specific tensors, numerical aspects of quantization, model layers that could be quantized for maximizing training throughput. We discuss recipes for pre-training LLMs with MXFP8-formats. The MXFP8 pre-training recipe is implemented in Transformer Engine [32] and the conversion numerics in cuDNN and cuBLAS libraries.

As part of our future work, we plan to extend our study to post-training phases (e.g. phases that perform reinforcement learning (RL)-based policy optimizations and their variants [33] as well as supervised fine-tuning stages). Additionally, from a hardware systems performance perspective, since tensors have to be quantized along two separate axes with MX-formats, the training framework needs to store two copies for each tensor (we discussed this in section 3.2). We plan to study scaling methods that would lower the storage overhead.

Acknowledgments

We thank Sweta Priyadarshi, Mikail Khona and Ben Lanir for helping with the draft revisions as well as Mohammad Shoeybi, Carlo del Mundo, Michael Andersch, Eric Chung and Bryan Catanzaro for valuable feedback and discussions.

References

- [1] Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Jasmine Klar, Eric Chung, Yuan Yu, Michael Schulte, Ralph Wittig, Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Summer Deng, Maxim Naumov, Paulius Micikevicius, Michael Siu, and Colin Verrilli. Ocp microscaling (mx) specification. *Open Compute Project*, 2023.
- [2] Nvidia Blackwell Architecture Technical Brief. URL <https://resources.nvidia.com/en-us-blackwell-architecture>.
- [3] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, Stosic Dusan, Venmugil Elango, Maximilian Golub, Alexander Heinecke, Phil James-Roxby, Dharmesh Jani, Gaurav Kolhe, Martin Langhammer, Ada Li, Levi Melnick, Maral Mesmakhosroshahi, Andres Rodriguez, Michael Schulte, Rasoul Shafipour, Lei Shao, Michael Siu, Pradeep Dubey, Paulius Micikevicius, Maxim Naumov, Colin Verrilli, Ralph Wittig, Doug Burger, and Eric Chung. Microscaling data formats for deep learning, 2023. URL <https://arxiv.org/abs/2310.10537>.
- [4] Bita Rouhani, Ritchie Zhao, Venmugil Elango, Rasoul Shafipour, Mathew Hall, Maral Mesmakhosroshahi, Ankit More, Levi Melnick, Maximilian Golub, Girish Varatkar, Lei Shao, Gaurav Kolhe, Dimitry Melts, Jasmine Klar, Renee L'Heureux, Matt Perry, Doug Burger, Eric Chung, Zhaoxia Deng, Sam Naghshineh, Jongsoo Park, and Maxim Naumov. With shared microexponents, a little shifting goes a long way, 2023. URL <https://arxiv.org/abs/2302.08007>.
- [5] Steve Dai, Rangharajan Venkatesan, Haoxing Ren, Brian Zimmer, William J. Dally, and Brucek Khailany. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference, 2021. URL <https://arxiv.org/abs/2102.04503>.
- [6] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008. doi: 10.1109/IEEESTD.2008.4610935.
- [7] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellem-pudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. Fp8 formats for deep learning, 2022. URL <https://arxiv.org/abs/2209.05433>.
- [8] Jupinder Parmar, Shrimai Prabhumoye, Joseph Jennings, Mostofa Patwary, Sandeep Subramanian, Dan Su, Chen Zhu, Deepak Narayanan, Aastha Jhunjhunwala, Ayush Dattagupta, Vibhu Jawa, Jiwei Liu, Ameya Mahabaleshwarkar, Osvald Nitski, Annika Brundyn, James Maki, Miguel Martinez, Jiaxuan You, John Kamalu, Patrick LeGresley, Denys Fridman, Jared Casper, Ashwath Aithal, Oleksii Kuchaiev, Mohammad Shoeybi, Jonathan Cohen, and Bryan Catanzaro. Nemotron-4 15b technical report, 2024. URL <https://arxiv.org/abs/2402.16819>.
- [9] NVIDIA, :, Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, Akhiad Bercovich, Aleksander Ficek, Alexis Bjorlin, Ali Taghibakhshi, Amala Sanjay Deshmukh, Ameya Sunil Mahabaleshwarkar, Andrew Tao, Anna Shors, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaju, Bobby Chen, Boris Ginsburg, Boxin Wang, Brandon Norick, Brian Butterfield, Bryan Catanzaro, Carlo del Mundo, Chengyu Dong, Christine Harvey, Christopher Parisien, Dan Su, Daniel Korzekwa, Danny Yin, Daria Gitman, David Mosal-lanezhad, Deepak Narayanan, Denys Fridman, Dima Rekesh, Ding Ma, Dmytro Pykhtiar, Dong Ahn, Duncan Riach, Dusan Stosic, Eileen Long, Elad Segal, Ellie Evans, Eric Chung, Erick Galinkin, Evelina Bakhturina, Ewa Dobrowolska, Fei Jia, Fuxiao Liu, Gargi Prasad, Gerald Shen, Guilin Liu, Guo Chen, Haifeng Qian, Helen Ngo, Hongbin Liu, Hui Li, Igor Gitman, Ilia Karmanov, Ivan Moshkov, Izik Golan, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jarno Seppanen, Jason Lu, Jason Sewall, Jiaqi Zeng, Jiaxuan You, Jimmy Zhang, Jing Zhang, Jining Huang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jon Barker, Jonathan Cohen, Joseph Jennings, Jupinder Parmar, Karan Sapra, Kari Briski, Kateryna Chumachenko, Katherine Luna, Keshav Santhanam, Kezhi Kong, Kirthi Sivamani, Krzysztof Pawelec, Kumar Anik, Kunlun Li, Lawrence McAfee, Leon Derczynski, Lindsey Pavao, Luis Vega, Lukas Voegtle,

Maciej Bala, Maer Rodrigues de Melo, Makesh Narsimhan Sreedhar, Marcin Chochowski, Markus Kliegl, Marta Stepniewska-Dziubinska, Matthieu Le, Matvei Novikov, Mehrzad Samadi, Michael Andersch, Michael Evans, Miguel Martinez, Mike Chrzanowski, Mike Ranzinger, Mikolaj Blaz, Misha Smelyanskiy, Mohamed Fawzy, Mohammad Shoeybi, Mostofa Patwary, Nayeon Lee, Nima Tajbakhsh, Ning Xu, Oleg Rybakov, Oleksii Kuchaiev, Olivier Delalleau, Osvald Nitski, Parth Chadha, Pasha Shamis, Paulius Micikevicius, Pavlo Molchanov, Peter Dykas, Philipp Fischer, Pierre-Yves Aquilanti, Piotr Bialecki, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Rabeeh Karimi, Rahul Kandu, Ran El-Yaniv, Raviraj Joshi, Roger Waleffe, Ruoxi Zhang, Sabrina Kavanaugh, Sahil Jain, Samuel Kriman, Sangkug Lym, Sanjeev Satheesh, Saurav Muralidharan, Sean Narenthiran, Selvaraj Anandaraj, Seomyeong Bak, Sergey Kashirsky, Seungju Han, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Sharon Clay, Shelby Thomas, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shyamala Prayaga, Siddhartha Jain, Sirshak Das, Slawek Kierat, Somshubra Majumdar, Song Han, Soumye Singhal, Sriharsha Niverty, Stefania Alborghetti, Suseella Panguluri, Swetha Bhendigeri, Syeda Nahida Akter, Szymon Migacz, Tal Shiri, Terry Kong, Timo Roman, Tomer Ronen, Trisha Saar, Tugrul Konuk, Tuomas Rintamaki, Tyler Poon, Ushnish De, Vahid Noroozi, Varun Singh, Vijay Korthikanti, Vitaly Kurin, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wenliang Dai, Wonmin Byeon, Xiaowei Ren, Yao Xu, Yejin Choi, Yian Zhang, Ying Lin, Yoshi Suhara, Zhiding Yu, Zhiqi Li, Zhiyu Li, Zhongbo Zhu, Zhuolin Yang, and Zijia Chen. Nemotron-h: A family of accurate and efficient hybrid mamba-transformer models, 2025. URL <https://arxiv.org/abs/2504.03624>.

- [10] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL <https://arxiv.org/abs/1909.08053>.
- [11] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL <https://arxiv.org/abs/2009.03300>.
- [12] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.
- [13] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.
- [14] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL <https://arxiv.org/abs/1911.11641>.
- [15] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL <https://arxiv.org/abs/1907.10641>.
- [16] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL <https://arxiv.org/abs/1905.07830>.
- [17] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL <https://arxiv.org/abs/1809.02789>.
- [18] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialqa: Commonsense reasoning about social interactions, 2019. URL <https://arxiv.org/abs/1904.09728>.
- [19] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL <https://aclanthology.org/N19-1421>.

- [20] Aaron Grattafiori et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- [21] Badreddine Noune, Philip Jones, Daniel Justus, Dominic Masters, and Carlo Luschi. 8-bit numerical formats for deep neural networks, 2022. URL <https://arxiv.org/abs/2206.02915>.
- [22] 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, Miaojun 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, Shanhua Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting 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, Yuduan 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. URL <https://arxiv.org/abs/2412.19437>.
- [23] Paulius Micikevicius, Stuart Oberman, Pradeep Dubey, Marius Cornea, Andres Rodriguez, Ian Bratt, Richard Grisenthwaite, Norm Jouppi, Chiachen Chou, Amber Huffman, Michael Schulte, Ralph Wittig, Dharmesh Jani, and Summer Deng. Ocp 8-bit floating point specification (ofp8). URL <https://www.opencompute.org/documents/ocp-8-bit-floating-point-specification-ofp8-revision-1-0-2023-12-01-pdf-1>.
- [24] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024. URL <https://arxiv.org/abs/2211.10438>.
- [25] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving, 2025. URL <https://arxiv.org/abs/2405.04532>.
- [26] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL <https://arxiv.org/abs/2210.17323>.
- [27] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL <https://arxiv.org/abs/2306.00978>.
- [28] Maxim Fishman, Brian Chmiel, Ron Banner, and Daniel Soudry. Scaling fp8 training to trillion-token llms, 2025. URL <https://arxiv.org/abs/2409.12517>.

- [29] Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>, April 2025. Accessed 12 May 2025.
- [30] Albert Tseng, Tao Yu, and Youngsuk Park. Training llms with mxfp4, 2025. URL <https://arxiv.org/abs/2502.20586>.
- [31] Ruizhe Wang, Yeyun Gong, Xiao Liu, Guoshuai Zhao, Ziyue Yang, Baining Guo, Zhengjun Zha, and Peng Cheng. Optimizing large language model training using fp4 quantization, 2025. URL <https://arxiv.org/abs/2501.17116>.
- [32] Nvidia. Transformer engine. <https://github.com/NVIDIA/TransformerEngine/>.
- [33] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- [34] Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective, 2024. URL <https://arxiv.org/abs/2410.05192>.
- [35] Nvidia, :, Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H. Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, Sirshak Das, Ayush Dattagupta, Olivier Delalleau, Leon Derczynski, Yi Dong, Daniel Egert, Ellie Evans, Aleksander Ficek, Denys Fridman, Shaona Ghosh, Boris Ginsburg, Igor Gitman, Tomasz Grzegorzek, Robert Hero, Jining Huang, Vibhu Jawa, Joseph Jennings, Aastha Jhunjhunwala, John Kamalu, Sadaf Khan, Oleksii Kuchaiiev, Patrick LeGresley, Hui Li, Jiwei Liu, Zihan Liu, Eileen Long, Ameya Sunil Mahabaleshwarkar, Somshubra Majumdar, James Maki, Miguel Martinez, Maer Rodrigues de Melo, Ivan Moshkov, Deepak Narayanan, Sean Narendhiran, Jesus Navarro, Phong Nguyen, Osvald Nitski, Vahid Noroozi, Guruprasad Nuttheti, Christopher Parisien, Jupinder Parmar, Mostofa Patwary, Krzysztof Pawelec, Wei Ping, Shrimai Prabhumoye, Rajarshi Roy, Trisha Saar, Vasanth Rao Naik Sabavat, Sanjeev Satheesh, Jane Polak Scowcroft, Jason Sewall, Pavel Shamis, Gerald Shen, Mohammad Shoeybi, Dave Sizer, Misha Smelyanskiy, Felipe Soares, Makesh Narsimhan Sreedhar, Dan Su, Sandeep Subramanian, Shengyang Sun, Shubham Toshniwal, Hao Wang, Zhilin Wang, Jiaxuan You, Jiaqi Zeng, Jimmy Zhang, Jing Zhang, Vivienne Zhang, Yian Zhang, and Chen Zhu. Nemotron-4 340b technical report, 2024. URL <https://arxiv.org/abs/2406.11704>.

A Appendix

A.1 UE8M0 rounding

Computation described in Algorithm 1 is a simplification. The simplification comes from storing the output of $\log_2(\text{float } x)$ in FP32. \log_2 function on device internally performs a round-to-nearest of the resulting return value and thus the result of $\text{ceil}(\log_2(x))$ can be different if the output of \log_2 is not stored in a sufficiently large data-type. Hence, for emulation purposes we work directly with the bit representation of the ratio of `amax` and `destmax`. We, next, describe the computation flow.

Background: As a reminder, the quantization process from 32 high-precision values, V_i , to quantized values, Q_i ; $1 \leq i \leq 32$, is given by: $Q_i = \text{Quantize_to_fp8}(V_i/X)$. X is the scale factor; the exponent of X is stored in an 8bit integer container in memory and interpreted as 2 raised to this exponent value by the hardware (after a bias correction). This scale factor *decodes* Q_i back to V_i (with quantization loss).

Value of scale factor (X) = `float_to_8bits(amax/destmax)`, where `amax` is absolute maximum in input/source block (of 32 elements) and `destmax` is the largest positive number in the destination (MX) number system. `float_to_8bits` converts a floating-point number to a power-of-two number.

A float (FP32) number can be represented in IEEE convention as $2^E \times 1.\text{mantissa}$ (normal) or $2^{-126} \times 0.\text{mantissa}$ (subnormal). E can lie between -127 to 127 (or 0 to 254 with the exponent bias) and can be represented in the 8-bit container for scale factor. -126 (or 1 with the exponent bias) is also representable by 8-bit container. `mantissa` lies between [0, 1). So, the question is: should mantissa bits be rounded-up, rounded-down, round-to-nearest, discarded, etc. to create a power-of-two number? We find round-up to be the best choice for pre-training with MX-formats.

Rounding: For `float_to_8bits()`, the recommended order of computation is:

1. Compute the decoding scale as: `decode_scale = block_amax/destmax`
2. if `decode_scale` is below 2^{-127} , then set it to 2^{-127} (which is the smallest value representable in UE8M0)
3. For all other values that are not powers of 2, *round-up* to the closest representable UE8M0 value.

By construction `amax/destmax` never exceeds 2^{127} (which is the largest value representable in UE8M0) with FP8, FP6 or FP4 formats. The above computations are done in the bit-space in emulation.

A.2 MX-format conversion

Section 2 relies on the standard MX-format conversion algorithm defined in [3], but for completeness we show it here given a shared scaling exponent X as computed in 3.3.

Quantizing FP32 values to MX type: After X is computed, V_i/X is computed and the resulting value is quantized to a FP8-representable number (`Quantize_to_fp8()`). Round-to-nearest-ties-to-even (RN) rounding is used during this quantization step. The conversion process is saturating, i.e. if after rounding the resulting value exceeds FP8 max or is less than FP8 min value, then the result is clamped to respective max or min value.

Quantization operations add computation overhead — Blackwell has hardware support for rounding the scale (using our proposed method) and quantizing values to lower this overhead.

A.3 Why is special hardware needed for MX-formats?

Computing the matrix-product of two tensors involves performing dot-products between sub-vectors of the two tensors. Therefore, scaling factors need to be processed once per group of values that share the scale. Since MX-formats have fine-grained scaling, scale factors are processed once after each block's dot-product is computed, thus, many times per tensor-wide dot-product. This is expensive to do in software, so hardware needs to add support for accelerating tensor operations involving MX-formats (e.g. Blackwell).

A.4 MXFP8 pre-training for a mixture-of-experts model

Section 3.1 discusses empirical data that shows MXFP8 matches BF16 accuracy (both training loss as well as downstream task accuracy). Transformer based mixture-of-experts (MoE) models are popular in literature. Figure 6 shows that MXFP8 pre-training also matches BF16 pre-training loss curve for a MoE setup that we experimented with. The MoE model has 16 billion total parameters and ~ 2.5 billion active parameters and we train the model on 1 trillion tokens. We follow the same guidelines discussed in section 3.2 for pre-training the MoE model. The pre-training phase uses a WSD [34] learning rate schedule. The final loss of the MXFP8-trained MoE model is within 0.1% of BF16-training.

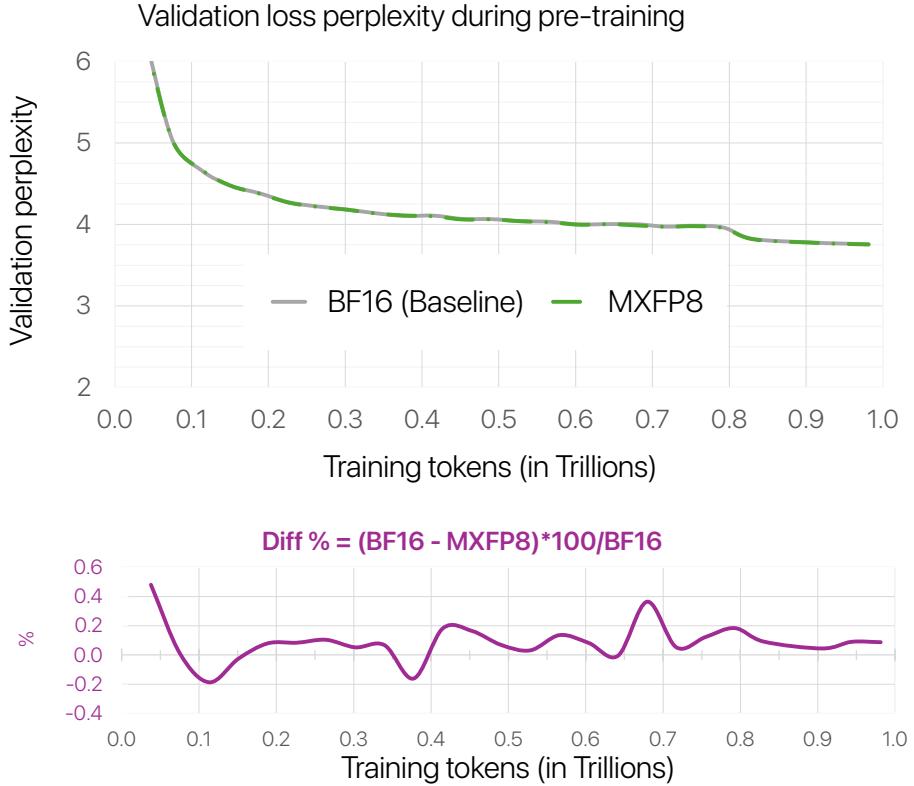


Figure 6: MXFP8 versus BF16 for a MoE model

A.5 Model configurations

We conduct numerical experiments on LLM pre-training with variants of Nemotron-4 [8] models. Training and model details are described below. The 1T and 300B tokens dataset are a subset of the 17T data set discussed in [35]. Table 2 details the parameters for the various models that were used.

Table 2: Configuration for the LLM models.

Model	Layers	Hidden Size	Attention Heads	Sequence Length	Batch Size	Initial LR	Final LR
843M	24	1024	16	4096	256	2.5e-4	2.5e-7
8B	32	4096	32	4096	1024	3e-4	3e-7