arXiv:2310.10537v3 [cs.LG] 19 Oct 2023

Microscaling Data Formats for Deep Learning

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** **Eric Chung**

Microsoft *AMD* *Intel* *Meta* *NVIDIA* *Qualcomm Technologies Inc.*

Abstract

Narrow bit-width data formats are key to reducing the computational and storage costs of modern deep learning applications. This paper evaluates Microscaling (MX) data formats that combine a per-block scaling factor with narrow floating-point and integer types for individual elements. MX formats balance the competing needs of hardware efficiency, model accuracy, and user friction. Empirical results on over two dozen benchmarks demonstrate practicality of MX data formats as a drop-in replacement for baseline FP32 for AI inference and training with low user friction. We also show the first instance of training generative language models at sub-8-bit weights, activations, *and* gradients with minimal accuracy loss and no modifications to the training recipe.

☆
Weight
Activation
Gradient] All sub-8 bit

1 Introduction

Recent advances in AI capabilities such as conversational question answering, intelligent code completion, and text-to-image generation have seen rapid adoption in practical technologies. These advances have been realized primarily through scaling up the size of the underlying deep learning model. However, this scaling up has led to a significant increase in the computing power and storage capacity necessary to train and deploy such models.

One method to reduce deep learning models' computational and storage cost is to use low bit-width data formats instead of the conventional FP32. Great strides have been made to enable training using FP16, Bfloat16, and most recently FP8 [1], as well as to perform inference in narrow integer formats like INT8. Native support for low bit-width formats is now commonplace in AI-oriented hardware such as GPUs, TPUs, and edge inference devices. The narrowest formats, such as FP8 and INT8, require per-tensor scaling factors to adjust to the dynamic range of each tensor. Tensor level scaling has has been shown to be insufficient, though, for sub-8-bit formats due to their limited dynamic range. Research has shown that micro scaled data formats that associate scaling factors with fine-grained sub-blocks of a tensor are more effective in sub-8 bit regime (e.g., [2; 3; 4; 5]).

For sub-8 bit, tensor level scaling insufficient.

Fine-grained sub-block scaling

This paper evaluates Microscaling (MX) data formats [6] — the first open standard for a family of micro-scaled datatypes aimed at deep learning training and inference. The MX standard aims to create an effective data format by achieving a balance among three key factors:

- **Hardware Efficiency** — Maximize compute and storage efficiency via reduced bit-width.
- **Model Accuracy** — Minimize the gap in the quality of results compared with baseline FP32 for AI training and inference.

[*] email correspondence: birouhan@microsoft.com

- **User Friction** — Ensure seamless integration within existing training and inference frameworks and generalizability across different workloads.

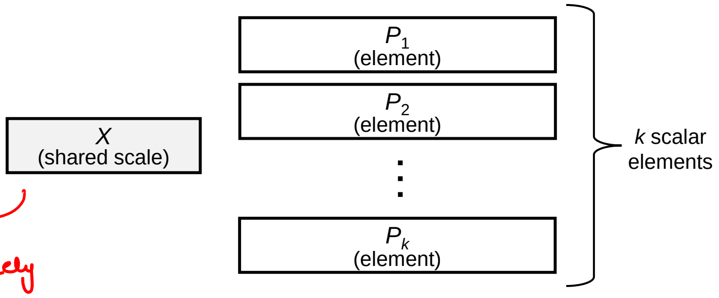
Details on the MX standard and the concrete binary formats can be found in the OCP Microscaling Specification [6]. This paper will focus on the empirical results of using MX formats for direct-cast inference, error diffusion inference, and finetuned inference, as well as training on various benchmarks. Our results corroborate the effectiveness of MX formats in balancing the competing demands of hardware efficiency, model accuracy, and user friction. 8-bit MX formats can perform inference directly on FP32 pretrained models with minimal accuracy loss and without the need for calibration or finetuning. Inference with 6-bit MX formats is also very close to FP32 after quantization-aware fine-tuning or using a post-training quantization method. Using 6-bit MX formats, we demonstrate the first instance of training large transformer models with sub-8-bit weights, activations, and gradients to an accuracy matching FP32 without modifications to the training recipe. Going even further, we show that training of large transformers can be done with 4-bit MX format weights, incurring only a minor accuracy drop.

The custom CUDA library to emulate MX formats on existing GPUs can be found at [7]. This library can be used to reproduce the experimental results reported in this paper.

2 Microscaling

A basic unit of data in an MX format represents a vector of k numbers and consists of a single *shared scale* X and k scalar *elements* $\{P_i\}_{i=1}^{k}$ (see Figure 1). This unit of data is called an MX block and is defined by the combination of *block size* k, scale data format, and element data format. The two data formats are independent of one another, and all k elements share the same element data format. The layout of an MX block is not prescribed — an implementation may store X contiguously with or separately from the elements.

Scale and data dtypes can be different



Can be stored separately

Figure 1: A single block in a Microscaling data format. The block encodes a vector of k numbers, each with value XP_i.

Let $\{v_i\}_{i=1}^{k}$ be the k real numbers represented in an MX block. The value of each number can be inferred as follows:

- If $X = \text{NaN}$, then $v_i = \text{NaN}$ for all i
- If $|XP_i| > Vmax_{Float32}$ then v_i is implementation-defined
- Otherwise, $v_i = XP_i$

where $Vmax_{Float32}$ refers to the largest representable magnitude in IEEE Float32.

2.1 Special Value Encodings

MX formats can encode NaN in up to two ways. First: if X is NaN, then all k values in the MX block is NaN regardless of the encodings of P_i. Second: if X is not NaN, each element P_i may individually encode NaN.

Depending on the element format, MX formats can encode Inf by letting X be a number (i.e., not a NaN) and each P_i individually encode Inf. The shared scale X does not encode Inf.

NaN
X can be NaN
or Individual Pi can be NaN

Inf
Only individual Pi can be Inf.
X can not be Inf

2.2 Concrete MX Formats

Table 1 shows the parameters that define the concrete MX formats, which are named by prepending "MX" to the name of the element data format. All concrete MX formats use E8M0 (an 8-bit exponent) as the format for the shared scale. The representable exponents of these formats is a superset of the representable exponents of FP32.

Details on the FP8 element data formats can be found in the OCP FP8 specification [1]. Details on the other element data formats and the E8M0 scale format can be found in the OCP Microscaling Specification [6].

Table 1: Concrete MX-compliant data formats and their parameters.

Format Name	Block Size	Scale Data Format	Scale Bits	Element Data Format	Element Bit-width
MXFP8	32	E8M0	8	FP8 (E4M3 / E5M2)	8
MXFP6	32	E8M0	8	FP6 (E2M3 / E3M2)	6
MXFP4	32	E8M0	8	FP4 (E2M1)	4
MXINT8	32	E8M0	8	INT8	8

3 Scalar Float to MX Format Conversion

In this paper, we use Algorithm 1 for conversion from scalar floating-point format (e.g., FP32) to an MX format. This algorithm follows the semantics outlined in Section 6.3 of the OCP Microscaling Specification [6], and is provided as a working example. Note that, the specification allows for other implementation-defined conversion recipes — i.e., conversion to MX formats is *not necessarily required* to follow Algorithm 1.

Algorithm 1 Convert vector of scalar floats $\{V_i\}_{i=1}^k$ to an MX block $\{X,\ \{P_i\}_{i=1}^k\}$

```
Require: $emax_{elem}$ = exponent of the largest normal number in the element data format
1: $shared\_exp \leftarrow \lfloor \log_2(\max_i(|V_i|)) \rfloor - emax_{elem}$
2: $X \leftarrow 2^{shared\_exp}$
3: for $i = 1$ to $k$ do
4:     $P_i = quantize\_to\_element\_format(V_i/X)$, clamping normal numbers
5: end for
6: return $X,\ \{P_i\}_{i=1}^k$
```

On Line 1, $shared\_exp$ contains an offset of $emax\_elem$ to map the max input exponent to the largest binade in the element data format. This enables full utilization of the element data format's exponent range.

On Line 4, when quantizing V_i/X, normal numbers that exceed the representable range of the element format are clamped to the maximum representable value, preserving the sign. Infs and NaNs are not clamped. This is in accordance with the OCP MX specification.

On Line 4, P_i is set to zero if the corresponding input V_i is a subnormal Float32 number. This is not described in the OCP MX specification and was done to simplify the algorithm.

MX Conv. + Transpose
≠
Transpose + MX Conv.

When converting multi-dimensional tensors, a principle axis must be selected for the shared scale (typically the reduction dimension in matrix multiplication). For a 2D matrix, the scale can be shared by every k element in a row or column. Transposing a 2D matrix in an MX format changes the axis of the shared scale — i.e., conversion to MX format and transposing are not commutative operations.

4 Experimental Results

4.1 Compute Flow

Figure 2 shows an example compute flow for training using an MX format. For operations involving dot products (e.g., matmul and convolution) in both forward and backward passes, the two inputs are converted to MX format, and the operation is performed using the efficient dot product from Section 6.2 of the OCP Microscaling Specification [6]. Vector operations (e.g., layernorm, Softmax, GELU, and residual add) are performed in a scalar floating-point format like Bfloat16 or FP32. The dot product operations produce outputs in the scalar float format. A master copy of the weights is kept in FP32, and this copy is updated in each training step. In all the training examples in this paper, we use the compute flow illustrated in Figure 2.

Output ← in BF16/FP32

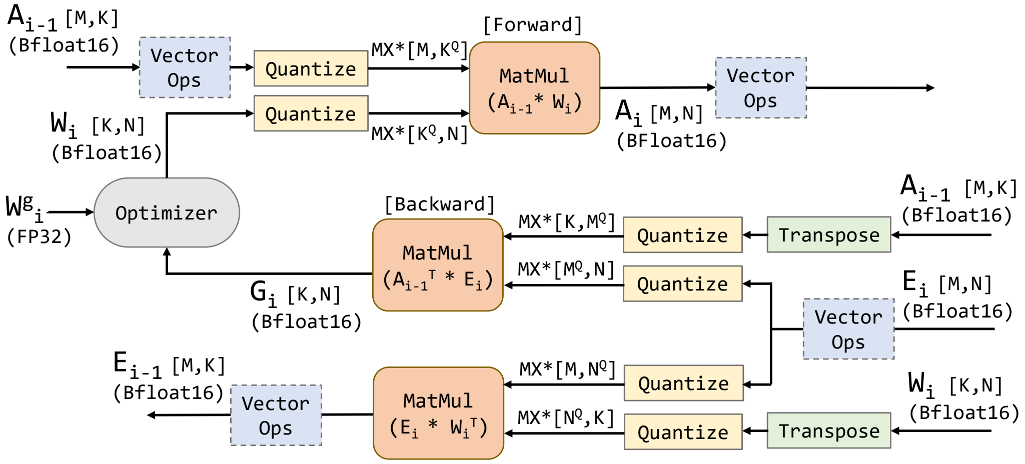


Figure 2: Compute flow with MX formats (denoted as MX*). In the diagram, MatMul includes any dot product operation such as matmul, linear, and convolution. Vector Ops include non-dot product operations like activations, normalization, Softmax, and residual add.

Due to non-commutative nature of transpose and quantization into MX formats (see Section 3), the quantized weights W_i and their transpose W_i^T must be stored as two separate tensors. Note that the two tensors do not need to be stored in working memory simultaneously unless a very fine-grained interleaving of the forward and backward passes is employed.

4.2 Methodology

We used a custom library to emulate MX formats on existing GPUs. The library is implemented as a custom CUDA extension in PyTorch and performs quantization following Figure 2. In particular, we explored four settings:

- *Direct-cast Inference*. The quantized inference is performed on a trained FP32 model. All GeMMs in the forward pass are quantized unless explicitly called out otherwise (the backward pass is not executed at all).
- *Error Diffusion Inference*. The error diffusion algorithm is a Post Training Quantization (PTQ) algorithm derived from GPFQ [8]. It performs quantization using a small calibration dataset. In this experiment, all activations and weights in the forward pass are quantized to the same format for simplicity. This PTQ process is a quick one-pass process without a training loop or needing any tuning parameter.
- *Finetuned Inference*. Quantization-aware finetuning is done on a trained FP32 model for a small number of epochs. For this fine-tuning, all GeMMs in the forward pass are quantized, while the backward pass is performed in FP32. Hyperparameter exploration is used to find proper finetuning hyperparameters.
- *Training*. A model is trained from scratch using a compute flow where all GeMMs in both forward and backward passes are quantized (see Figure 2. For mixed-precision training where the weights and activations use different data formats, the gradients (E_i in Figure 2) are quantized to the activation format.

Our benchmark suite contains two types of tasks: discriminative and generative.

4.3 Discriminative Inference

In this section, we examine inference results with MX formats across a variety of discriminative tasks including language translation, text encoding, image classification, speech recognition, and recommendation models. Table 2 summarizes the results related to **direct-cast inference**. Results for **finetuned inference** are reported in Table 4, and results for **PTQ with error diffusion inference** are reported in Table 3.

In these experiments, the same MX formats were used for both weights and activations following Algorithm 1. Round-half-to-nearest-even was used for conversion to MX formats. The results presented in Table 2 corroborates the effectiveness of MXINT8 as a drop-in replacement for FP32 with minimal accuracy drop. For MXFP8 and MXFP6, the general trend is that the variant of the format with more mantissa bits was better for direct-cast inference. With finetuned inference (Table 4), MXFP6_E2M3 is able to achieve close-to-parity with FP32.

More mantissa better for inference.

Table 2: Direct-cast inference with MX data formats. For each experiment, the FP32 baseline was quantized (both weights and activations) with no additional tweaks. MXINT8 is a compelling alternative to FP32 for low-friction direct-cast inference.

Task	Family	Model	Dataset	Metric	Baseline	MXINT8	MXFP8		MXFP6		MXFP4
					FP32		E4M3	E5M2	E2M3	E3M2	
Language Translation	Transformers (Enc-Dec)	Transformer-Base [9]	WMT-17	BLEU Score ↑	26.85	26.64	26.27	25.75	26.38	25.97	22.68
		Transformer-Large [9]			27.63	27.56	27.44	27.02	27.52	27.22	26.33
	LSTM	GNMT [10]	WMT-16		24.44	24.52	24.53	24.45	24.51	24.44	23.75
Language Encoding	Transformers (Enc-Only)	BERT-Base [11]	Wikipedia	F-1 Score ↑	88.63	88.58	88.47	87.04	88.38	88.05	84.94
		BERT-Large [11]			93.47	93.41	93.42	93.32	93.45	93.27	90.97
Image Classification	Vision Transformer	DeiT-Tiny [12]	ImageNet ILSVRC12	Top-1 Acc. ↑	72.16	72.20	71.37	70.11	71.56	70.16	56.72
		DeiT-Small [12]			80.54	80.56	79.83	79.00	80.11	79.04	71.35
	CNN	ResNet-18 [13]			70.79	70.80	69.08	66.16	69.71	66.10	48.77
		ResNet-50 [13]			77.40	77.27	75.94	73.78	76.42	73.75	42.39
		MobileNet v2 [14]			72.14	71.61	65.74	53.50	67.76	53.46	0.25
Speech Recognition	Transformer	Wav2Vec 2.0 [15]	LibriSpeech	WER ↓	18.90	18.83	23.71	21.99	20.63	21.98	42.62
Recommendations	MLPs	DLRM [16]	Criteo Terabyte	AUC ↑	0.803	0.803	0.802	0.801	0.802	0.801	0.7947

Table 3: Error diffusion for PTQ with MX data formats. Both activations and pre-trained weights from the baseline model are quantized to the column's datatype.

Task	Family	Model	Dataset	Metric	FP32	MXFP6		MXFP4
					Baseline	E2M3	E3M2	
Image Classification	Vision Transformer	DeiT-Tiny [12]	ImageNet ILSVRC12	Top-1 Acc. ↑	72.16	72.16	71.29	64.76
		DeiT-Small [12]			80.54	80.50	80.25	76.80
	CNN	ResNet-18 [13]			70.79	70.66	70.15	67.40
		ResNet-50 [13]			77.40	77.15	76.48	69.99
		MobileNet v2 [14]			72.14	70.22	65.32	18.88
Speech Recognition	Transformer	Wav2Vec 2.0 [15]	LibriSpeech	WER ↓	18.90	19.09	19.36	24.39

4.4 Generative Inference

We leveraged the open source LM Eval Harness by Eleuther AI for our evaluation of MX data formats in generative inference of OpenAI GPT3-175B and open source LLaMA-7B.[2] All benchmarks were run under zero-shot settings (i.e., no examples were presented to the models before evaluation). Our benchmark suite includes the following subset:

Lambada — Lambada is a long range prediction task, where the model must predict the last word in a long narrative passage. We used the version of lambada data used to evaluate GPT2 in LM Harness.

Wikitext — The wikitext task is based on the wikitext-2 dataset and requires the model to predict long sequences based on high quality Wikipedia articles. GPT3-175B was not evaluated on this task as Wikipedia data was part of its training corpus [17].

[2] https://github.com/EleutherAI/lm-evaluation-harness/tree/1736d78dd9615107e68ec7f74043b02d4ab68d12.

Table 4: Finetuned inference with MX data formats. Finetuning is performed for a few epochs starting from the FP32 model. Cells containing N/A means no finetuning was needed due to good direct-cast results. MXFP6_E2M3 achieves close-to-parity with FP32 after finetuning.

Task	Family	Model	Dataset	Metric	FP32 Baseline	MXFP6		MXFP4
						E2M3	E3M2	
Language Translation	Transformers (Enc-Dec)	Transformer-Base [9]	WMT-17	BLEU Score ↑	26.85	26.98	27.01	25.97
		Transformer-Large [9]			27.63	27.60	27.62	27.33
	LSTM	GNMT [10]	WMT-16		24.44	N/A	N/A	24.56
Image Classification	Vision Transformer	DeiT-Tiny [12]	ImageNet ILSVRC12	Top-1 Acc. ↑	72.16	72.09	70.86	66.41
		DeiT-Small [12]			80.54	80.43	79.76	77.61
	CNN	ResNet-18 [13]			70.79	70.6	69.85	67.19
		ResNet-50 [13]			77.40	77.27	76.54	74.86
		MobileNet v2 [14]			72.14	71.49	70.27	65.41
Speech Recognition	Transformer	Wav2Vec 2.0 [15]	LibriSpeech	WER ↓	18.90	N/A	21.46	29.64

ARC dataset — The Arc tasks are both multiple choice tasks consisting of nearly 8000 science exam questions, with the dataset split into easy and more challenging questions. The model is tasked with picking the correct answer from several options.

Hendryck's Test — Hendryck's test suite is a set of tasks that measure how knowledgeable a model is in 57 different fields. We used **computer science**, **international law**, and **jurisprudence** as a subset for this study. These tasks are all multiple choice questions, where the model must pick the correct answer from the options presented.

Table 5 and Table 6 show results for direct-cast inference on OpenAI GPT3-175B [17] and open source LLaMA-7B, respectively. Due to the size of these models, no quantization-aware finetuning was performed. The columns with a single MX format use that format for both weights and activations; the other columns list separate formats for weights (Wt) and activations (Act) and utilize mixed-precision.

MXINT8 matched baseline FP32 to within the standard deviation on all tasks for both GPT3-175B and LLaMA-7B. MXINT8 once again proves to be a compelling alternative to FP32 for low-friction direct-cast inference.

Table 5: GPT3-175B direct-cast inference results. Higher is better for all tasks. Each number is given ± the bootstrap estimated standard deviation. We only experiment with the higher mantissa width variant of each format (i.e., MXFP8_e4m3 and MXFP6_e2m3) given that the results in Section 5.2 show these variants works better for direct-cast inference.

Tasks	FP32	MXINT8	MXFP8	MXFP6	MXFP6 Wt MXFP8 Act	MXFP4 Wt MXFP8 Act	MXFP4 Wt MXFP6 Act	MXFP4
ARC easy ↑	0.744 ± 0.009	0.740 ± 0.009	0.738 ± 0.009	0.737 ± 0.009	0.740 ± 0.009	0.749 ± 0.009	0.744 ± 0.009	0.748 ± 0.010
ARC challenge ↑	0.480 ± 0.015	0.481 ± 0.015	0.485 ± 0.015	0.480 ± 0.015	0.478 ± 0.015	0.486 ± 0.015	0.487 ± 0.015	0.425 ± 0.014
Lambada ↑	0.755 ± 0.006	0.754 ± 0.006	0.708 ± 0.006	0.745 ± 0.006	0.725 ± 0.006	0.728 ± 0.007	0.754 ± 0.006	0.623 ± 0.007
College CS ↑	0.360 ± 0.049	0.340 ± 0.048	0.350 ± 0.048	0.350 ± 0.048	0.340 ± 0.048	0.340 ± 0.046	0.320 ± 0.047	0.240 ± 0.043
Int. law ↑	0.504 ± 0.046	0.537 ± 0.046	0.455 ± 0.046	0.521 ± 0.046	0.463 ± 0.046	0.331 ± 0.043	0.347 ± 0.043	0.298 ± 0.045
Jurisprudence ↑	0.454 ± 0.049	0.435 ± 0.048	0.491 ± 0.048	0.454 ± 0.048	0.472 ± 0.049	0.463 ± 0.048	0.418 ± 0.048	0.324 ± 0.045

Table 6: LLaMA-7B direct-cast inference results. Higher is better for all tasks except `wikitext`. For this benchmark only, the Softmax function was not quantized to Bfloat16.

Tasks	FP32	MXINT8	MXFP8	MXFP6	MXFP6 Wt MXFP8 Act	MXFP4 Wt MXFP8 Act	MXFP4 Wt MXFP6 Act	MXFP4
ARC easy ↑	0.729 ± 0.009	0.725 ± 0.009	0.716 ± 0.009	0.718 ± 0.009	0.726 ± 0.009	0.697 ± 0.010	0.696 ± 0.010	0.637 ± 0.010
ARC challenge ↑	0.447 ± 0.015	0.444 ± 0.015	0.430 ± 0.015	0.445 ± 0.015	0.442 ± 0.015	0.412 ± 0.014	0.406 ± 0.014	0.355 ± 0.014
Lambada ↑	0.736 ± 0.006	0.731 ± 0.006	0.720 ± 0.006	0.724 ± 0.006	0.721 ± 0.006	0.675 ± 0.006	0.678 ± 0.007	0.557 ± 0.007
College CS ↑	0.260 ± 0.044	0.220 ± 0.045	0.270 ± 0.042	0.240 ± 0.043	0.280 ± 0.045	0.240 ± 0.043	0.210 ± 0.041	0.220 ± 0.042
Int. law ↑	0.463 ± 0.046	0.430 ± 0.045	0.413 ± 0.045	0.422 ± 0.045	0.413 ± 0.045	0.398 ± 0.045	0.405 ± 0.045	0.331 ± 0.041
Jurisprudence ↑	0.361 ± 0.046	0.370 ± 0.047	0.380 ± 0.047	0.370 ± 0.046	0.352 ± 0.047	0.269 ± 0.045	0.296 ± 0.044	0.269 ± 0.043
wikitext ↓	9.488	9.504	9.768	9.628	9.683	11.476	11.147	27.201

4.5 Generative Training

Table 7 and Figure 3 show the language model loss obtained from training GPT-like models of various size (20M-1.5B) using MXFP6_e3m2 for both the forward and backward passes (see Figure 2). The training is done using the ADAM optimizer, with hyperparameters tuned for FP32. The same hyperparameters were reused for the MX format runs with no changes. All the models are trained to efficiency with number of steps calculated based on the scaling power-laws [18]. Round-half-away-from-zero rounding was used for conversion to MX formats.

The results in Table 7 and Figure 3 show that MXFP6_e3m2 is capable of delivering a model quality matching that of FP32 at much lower circuitry footprint. **MXFP6 provides the first demonstration of training generative language models to parity with FP32 using 6-bit weights, activations, and gradients with no modification to the training recipe.**

Pushing the limits even further, Table 8 and Figure 4 show the results from training the same GPT-like models, this time under a mixed-precision setting with MXFP4 weights and MXFP6_e3m2 activations. The gradients used the same data format as the activations. The training hyperparameters were the same as before. **Our results demonstrate that generative language models can be trained with MXFP4 weights and MXFP6 activations and gradients incurring only a minor penalty in the model loss.** This is once again with no modifications to the training recipe.

Model	FP32	MXFP6	
		E2M3	E3M2
GPT-20M	3.98	4.02	4.01
GPT-150M	3.30	3.33	3.32
GPT-300M	3.11	3.13	3.12
GPT-1.5B	2.74	2.75	2.75

Table 7: Language model loss for training from scratch using MXFP6_E3M2 for weights, activations, and gradients.

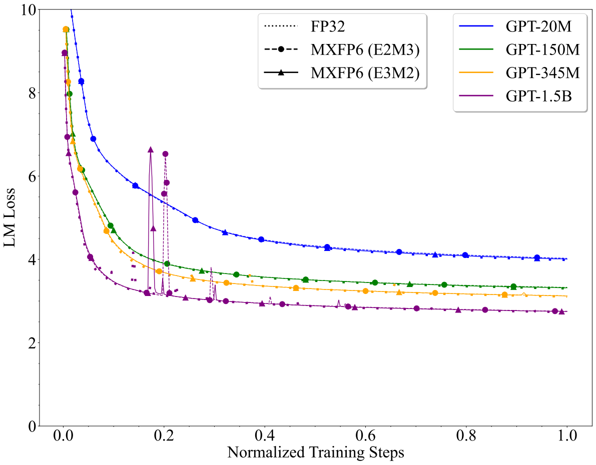


Figure 3: GPT training loss curve, using MXFP6_E3M2 for weights, activations, and gradients.

Model	FP32	MXFP4 Wt MXFP6 Act
GPT-20M	3.98	4.04
GPT-150M	3.30	3.33
GPT-300M	3.11	3.14
GPT-1.5B	2.74	2.76

Table 8: Language model loss for training from scratch using MXFP4 for weights and MXFP6_E3M2 for activations and gradients.

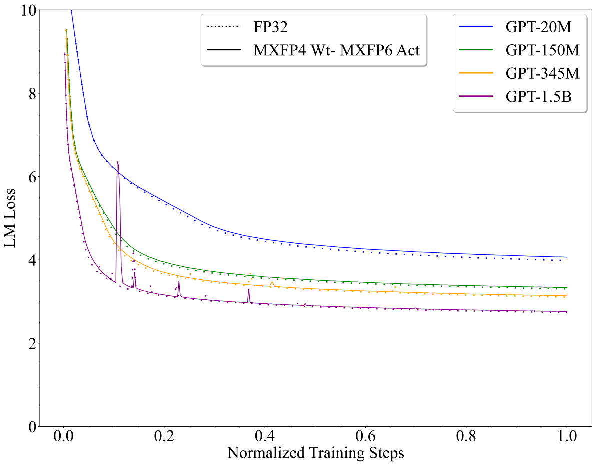


Figure 4: GPT mixed-precision training loss curve, using MXFP4 for weights and MXFP6_E3M2 for activations and gradients.

5 Conclusion

This paper evaluates MX data formats that integrate a block-level scale on top of narrow bit-width elements. The evaluated concrete MX formats provide compelling alternatives to FP32 training and inference with minimal user friction. Experimental results show the effectiveness of MX formats for a variety of deep learning models including generative language models, image classification, speech recognition, recommendation models, and translation.

In particular, MXINT8 is a compelling drop-in replacement to FP32 for low-friction direct-cast inference. MXFP6 closely matches FP32 for inference after quantization-aware finetuning. MXFP6 also, for the first time, enables generative language model training at sub-8-bit weights, activations, and gradients without sacrificing model accuracy or needing changes to the training recipe. Reducing the bit-width even further, we showcase training with MXFP4 weights and MXFP6 activations and gradients, incurring only a minor loss penalty for generative language models.

Direct Inference
MXINT8
QAT
MXFP6
↳ E2M3
Generative Training
MXFP6
E3M2
or
MXFP4 weight
MXFP6 grads &
E3M2 activations

Acknowledgment

The authors would like to thank the following individuals for their invaluable support and contributions: Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Yuan Yu, Rani Borkar, Saurabh Dighe, Brian Harry, Matt Perry, Renee L'Heureux, Dimitry Melts, Jasmine Klar, and Steve Scott.

References

[1] 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). *Open Compute Project*, 2023.

[2] Mario Drumond, Tao Lin, Martin Jaggi, and Babak Falsafi. Training DNNs with Hybrid Block Floating Point. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[3] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, Alessandro Forin, Haishan Zhu, Taesik Na, Prerak Patel, Shuai Che, Lok Chand Koppaka, XIA SONG, Subhojit Som, Kaustav Das, Saurabh T, Steve Reinhardt, Sitaram Lanka, Eric Chung, and Doug Burger. Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:10271–10281, 2020.

[4] Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. VS-Quant: Per-vector Scaled Quantization for Accurate Low-Precision Neural Network Inference. *Machine Learning and Systems (MLSys*, 3:873–884, 2021.

[5] Bita Darvish 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, and Eric Chung. With Shared Microexponents, A Little Shifting Goes a Long Way. *Int'l Symp. on Computer Architecture (ISCA)*, pages 1–13, 2023.

[6] Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Mathew Zhao, Ritchie amd 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.

[7] Microscaling PyTorch Library. 2023. URL `https://github.com/microsoft/microxcaling`.

[8] Jinjie Zhang, Yixuan Zhou, and Rayan Saab. Post-training quantization for neural networks with provable guarantees. *arXiv:2201.11113*, 2022.

[9] Transformer For PyTorch. URL `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Translation/Transformer`.

[10] GNMT v2 For PyTorch. URL `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Translation/GNMT`.

[11] NVIDIA/Megatron-LM: Ongoing research training transformer. URL `https://github.com/NVIDIA/Megatron-LM`.

[12] Data-Efficient architectures and training for Image classification. URL `https://github.com/facebookresearch/deit`.

[13] Convolutional Network for Image Classification in PyTorch. URL `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets`.

[14] Torchvision MobileNetV2. URL `https://github.com/pytorch/vision`.

[15] wav2vec 2.0. URL `https://github.com/facebookresearch/fairseq/tree/main/examples/wav2vec`.

[16] Deep Learning Recommendation Model for Personalization and Recommendation Systems. URL `https://github.com/facebookresearch/dlrm`.

[17] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

[18] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*, 2020.