S72778 Stable and scalable FP8 Deep Learning Training on Blackwell

Kirthi Shankar Sivamani, NVIDIA | March 20th, 2025

Agenda

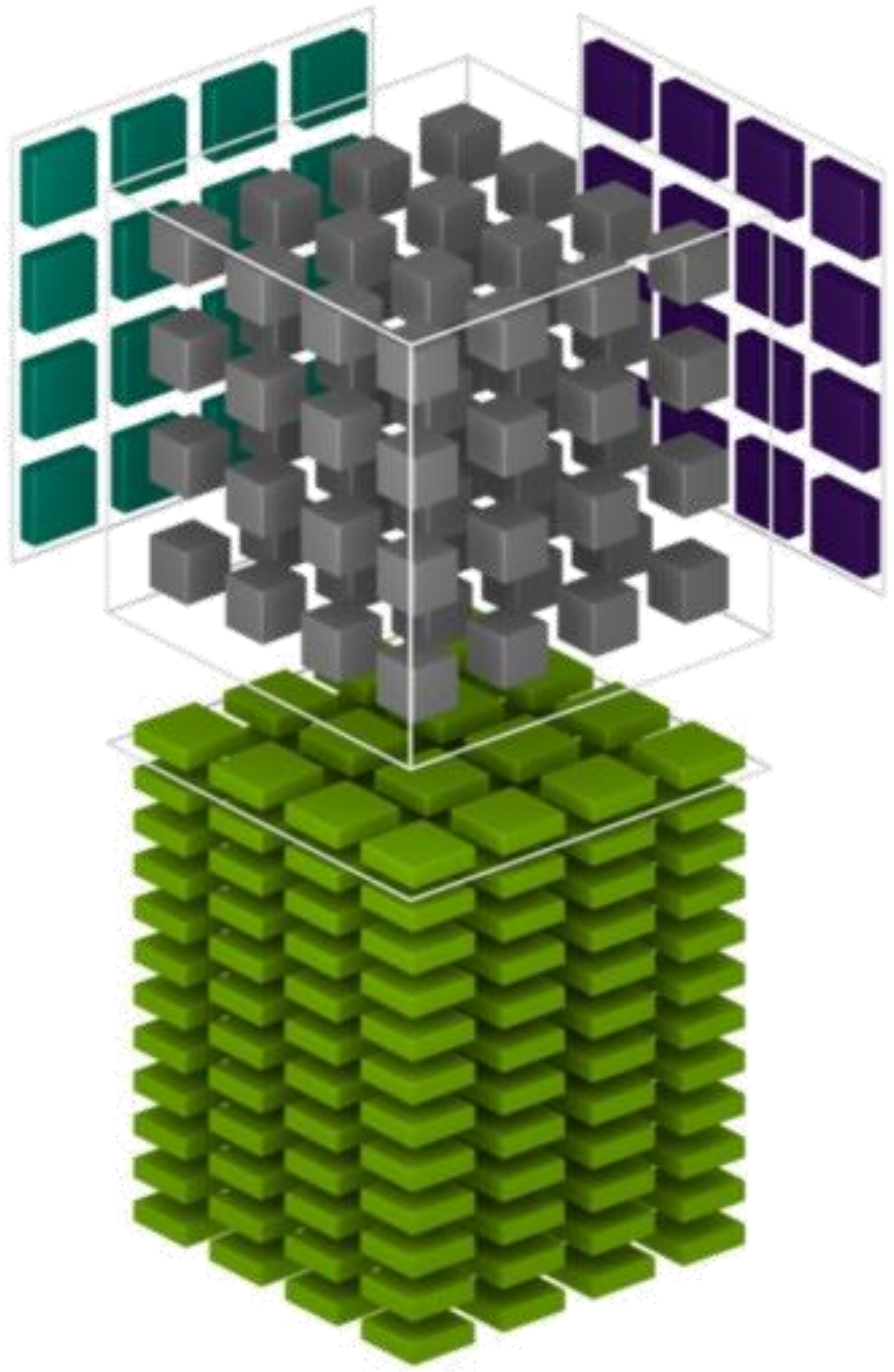
- FP8 and mixed precision primer
- MXFP8 Introduction
- Transformer Engine and newer recipes
- Results

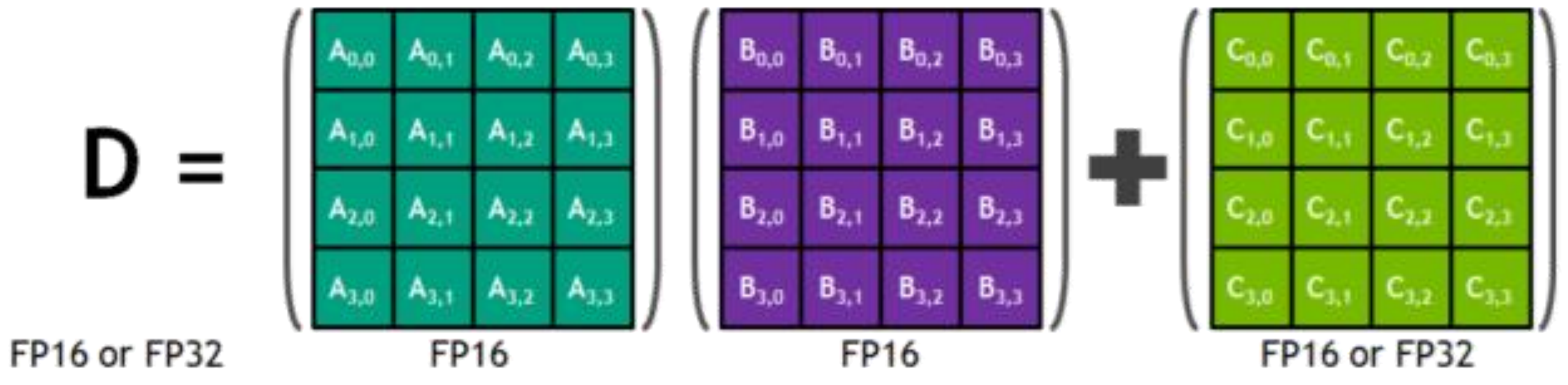
FP8 and mixed precision primer

TensorCores and Mixed Precision

- Starting with Volta, NVIDIA GPUs feature **TensorCores**
- They greatly speed up matrix multiplication and convolution
- To get the maximum performance, training in **mixed precision** is required.
- Hopper introduced FP8 Tensor Cores.

NVIDIA Device	TFLOPS
H100 TF32 Tensorcores	500
H100 FP16 Tensorcores	1000
H100 FP8 Tensorcores	2000
B100 FP8 Tensorcores	5000





Floating point and FP8 theory

FP32 = 0.3952

	sign	exponent	mantissa		
FP16	0	0 1 1 0 1	1 0 0 1 0 1 0 0 1 1	= 0.395264	More precision
BF16	0	0 1 1 1 1 1 0 1	1 0 0 1 0 1 0	= 0.394531	More range
FP8 E4M3	0	0 1 0 1	1 0 1	= 0.40625	More precision
FP8 E5M2	0	0 1 1 0 1	1 0	= 0.375	More range

E4M3 vs E5M2

	E4M3	E5M2
Exponent bias	7	15
Infinities	N/A	$S.11111.00_2$
NaN	$S.1111.111_2$	$S.11111.\{01, 10, 11\}_2$
Zeros	$S.0000.000_2$	$S.00000.00_2$
Max normal	$S.1111.110_2 = 1.75 * 2^8 = 448$	$S.11110.11_2 = 1.75 * 2^{15} = 57,344$
Min normal	$S.0001.000_2 = 2^{-6}$	$S.00001.00_2 = 2^{-14}$
Max subnorm	$S.0000.111_2 = 0.875 * 2^{-6}$	$S.00000.11_2 = 0.75 * 2^{-14}$
Min subnorm	$S.0000.001_2 = 2^{-9}$	$S.00000.01_2 = 2^{-16}$

FP16 Mixed Precision Recipe

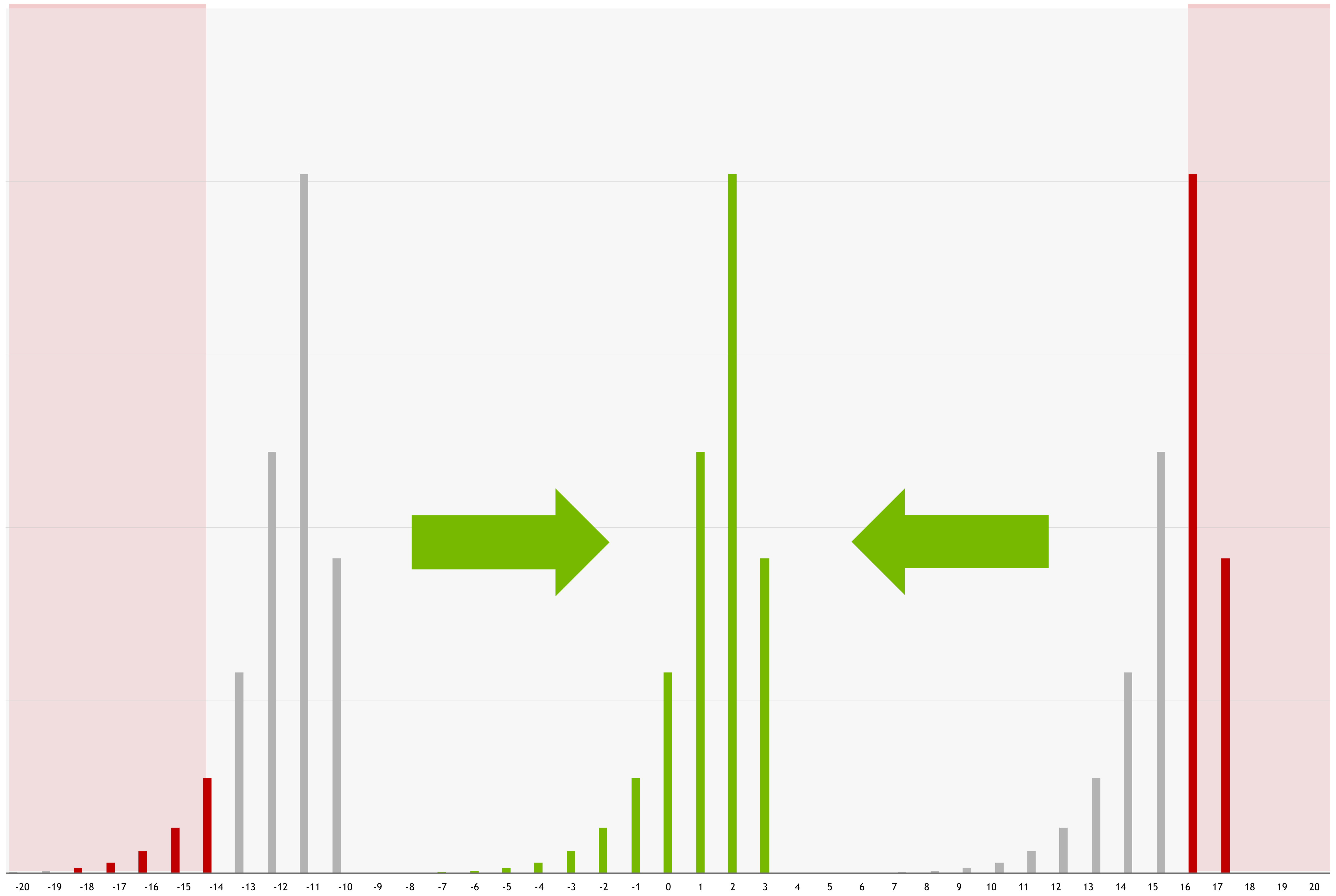
- Partition the DL network graph into safe and unsafe regions
  - Safe regions contain operations benefitting from reduced precision and whose outputs' dynamic ranges are similar to the inputs
- Use the scaling factor during the backward pass
  - Scaling factor is used to avoid over- and underflows in the value distribution of the tensors

FP16 Mixed Precision Recipe

- Partition the DL network graph into safe and unsafe regions
  - Safe regions contain operations benefitting from reduced precision and whose outputs' dynamic ranges are similar to the inputs
- Use the scaling factor during the backward pass
  - Scaling factor is used to avoid over- and underflows in the value distribution of the tensors

FP16 Mixed Precision Recipe

- Partition the DL network graph into safe and unsafe regions
  - Safe regions contain operations benefitting from reduced precision and whose outputs' dynamic ranges are similar to the inputs
- Use the scaling factor during the backward pass
  - Scaling factor is used to avoid over- and underflows in the value distribution of the tensors

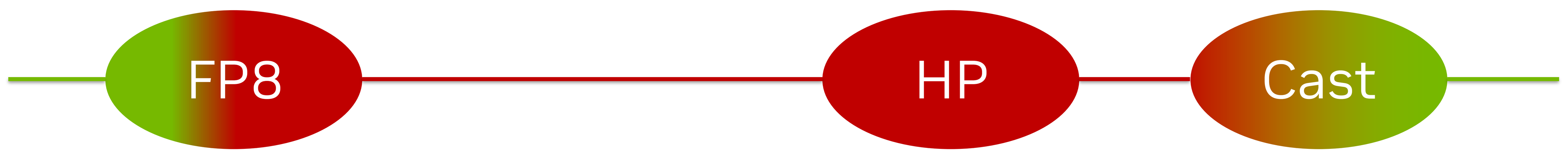


FP8 Delayed Scaling Recipe

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough

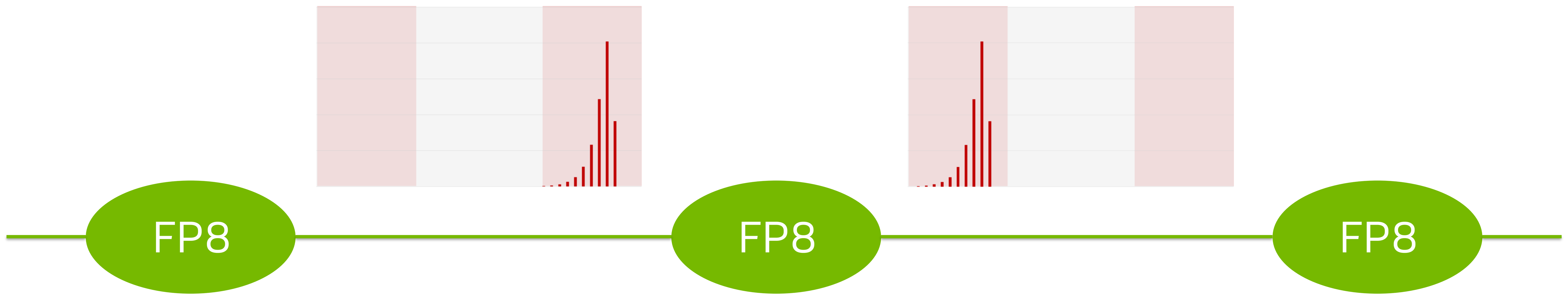
FP8 Delayed Scaling Recipe

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough



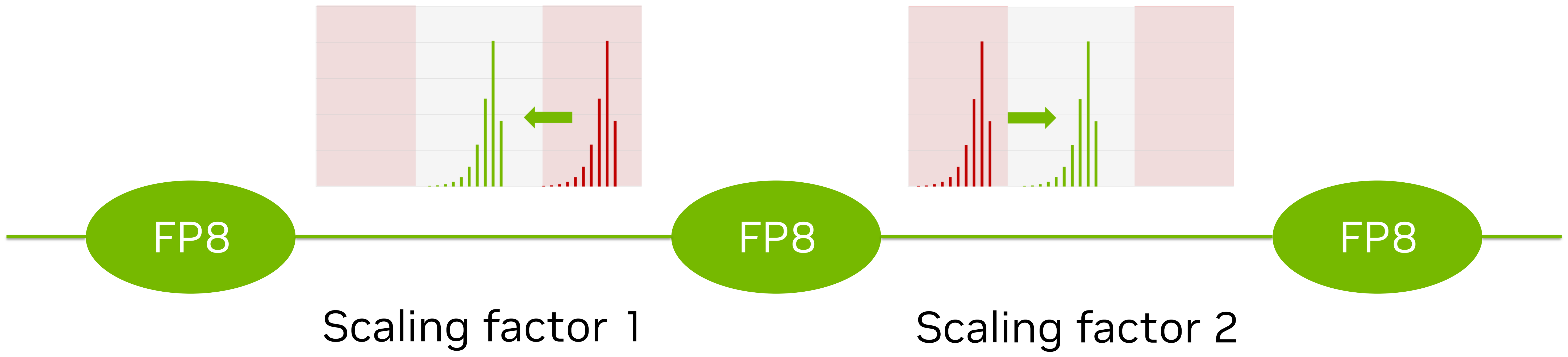
FP8 Delayed Scaling Recipe

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough



FP8 Delayed Scaling Recipe

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough



MXFP8 Introduction

MX Formats

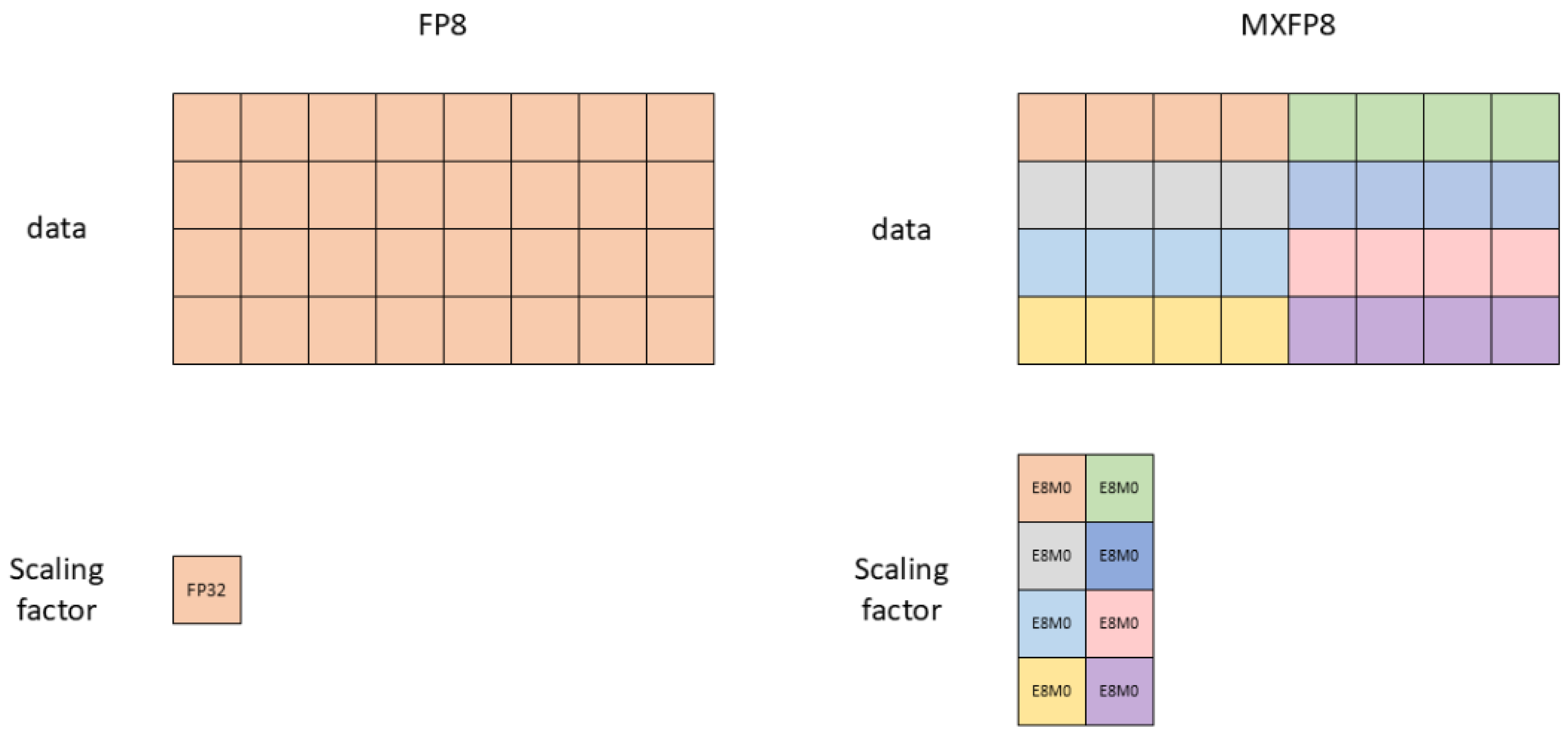
- OCP introduced and formalized microscaled (MX) formats for floating point numbers:
  - https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf

- Unlike regular floating point and FP8 formats which involve a single scale per tensor, MX formats have multiple scaling factors per tensor are compliant formats are characterized by 3 elements:
  - Element dtype and encoding
  - Scale dtype and encoding
  - Scaling block size

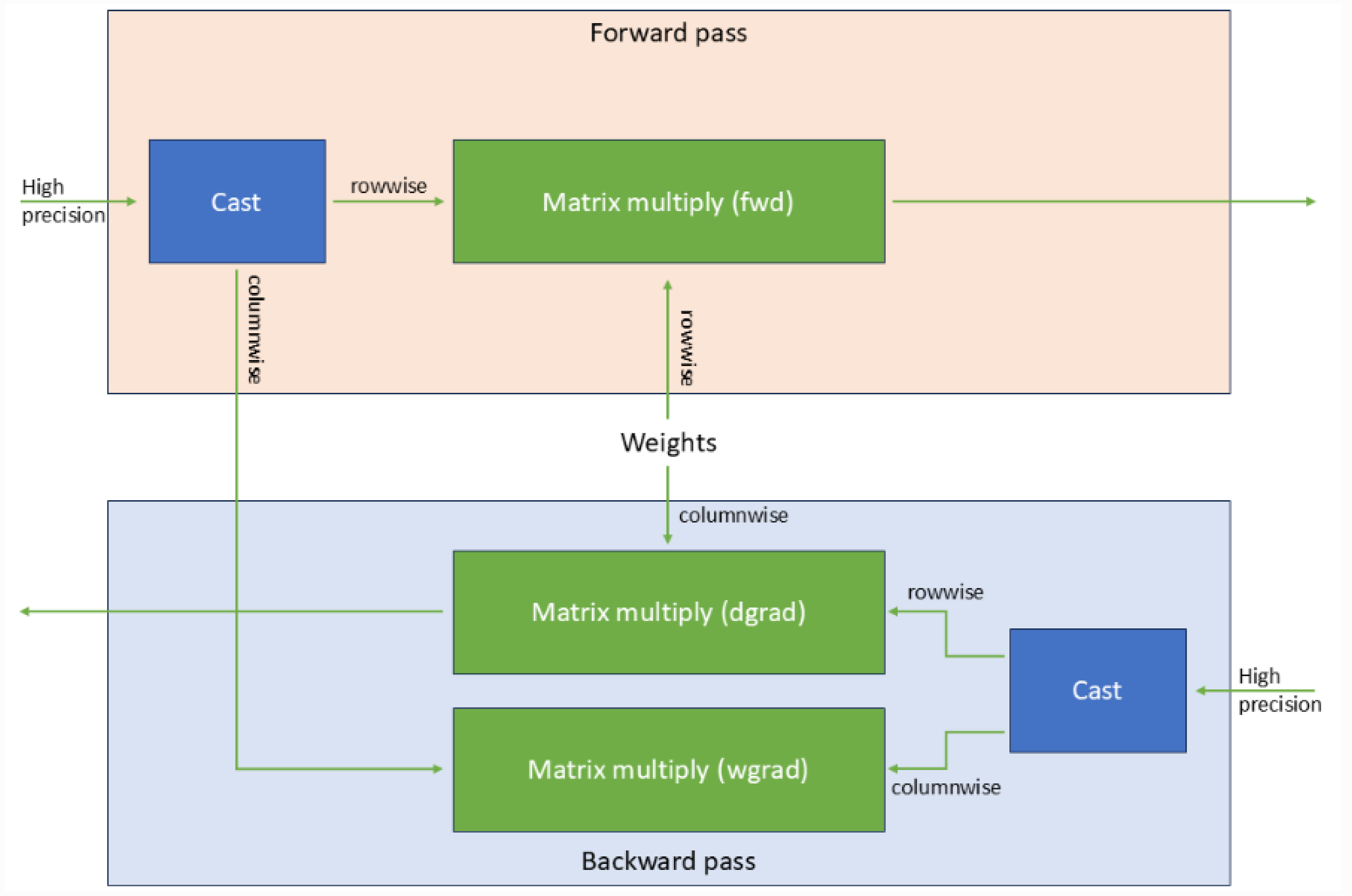
- For MXFP8:
  - Element dtype: E4M3 or E5M2
  - Scale dtype = E8M0
  - Block size = 32

For more details: GTC S72458 - Blackwell Numerics for AI

MXFP8


FP8
MXFP8
data
data
Scaling factor
FP32
Scaling factor
E8M0
E8M0
E8M0
E8M0
E8M0
E8M0
E8M0
E8M0

Recipe details



Transformer Engine

Transformer Engine intro

- An open-source library implementing the FP8 recipe for Transformer building blocks.
- Optimized for FP8 and other datatypes
- PyTorch and Jax are supported frameworks.
- Composable with the native framework operators.
- Supports different types of model parallelism
  - DP, TP, PP, CP
- https://github.com/NVIDIA/TransformerEngine
- Docs:
  - https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/index.html

MXFP8 with Transformer Engine

```python
import torch
import transformer_engine.pytorch as te
from transformer_engine.common import recipe

# Set dimensions.
in_features = 768
out_features = 3072
hidden_size = 2048

# Initialize model and inputs.
model = te.Linear(in_features, out_features, bias=True)
inp = torch.randn(hidden_size, in_features, device="cuda")

# Create MXFP8 recipe.
fp8_recipe = recipe.MXFP8BlockScaling()

# Enable autocasting to FP8.
with te.fp8_autocast(enabled=True, fp8_recipe=fp8_recipe):
    out = model(inp)

# Calculate loss and gradients.
loss = out.sum()
loss.backward()
```

```python
import torch
import transformer_engine.pytorch as te
from transformer_engine.common import recipe

# Set dimensions.
batch_size = 2
hidden_size = 768
seq_len = 2048
nheads = 12

# Create MXFP8 recipe.
fp8_recipe = recipe.MXFP8BlockScaling()

# Initialize model and inputs.
with te.fp8_model_init(recipe=fp8_recipe):
    model = te.TransformerLayer(hidden_size, 4*hidden_size,
                                num_attention_heads=nheads,
                                fuse_qkv_params=True)
inp = torch.randn(batch_size, seq_len, hidden_size, device="cuda")

# Enable autocasting to FP8.
with te.fp8_autocast(enabled=True, fp8_recipe=fp8_recipe):
    out = model(inp)

# Calculate loss and gradients.
loss = out.sum()
loss.backward()
```

Alternate recipes and results

FP8 Recipes

Sub-Tensor scaling Recipes

Pros

- FP8 format might not always cover the dynamic range of the entire tensor. It's a natural direction to explore sub-tensor (fine-granularity) scaling, so FP8 could cover the dynamic range in block level.
- Sub-Tensor scaling also allows to use E4M3 format for dY with more precision (only for 1x128 block)

Cons

- Sub-Tensor Scaling comes with a cost of GEMM performance

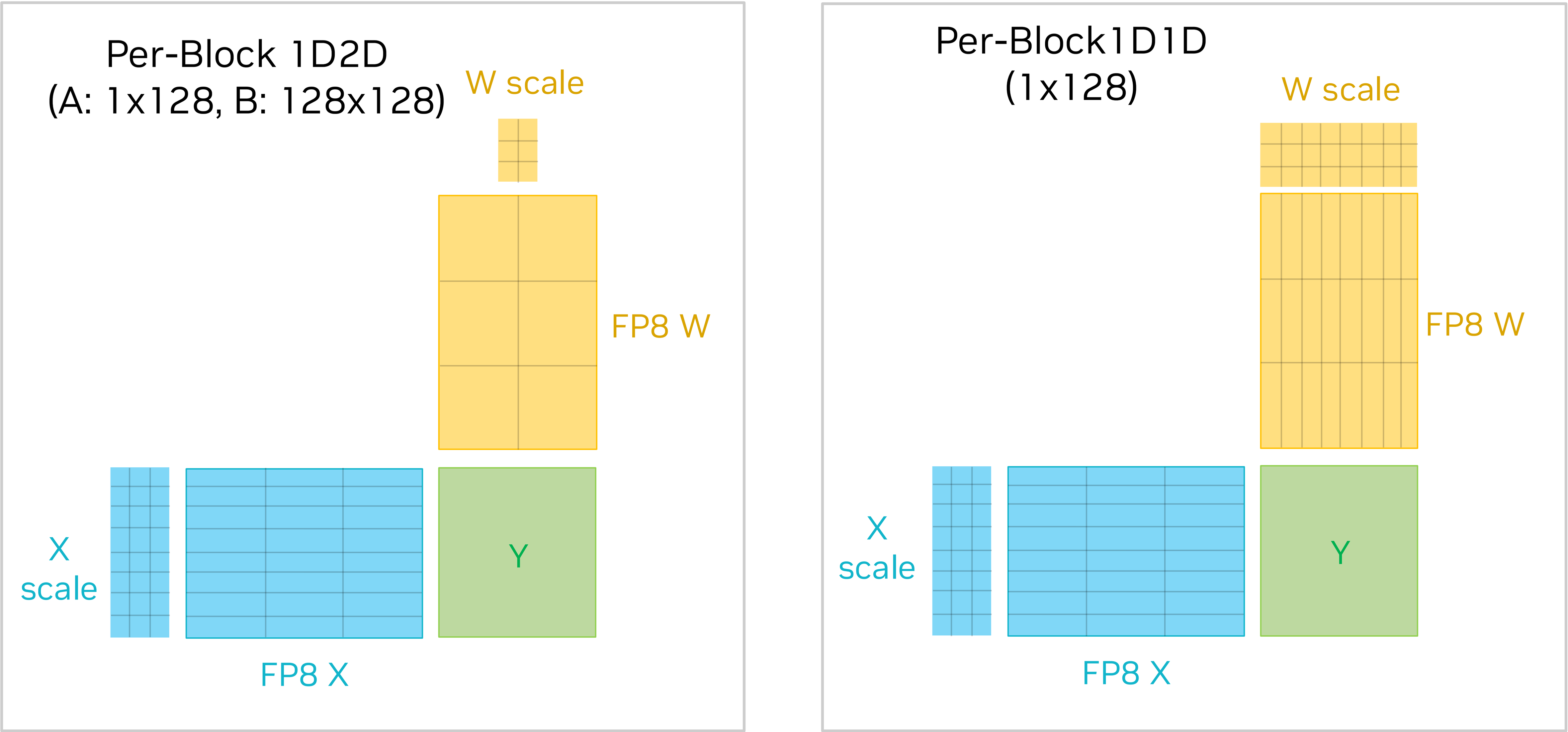
Sub-channel Recipe

Deepseek v3

- FP8 subchannel recipe
  - MoE model with 671B total parameters (37B activated parameters) 15T tokens, performance comparable to GPT-4o

Sub-channel

- **Scaling granularities**.
  - 1x128 for X, dY
  - 128x128 for W
- **GEMM**
  - 1D2D: Fprop Dgrad. 1D1D: Wgrad
- **FP8 dtype**. E4M3 everywhere.
- **Scale factor precision**. FP32 everywhere except for:
  - e8 in X, X^T in fc1, proj
  - e8 in dy, dy^T in fc2
- 1st and last decoder blocks in BF16

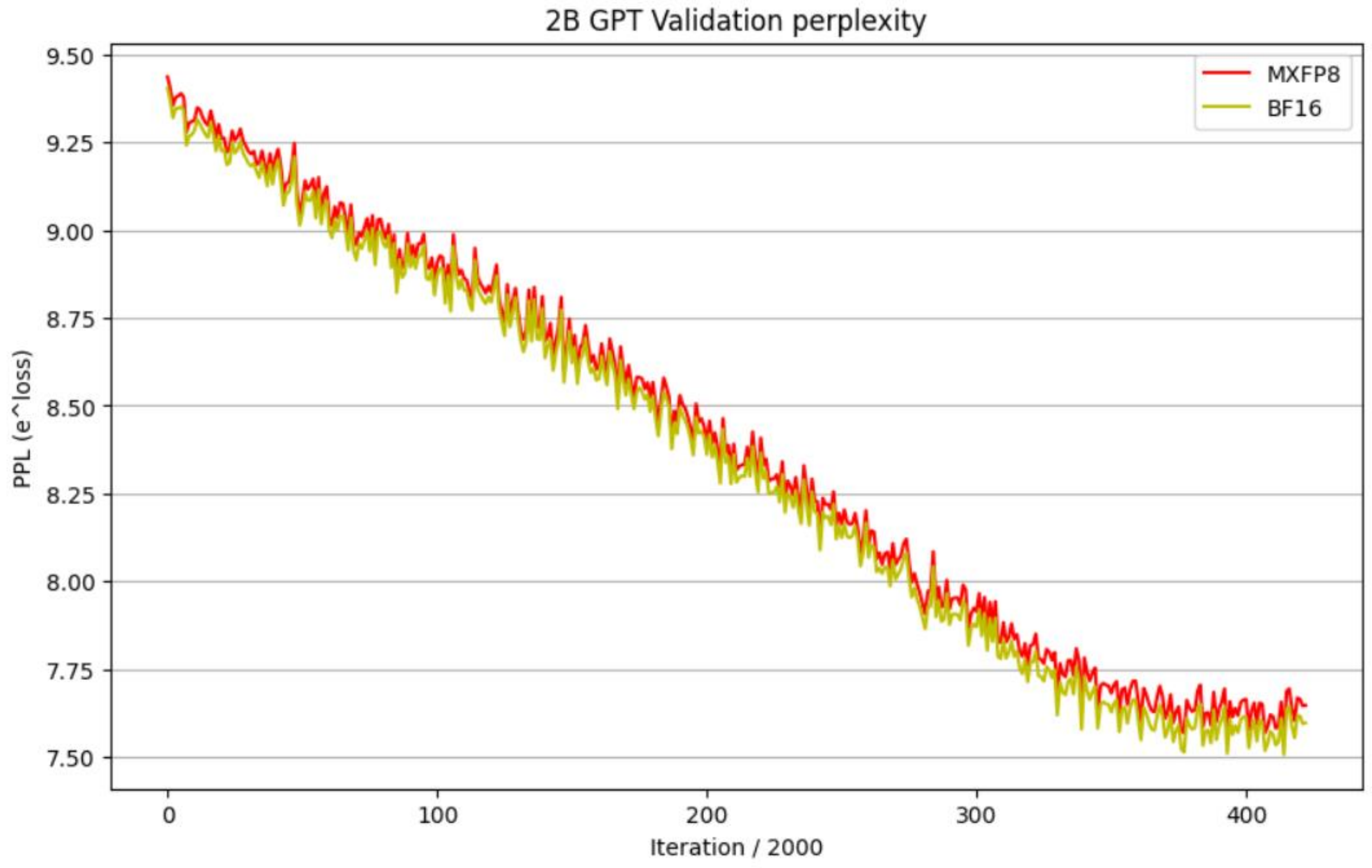


Results

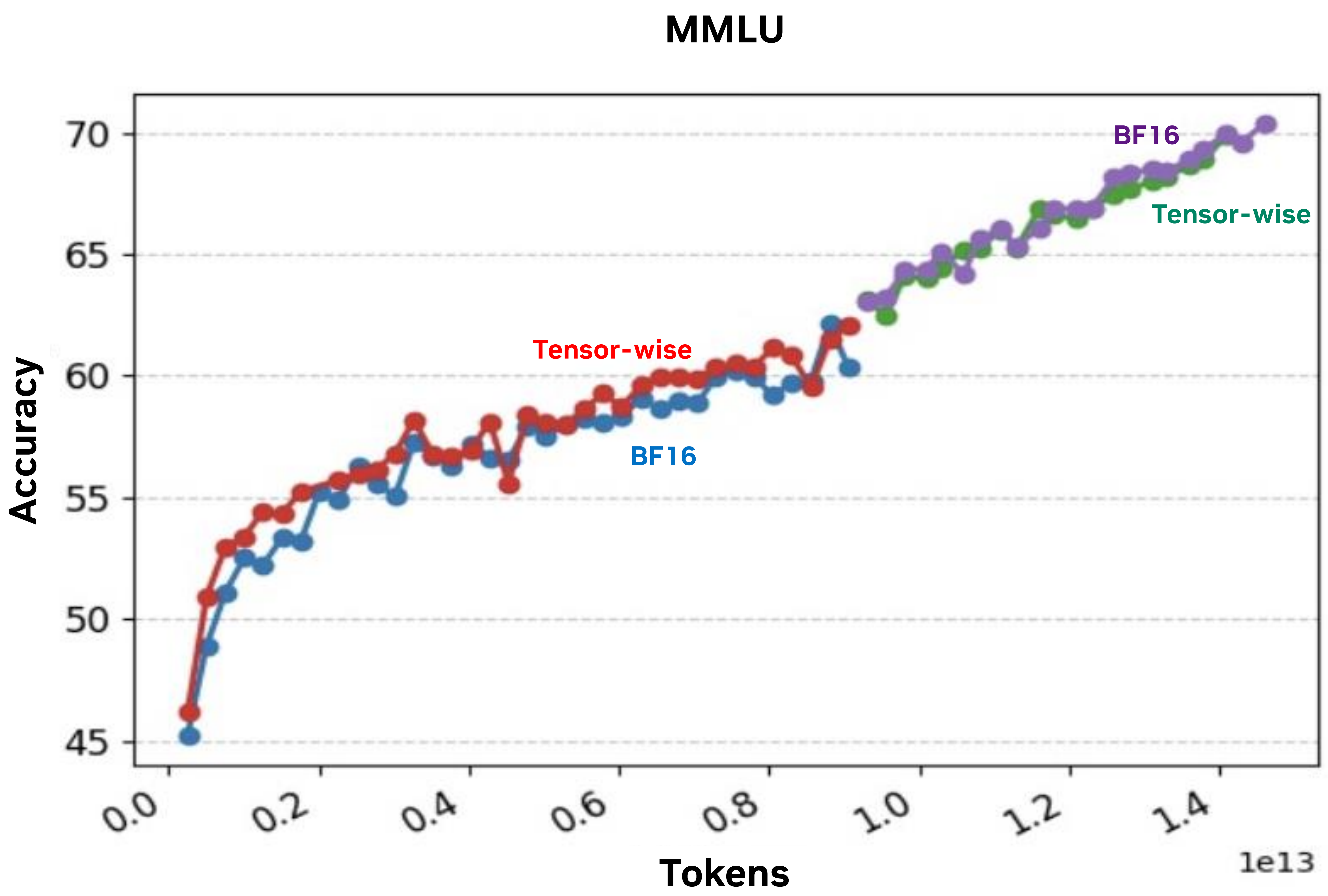
- MXFP8 on blackwell
  - Fully trained GPT 843M and 2B on B200 using MXFP8 recipe.
  - Fully trained Nemotron 56B using current scaling recipe on H100.
  - Downstream tasks on MMLU, Average Reasoning, Average Code, etc. perform on-par with BF16 baseline.
- Current scaling on Hopper
  - Per-Tensor Current Scaling with 1st+last decoder block in BF16
  - Nemotron 4: 8B size, 15T tokens
  - Nemotron 4: 25B size, 3T tokens

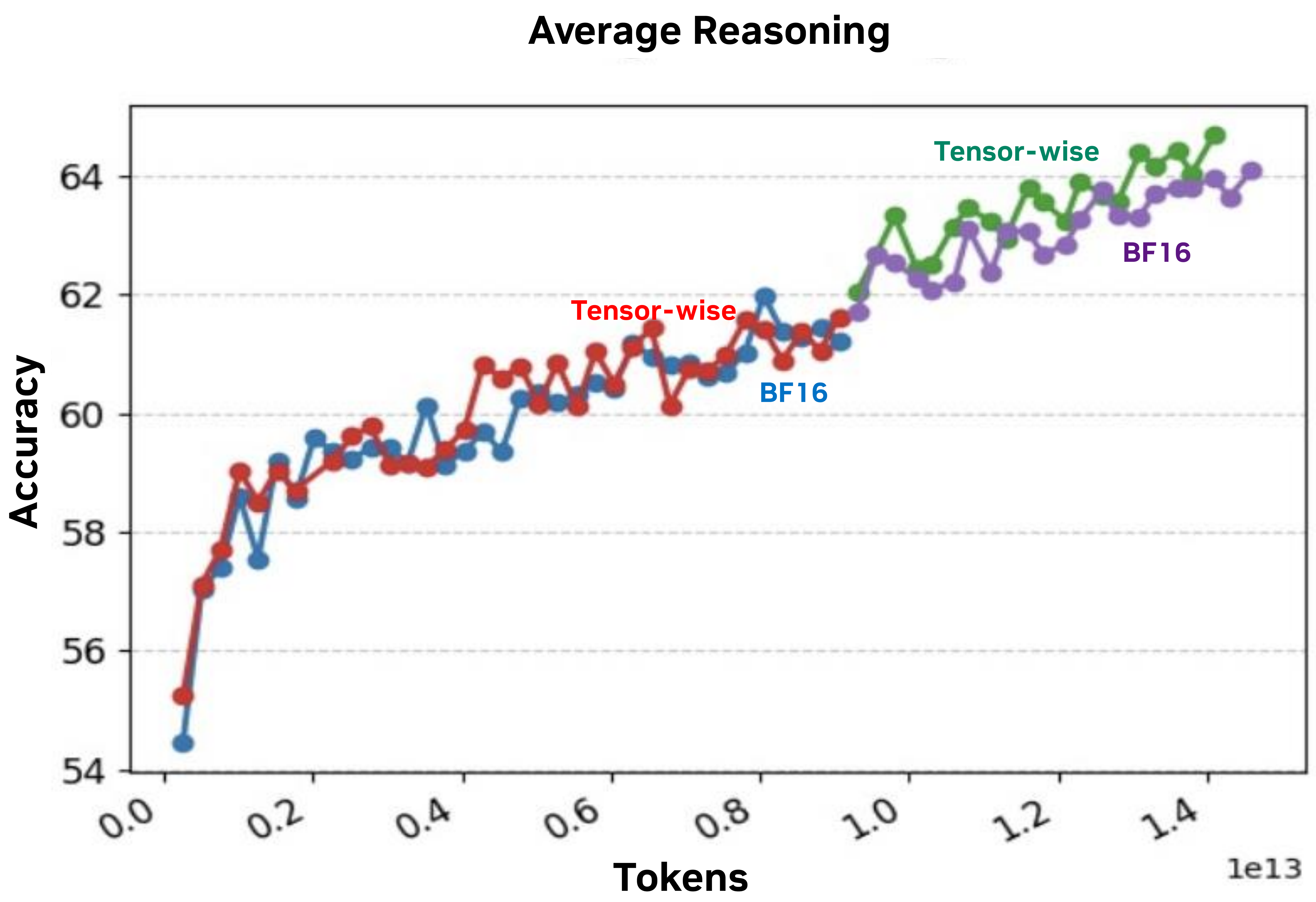
GPT 2B


2B GPT Training Loss
MXFP8
BF16
LM-Loss
2.6
2.5
2.4
2.3
2.2
2.1
2.0
0
2000
4000
6000
8000
10000
Iteration / 100


2B GPT Validation perplexity
MXFP8
BF16
PPL (e^loss)
9.50
9.25
9.00
8.75
8.50
8.25
8.00
7.75
7.50
0
100
200
300
400
Iteration / 2000

Nemotron 4 (8B size, 15T tokens)

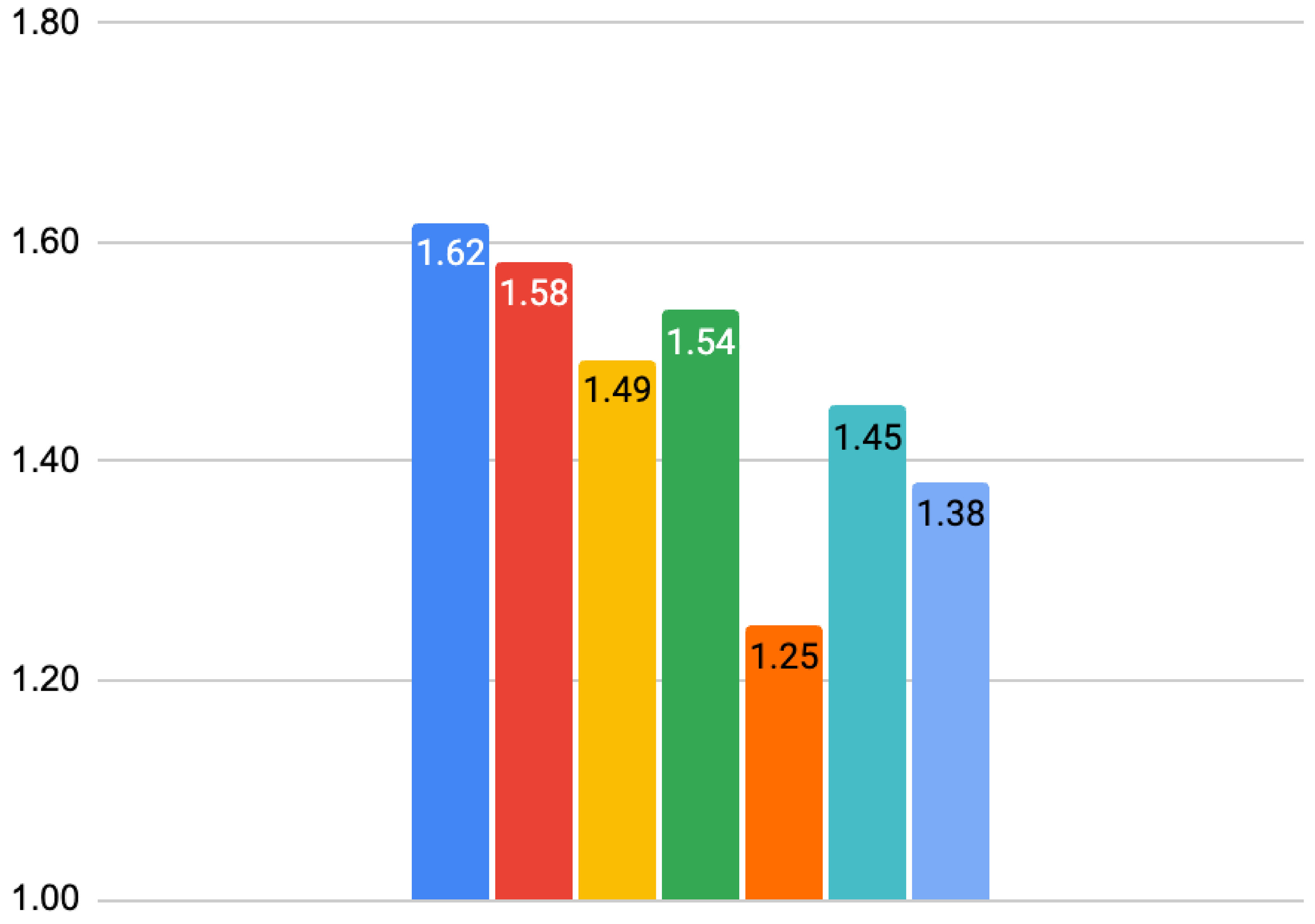




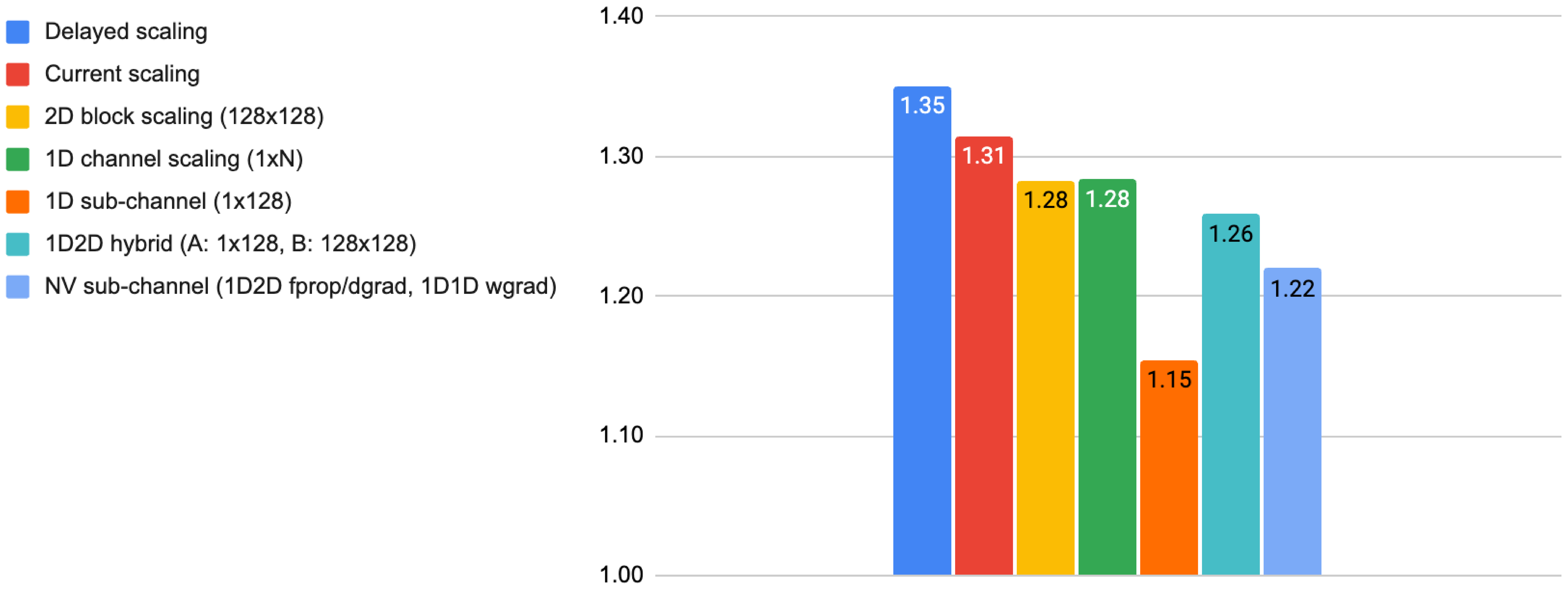
Projected Speedup

E2E Step time speedup compared to BF16

Speedup compared to BF16 (GPT3/175B)



Speedup compared to BF16 (Nemotron 8B)

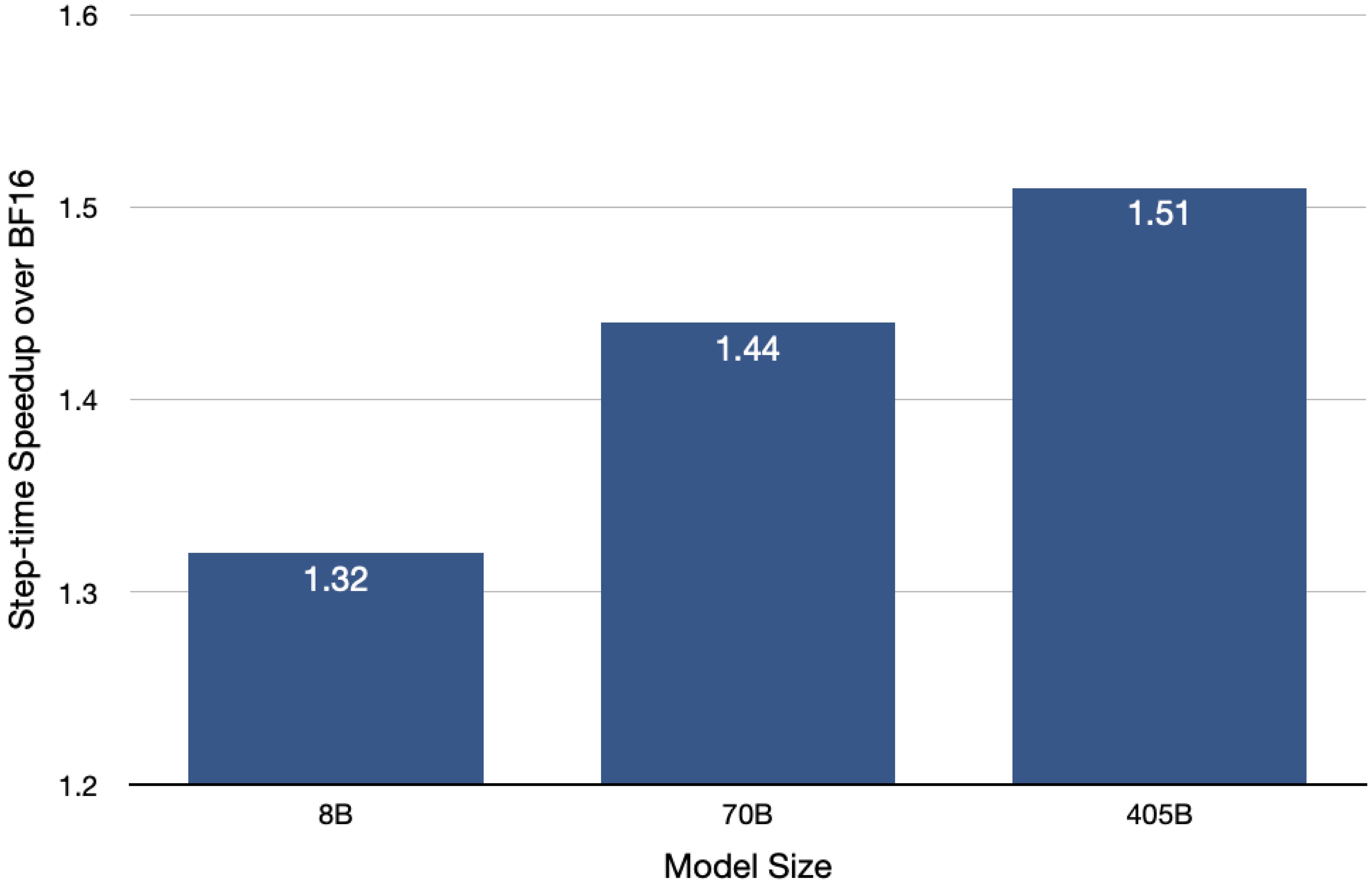


Methodology (Measurements with projections)

- Time breakdown from measured DS run on H100
- Apply projected overhead for each recipe (e.g., GEMM, quantization, communication)

Measured speedups

Tensor-wise dynamic scaling
Llama 3.1 on H100



Model	# GPUs	DP	MBS	GA	GBS
8B	8	4	1	32	128
70B	64	2	1	64	128
405B	576	4	1	63	252

Parallelism Configuration

Upcoming in Transformer Engine

What's upcoming in Transformer Engine

- Per Tensor current scaling.
  - https://github.com/NVIDIA/TransformerEngine/pull/1471
- Expanding MXFP8 support:
  - Attention
  - TP overlap
  - Mixture of experts
- NVFP4 support for inference.

NVIDIA