verda

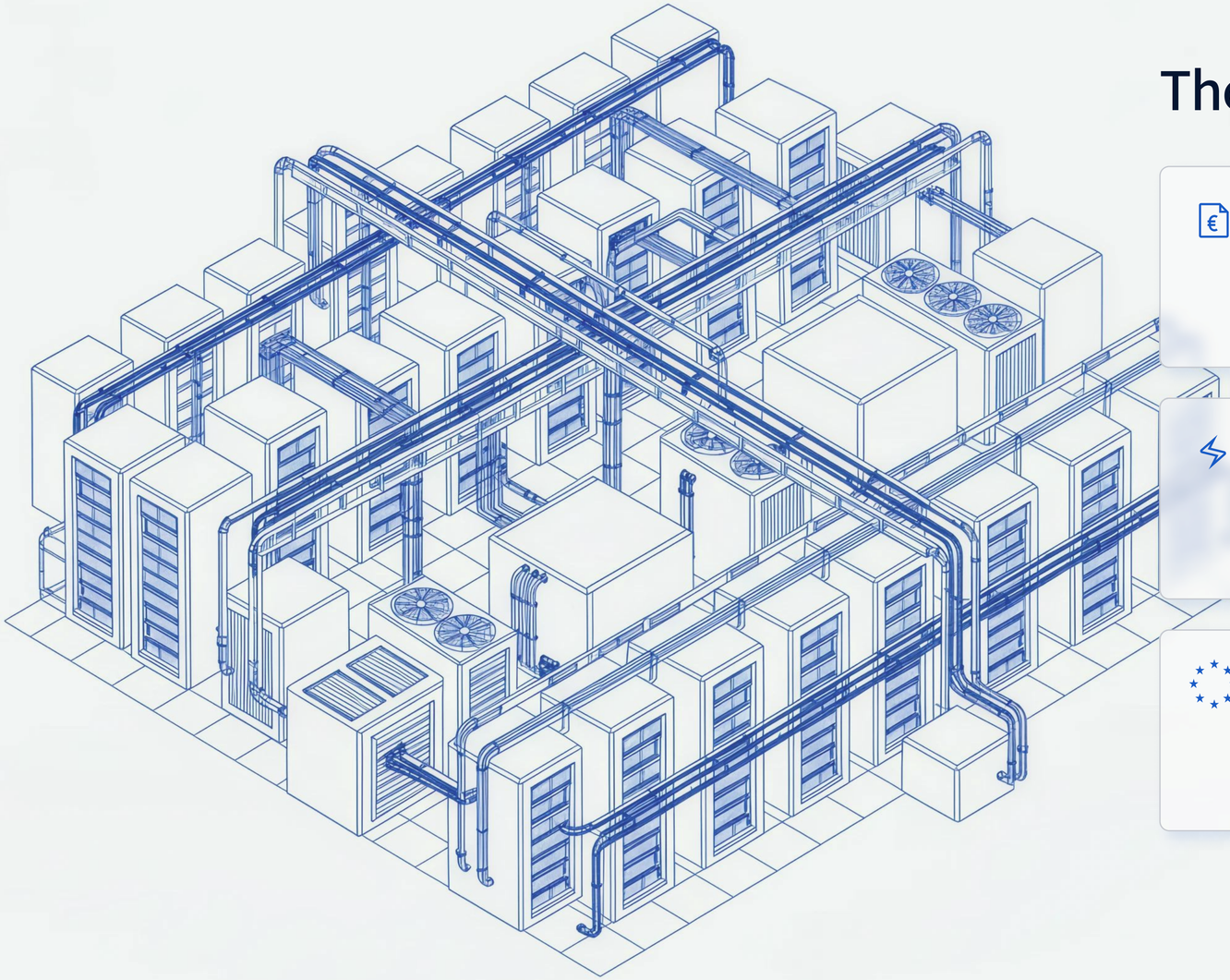
MLOps Meet-up Helsinki

Unlocking NVFP4: Low Precision Numerics on NVIDIA Blackwell

Speaker:

Riccardo Mereu
ML Engineer

Hyperscalers were built for CPU - AI cloud is different



The AI Cloud

Compute cost matters

Legacy cloud was built around saving developer time.
AI cloud optimizes for performance and compute costs.

GPU Data Centers need power

Data center build out starts from zero as energy density, power supply and cooling innovations make old footprint obsolete

Sovereignty is critical

Geopolitical shift towards national sovereignty especially in Europe requires an AI-native Hyperscaler built in Europe

Our AI-Native cloud beats hyperscalers for AI workloads

Clean modern architecture

Redesigned from scratch for GPU workloads, beating legacy piecemeal cloud architecture

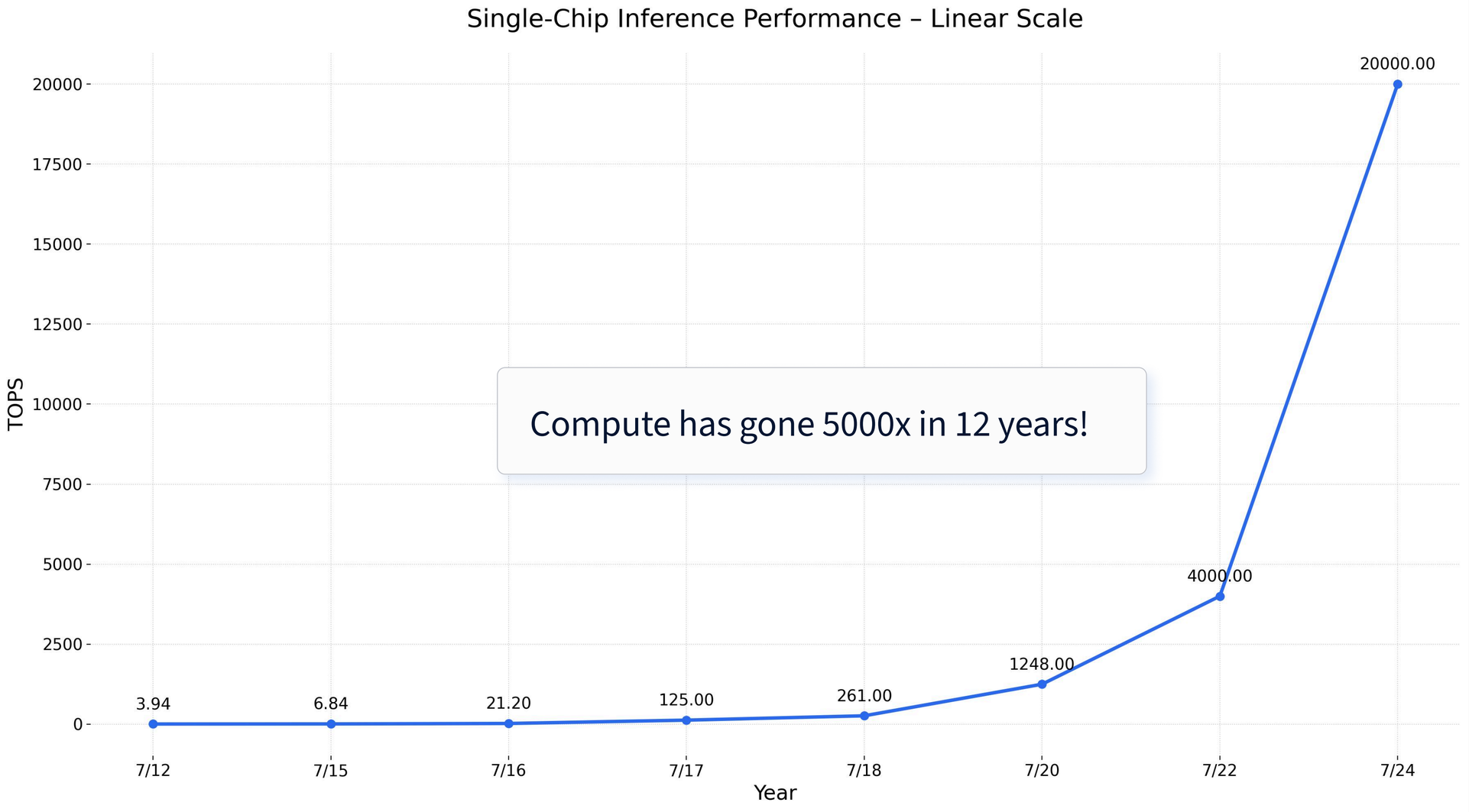
Unified control plane

One cohesive compute layer, beating legacy fragmented layer system on utilisation and developer experience

Deep AI framework integration

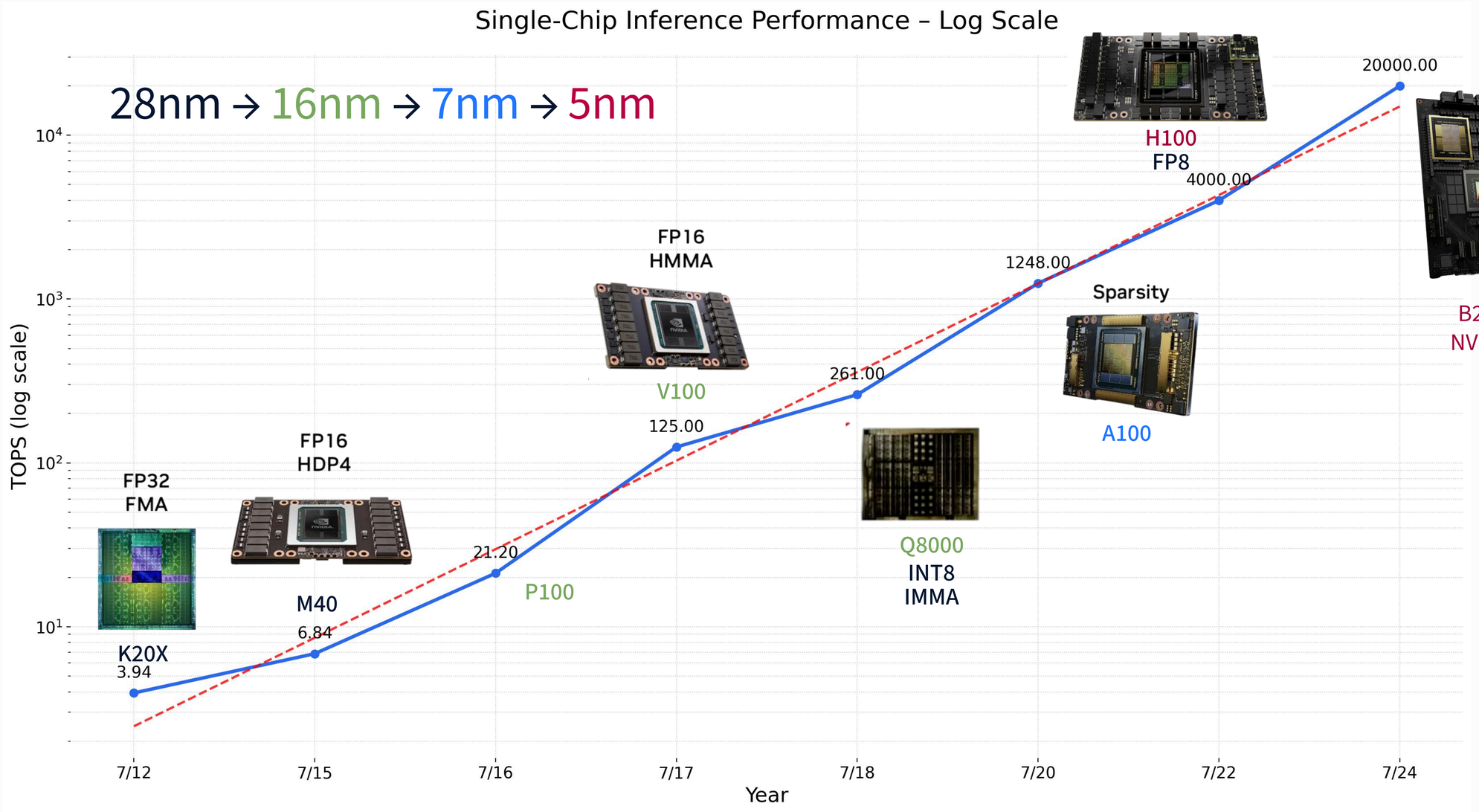
Natively optimised for PyTorch, JAX and more, enabling faster adoption of new AI tooling

Why low precision matters for a cloud company?



Source: Hardware for Deep Learning, Bill Dally

How did compute go 5000x?



GB300 NVL72

72 Blackwell Ultra GPUs in a coherent domain

→ **Coming soon to Verda**

Source: Trends in Deep Learning Hardware, Bill Dally

Where does the increases come from?

Number Representation

FP32 → BF16 → FP8 → FP4 — 32x (64x w/ sparsity)

Amortised Instructions (Tensor cores)

FMA → HMMA → WGMMA → UMMA — 12.5x

Process (fab size)

28nm → 16nm → 7nm → 5nm — ~2.5x (more transistors per mm^2)

HW-SW co-design

FA, FA2, FA3, FA4

verda

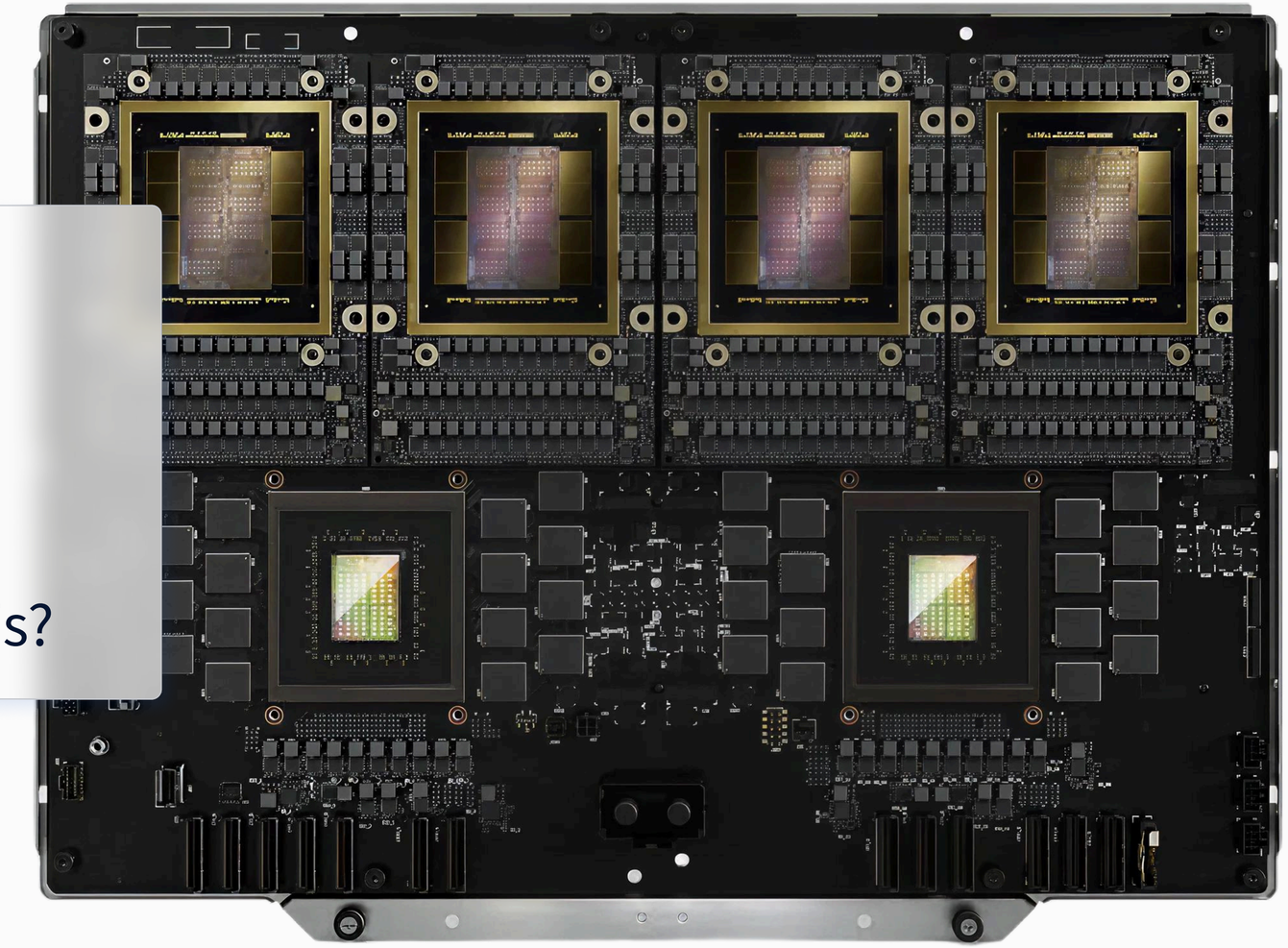
Section

Low Precision Numerics

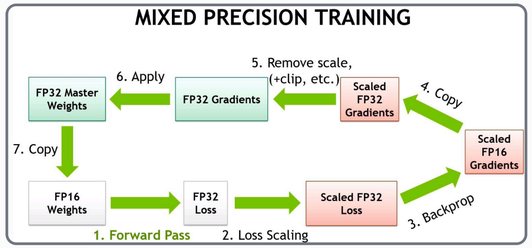
Low Precision Numerics on NVIDIA Blackwell

Outline

1. Why do we need Low Precision?
2. What is NVFP4?
3. How NVIDIA Blackwell enable this?



Timeline: Floating Point in AI



NVIDIA
Pre-Volta Arch.

2012–2015

Early DL models
(DanNet, AlexNet, ...)
CUDA - FP32

Caffe/Theano/TF/
NVIDIA Apex
FP16

2016

NVIDIA
Tesla V100

2017

Google *TPUv2*
BF16

NVIDIA
Ampere A100
(native BF16)

2020

NVIDIA
Hopper HX00 - **FP8**

2022

OPEN Compute Project

OCP Microscaling Formats (MX) Specification

Version 1.0

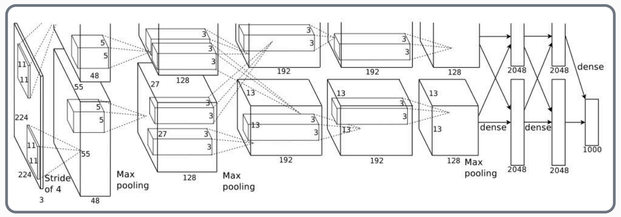
OCP Microscaling
MXFP6 - MXFP4

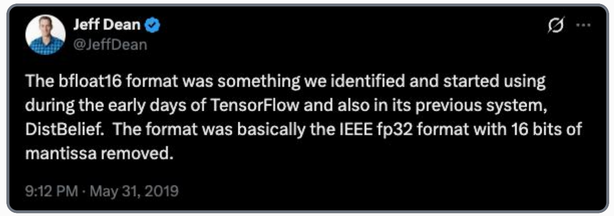
2023

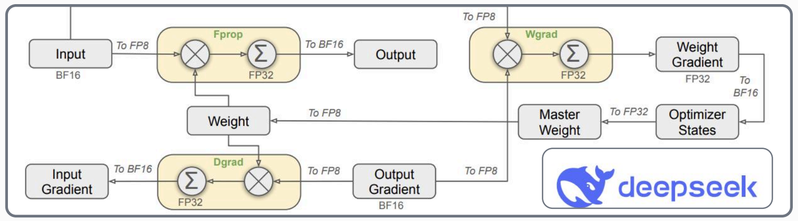
NVIDIA
Blackwell
(BX00 and GBX00)
NVFP4

2024

DeepSeek
(and frontier labs)
FP8 QAT







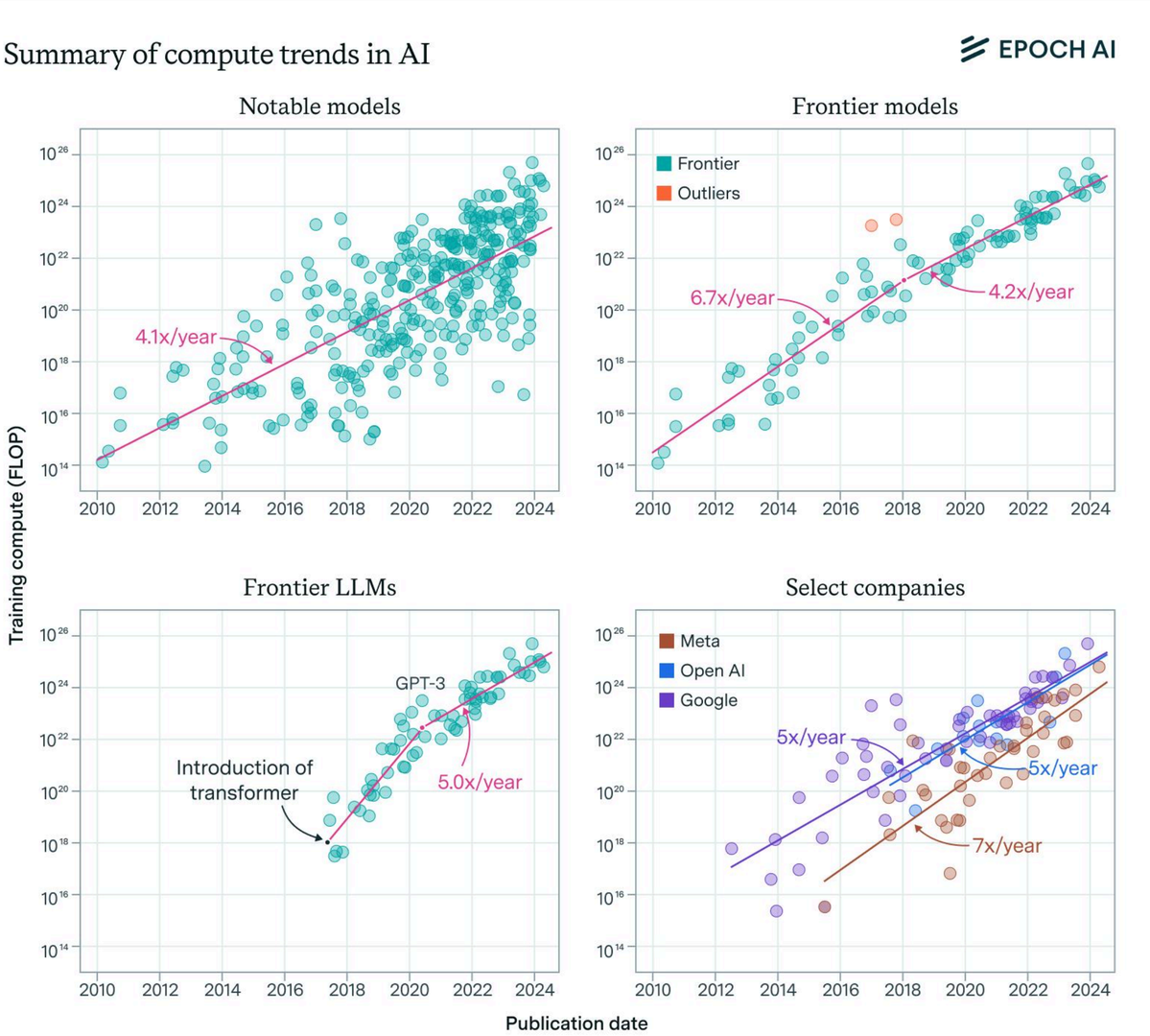
Why FP4 Training?

Training compute for frontier LLMs is growing at **4-5x/year**, demanding:

- smaller data formats
- high throughput

HW evolution:

- FP8 (4th gen tensor cores): ~1.5-2x throughput vs. FP16
- FP4 (5th gen tensor cores): 4x throughput vs. FP16

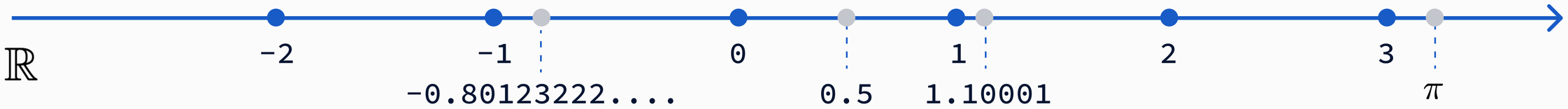


Source: EpochAI

Floating Point Basics

Real Number Line:

- Arbitrarily large and small values → **Infinite range**
- Also between any pair of numbers → **Infinite precision**



We cannot do infinite:

- Finite memory
- Finite silicon area (or time to complete operations)

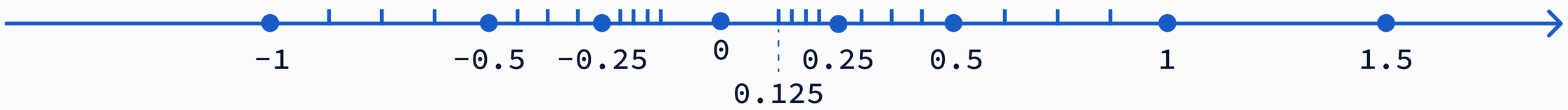
Need for number representation to sample the real line:

- Integer variants
- Floating Point

Floating Point Basics

FP Line:

- FP samples the real number line with equal number of samples between each pair of adjacent powers of 2
- *Example: E5M10 has 1024 samples between 0.25 and 0.5, between 128 and 256*



Bit Fields (ExMy):

- **Sign:** 1 bit (0 positive, 1 negative)
- **Exponent:** E bits, which power of 2 is sampled → *dynamic range*
- **Mantissa:** M bits, samples between powers of two → *precision*

$$N_{10} = (-1)^S \times 1.M_{10} \times 2^{E_{10} - \text{bias}}$$

Exponent bias = $2^{E_N - 1} - 1$

Floating Point Basics

FP16 Example (E5M10)

- **Sign:** 1 bit (0 positive, 1 negative)
- **Exponent:** 5 bits
- **Mantissa:** 10 bits
- **Exponent bias:** $2^{5-1} - 1 = 15$

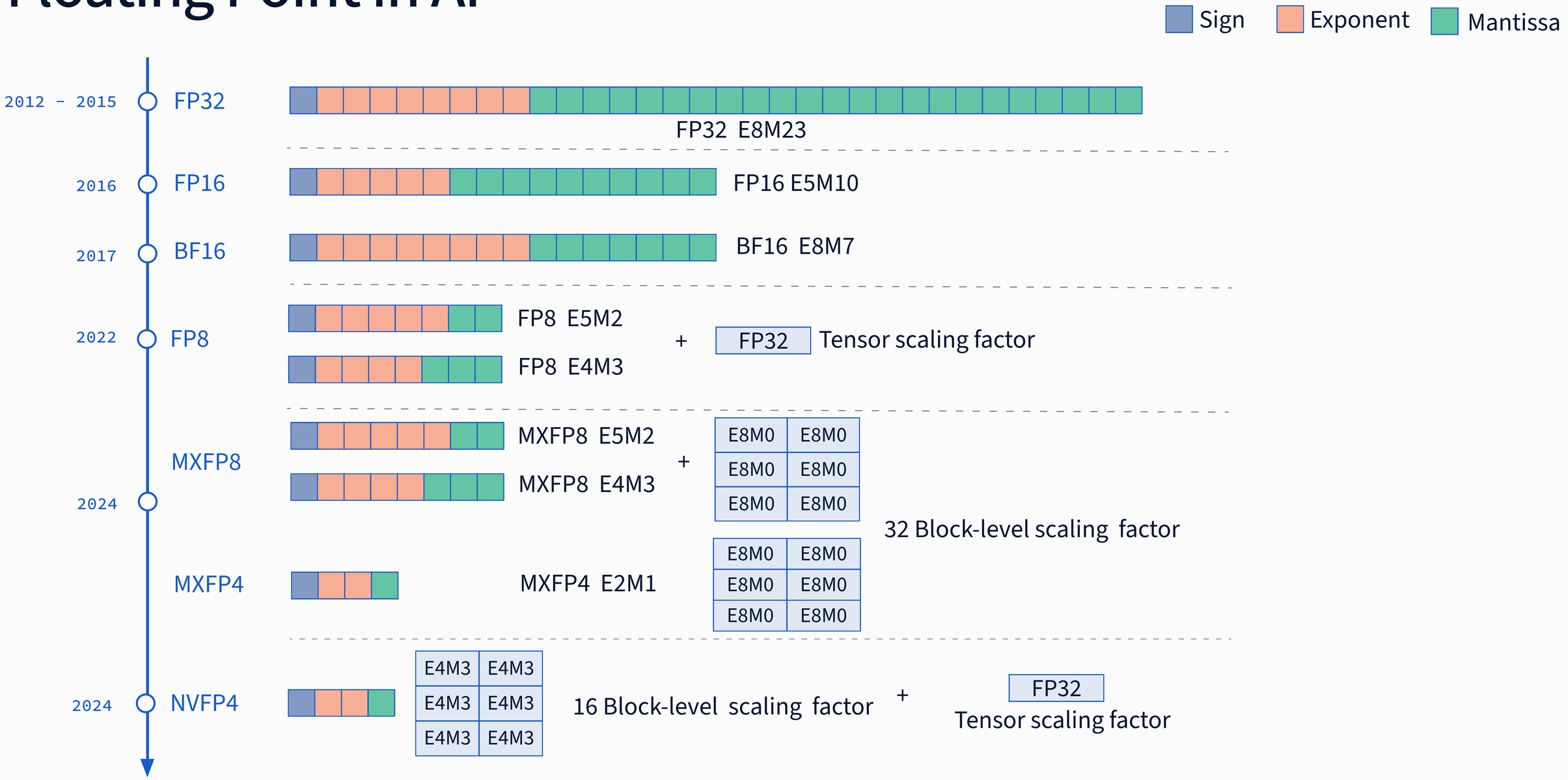
$$N_{10} = (-1)^S \times 1.M_{10} \times 2^{E_{10} - \text{bias}}$$

FP 16 bits:

`0.10001.1010000000` = $(6.510)_{10}$

$$(-1)^0 \times 1.101_2 \times 2^{10001_2 - 15} = (1 \times (1 + 2^{-1} + 2^{-3}) \times 2^2)_{10} =$$
$$= (4 \times 1.62510)_{10} = (6.510)_{10}$$

Floating Point in AI



Model Quantization Algorithms

There are several strategies to quantize neural networks:

- **Post Training Quantization (PTQ)**
  - Turn pre-trained ***weights*** *and/or* ***activations*** into low-precision formats
  - No need for training data or training hardware
  - May lead to a lower quality model

Neural Network Inference

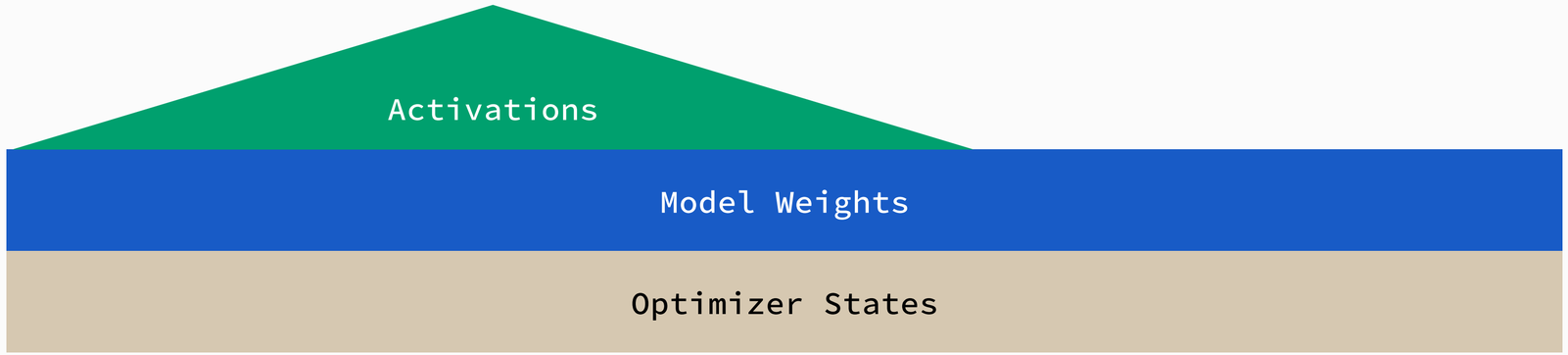
Slides inspired by GPU Mode Lec. 30:
https://www.youtube.com/@GPUMODE

Model Quantization Algorithms

There are several strategies to quantize neural networks:

- **Post Training Quantization (PTQ)**
  - Turn pre-trained ***weights*** *and/or* ***activations*** into low-precision formats
  - No need for training data or training hardware
  - May lead to a lower quality model
- **Quantization Aware Training (QAT)**
  - Adds quantization to the forward for training/fine-tuning

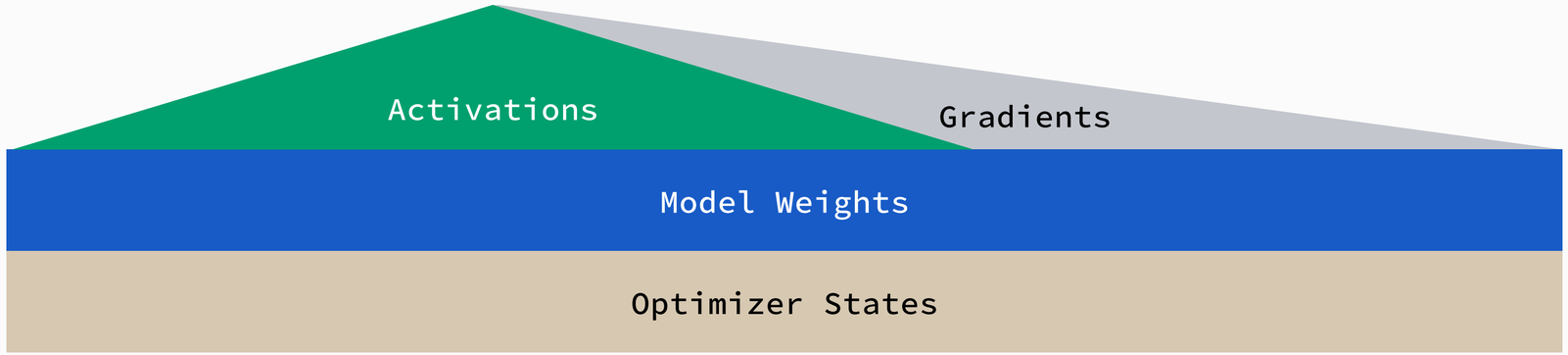
Neural Network FWD

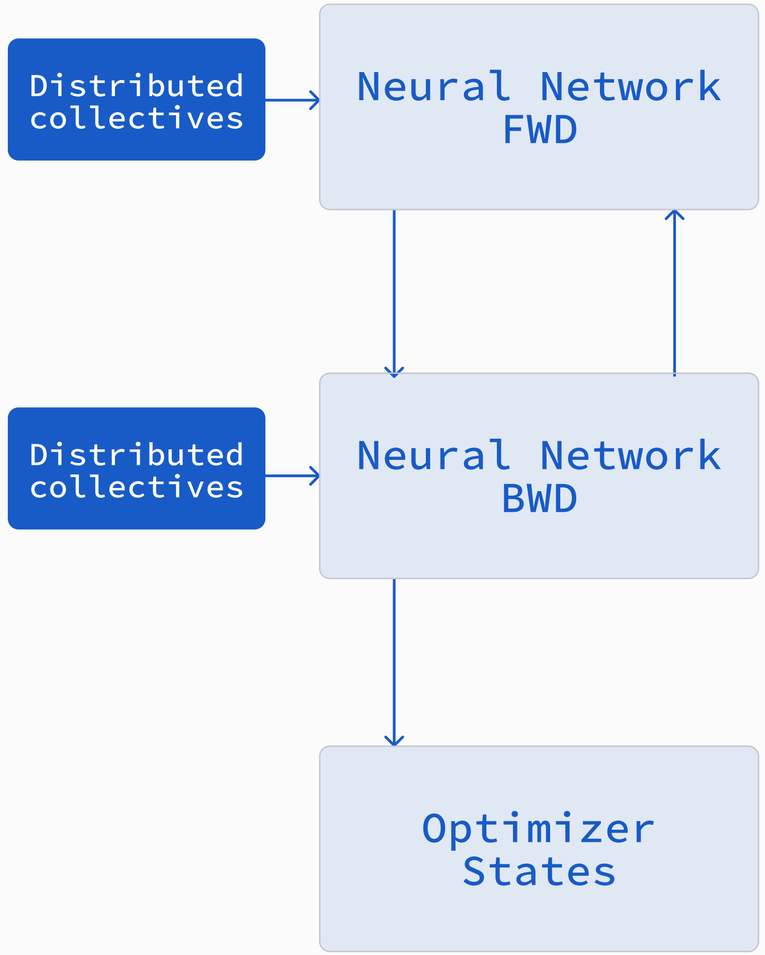


Model Quantization Algorithms

There are several strategies to quantize neural networks:

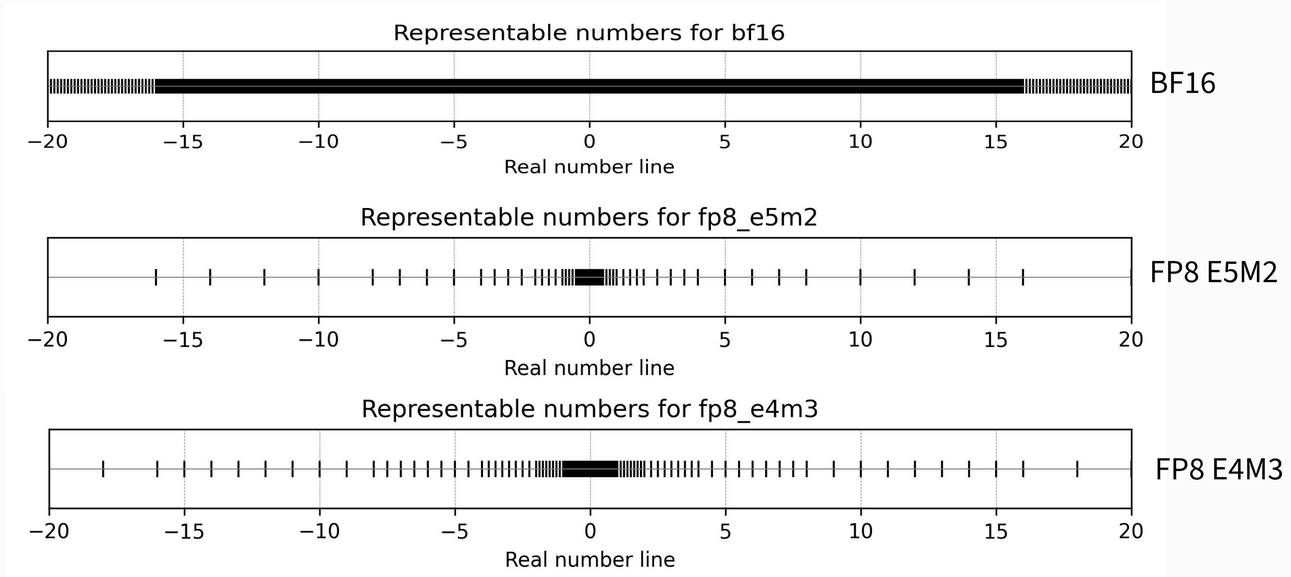
- **Post Training Quantization (PTQ)**
  - Turn pre-trained ***weights*** *and/or* ***activations*** into low-precision formats
  - No need for training data or training hardware
  - May lead to a lower quality model
- **Quantization Aware Training (QAT)**
  - Adds quantization to the forward for training/fine-tuning
- **Quantized Training (QT)**
  - Adds quantization also to the backward pass improving over QAT



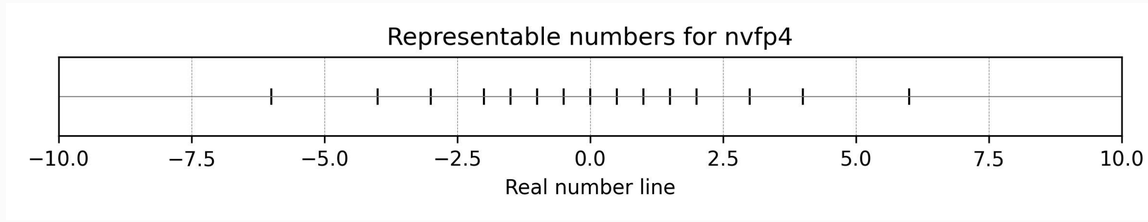


NVFP4: Are 16 values enough?

sign	exponent	mantissa		sign	exponent	mantissa	
0	0	0	0 = 0.0	1	0	0	0 = -0.0
0	0	0	1 = 0.5	1	0	0	1 = -0.5
0	0	1	0 = 1.0	1	0	1	0 = -1.0
0	0	1	1 = 1.5	1	0	1	1 = -1.5
0	1	0	0 = 2.0	1	1	0	0 = -2.0
0	1	0	1 = 3.0	1	1	0	1 = -3.0
0	1	1	0 = 4.0	1	1	1	0 = -4.0
0	1	1	1 = 6.0	1	1	1	1 = -6.0



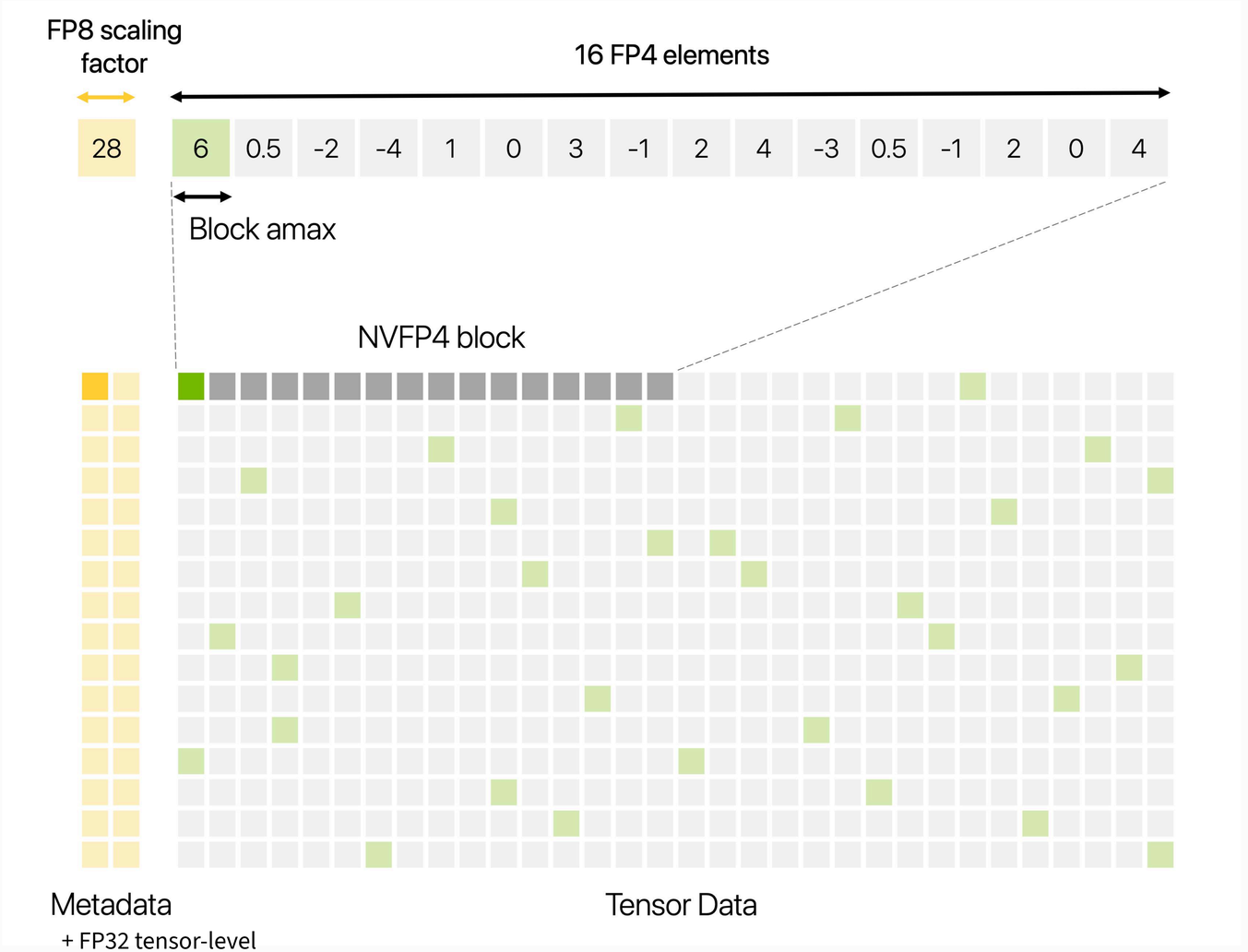
NVFP4 E2M1



NVFP4 vs. MXFP4

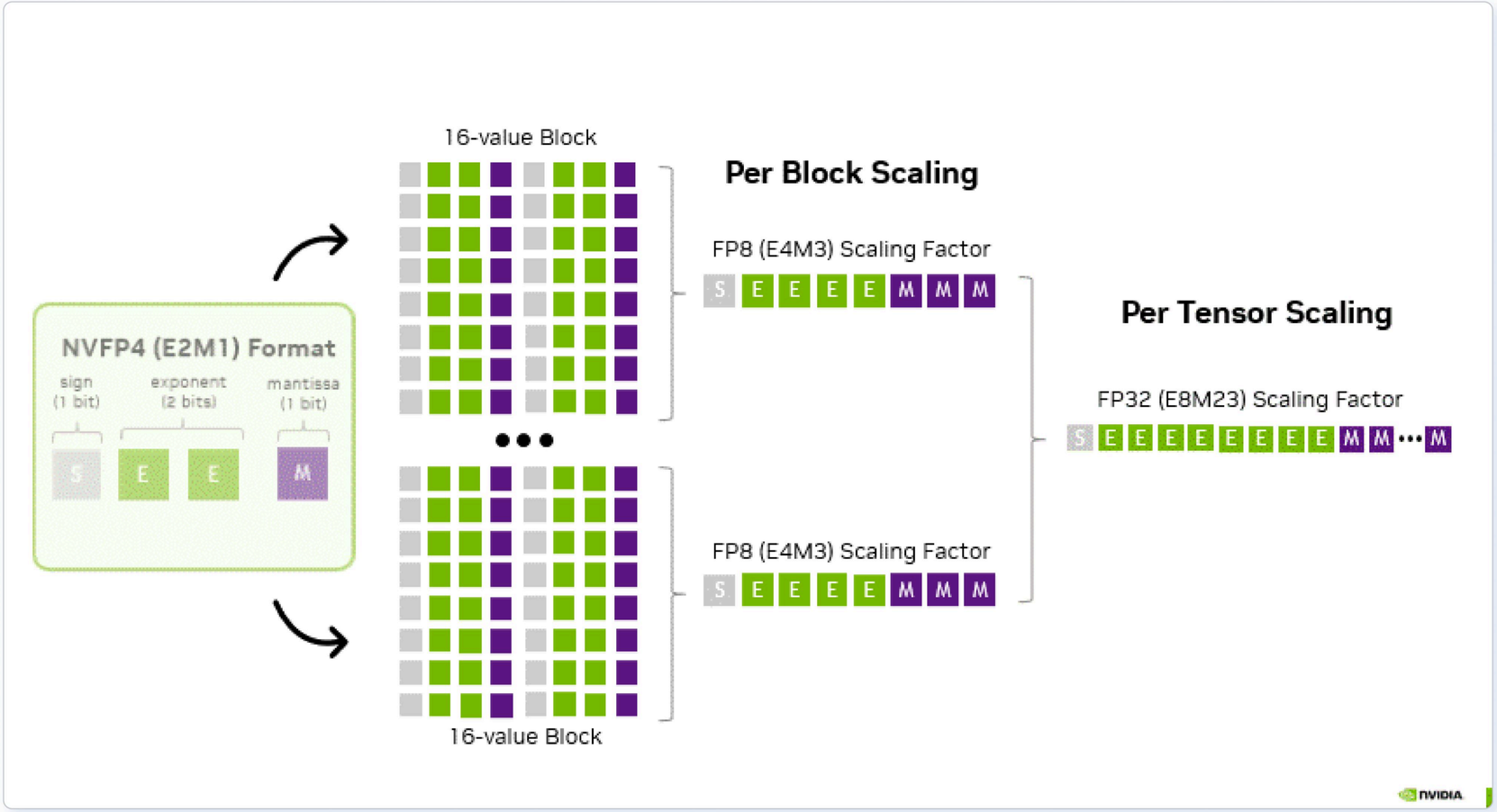
The smaller block size and more precise scaling allows:

- to increase accuracy of outliers
- to minimize the values quantized to zero



Source: Pretraining Large Language Models with NVFP4 – arxiv:2509.25149

NVIDIA NVFP4

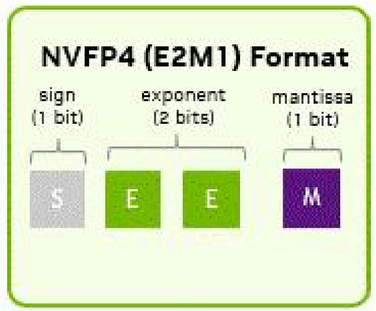


Why two levels?

- E4M3 block scales provide fractional precision for accurate scaling, but max out at 448.
- FP32 tensor scale first shrinks values into E4M3's limited range to prevent overflow.

Source: Pretraining Large Language Models with NVFP4 - arxiv:2509.25149

NVIDIA NVFP4



Why two levels?

- E4M3 block scales provide fractional precision for accurate scaling, but max out at 448.
- FP32 tensor scale first shrinks values into E4M3's limited range to prevent overflow.

Source: Pretraining Large Language Models with NVFP4 - arxiv:2509.25149

NVFP4 Recipe

- 2D Scaling

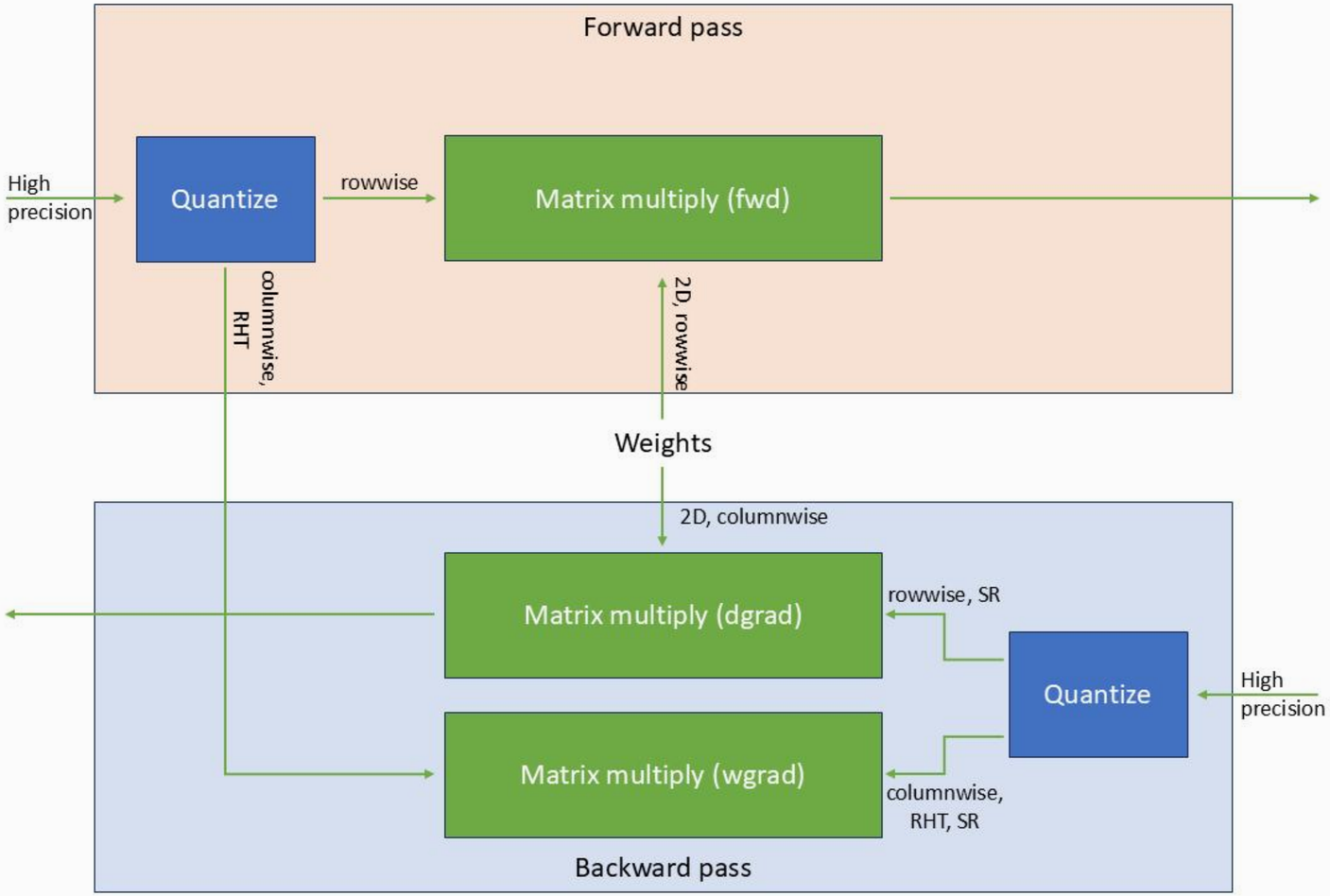
$$y = Q(x) \cdot Q(W^T) \quad \texttt{fwd pass}$$

$$\partial x = Q(\partial y) \cdot Q(W) \quad \texttt{bwd dgrad}$$

Issue: $Q(W) \neq Q(W^T)$

Solution:

2D 16 x 16 blocks when quantizing weights



Linear Layer flow w/ NVFP4

Source: Pretraining Large Language Models with NVFP4 – arxiv:2509.25149

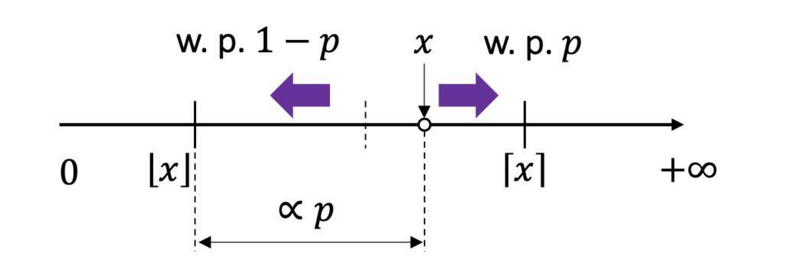
NVFP4 Recipe

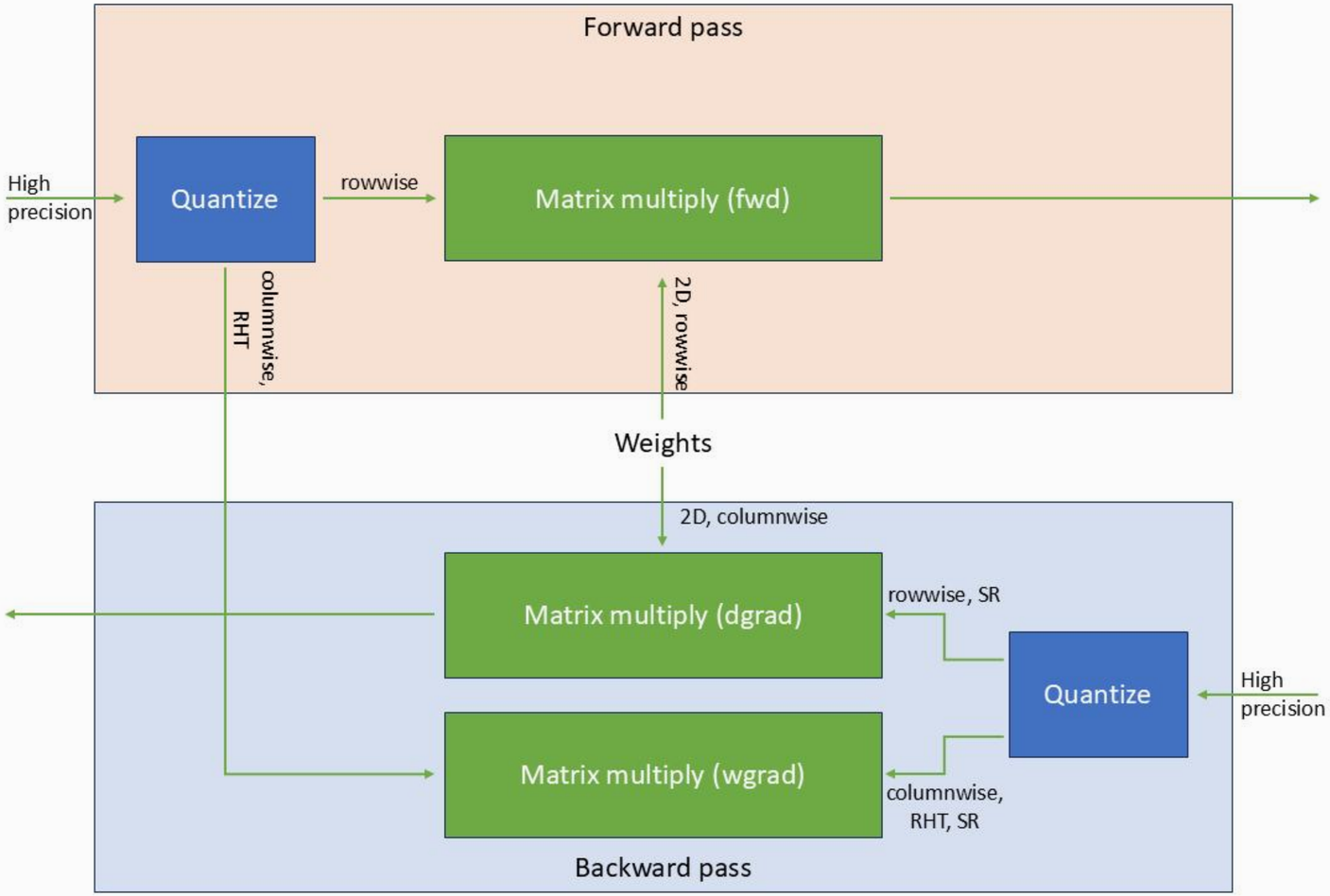
- 2D Scaling
- Stochastic Rounding

$$\mathrm{Round}(x) = \begin{cases} \lfloor x \rfloor, & \text{w/ prob. } 1-p, \\ \lceil x \rceil, & \text{w/ prob. } p. \end{cases}$$

$$p = (x - \lfloor x \rfloor)/(\lceil x \rceil - \lfloor x \rfloor)$$

$$\mathbb{E}[\mathrm{Round}(x)] = x$$





Linear Layer flow w/ NVFP4

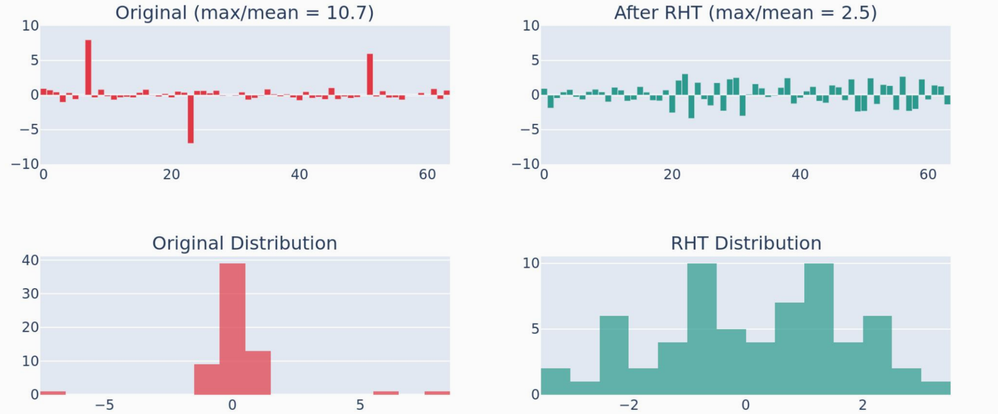
Source: Pretraining Large Language Models with NVFP4 - arxiv:2509.25149

NVFP4 Recipe

- 2D Scaling
- Stochastic Rounding
- Random Hadamard Transforms:
  - Activation outliers (`Wgrad` inputs)

$$H^T H = I, \quad (xH)(WH)^T = XW^T$$

$$\mathrm{RHT}(x) = H \cdot \mathrm{diag}(s) \cdot x$$
$$\text{where } s \in \{-1, +1\}^n$$



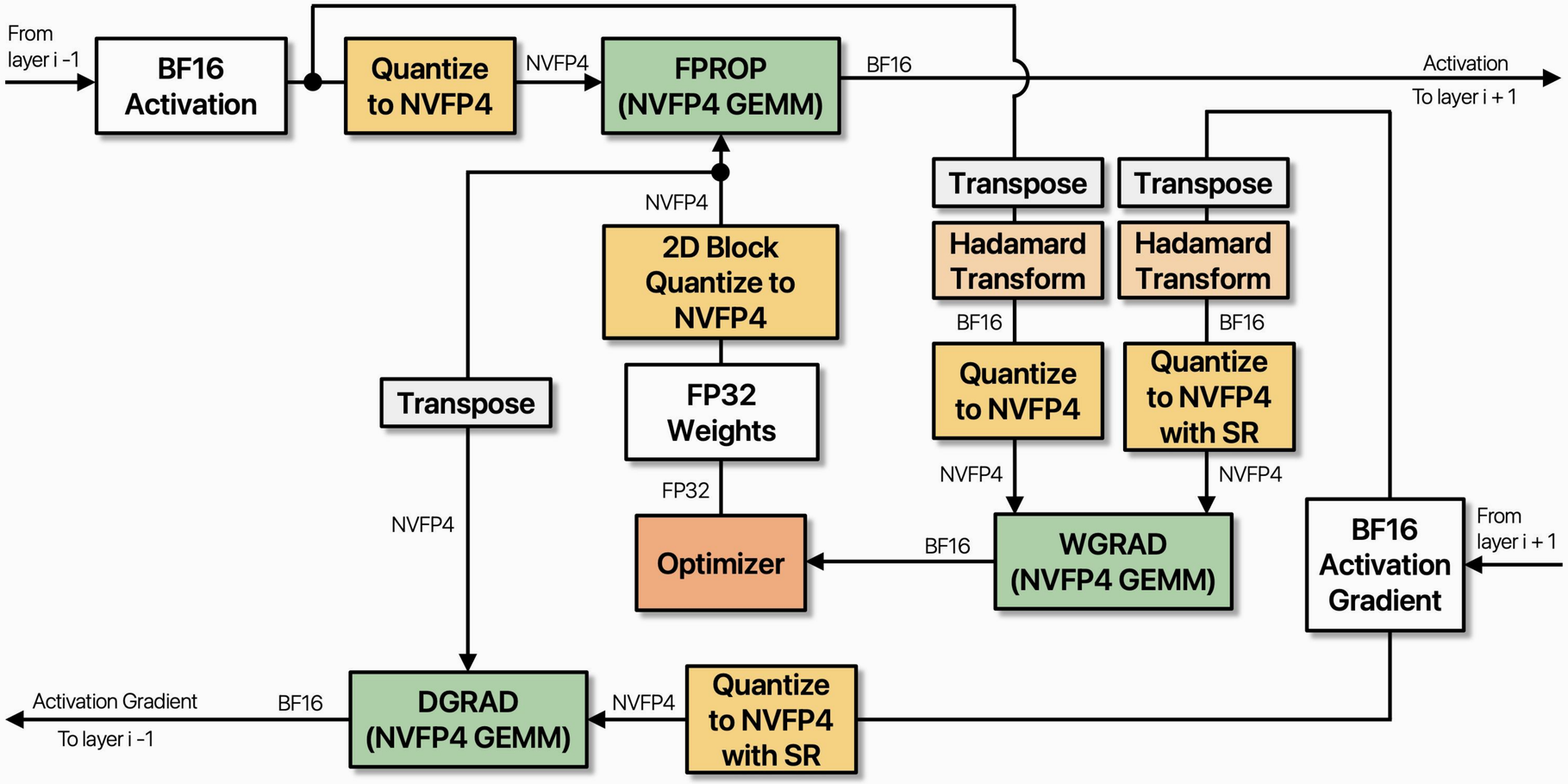

```python
def get_wgrad_sign_vector(device: int) -> torch.Tensor:
    """Hard-coded random signs for Hadamard transform.

    https://xkcd.com/221/

    """
    return torch.tensor(
        [1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, 1, -1, -1],
        dtype=torch.float32,
        device=device,
    )
```

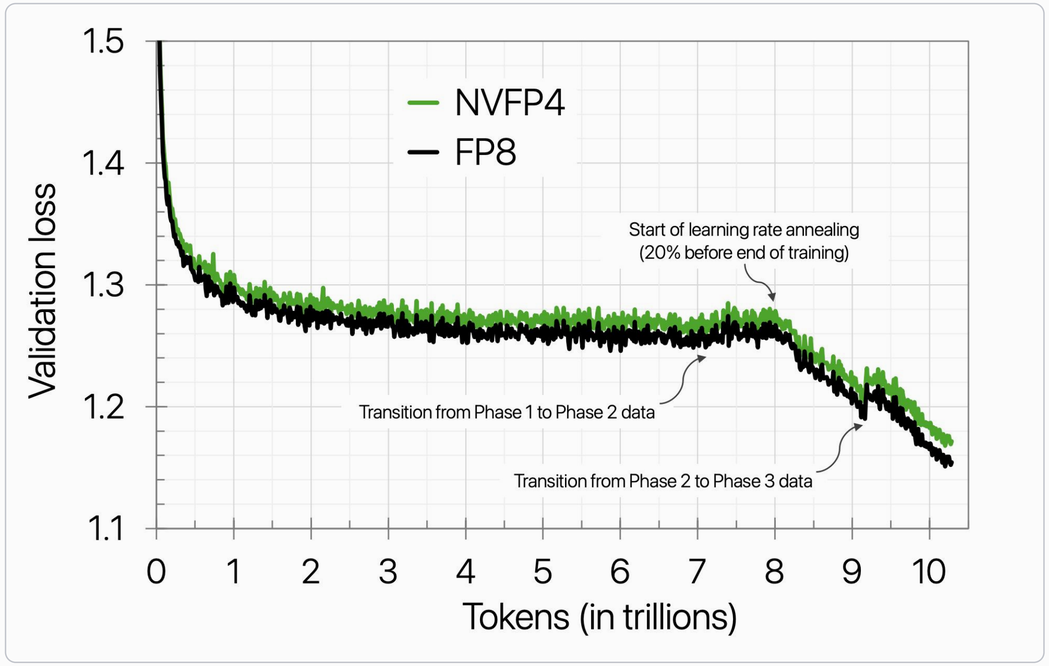
Source: Pretraining Large Language Models with NVFP4 - arxiv:2509.25149

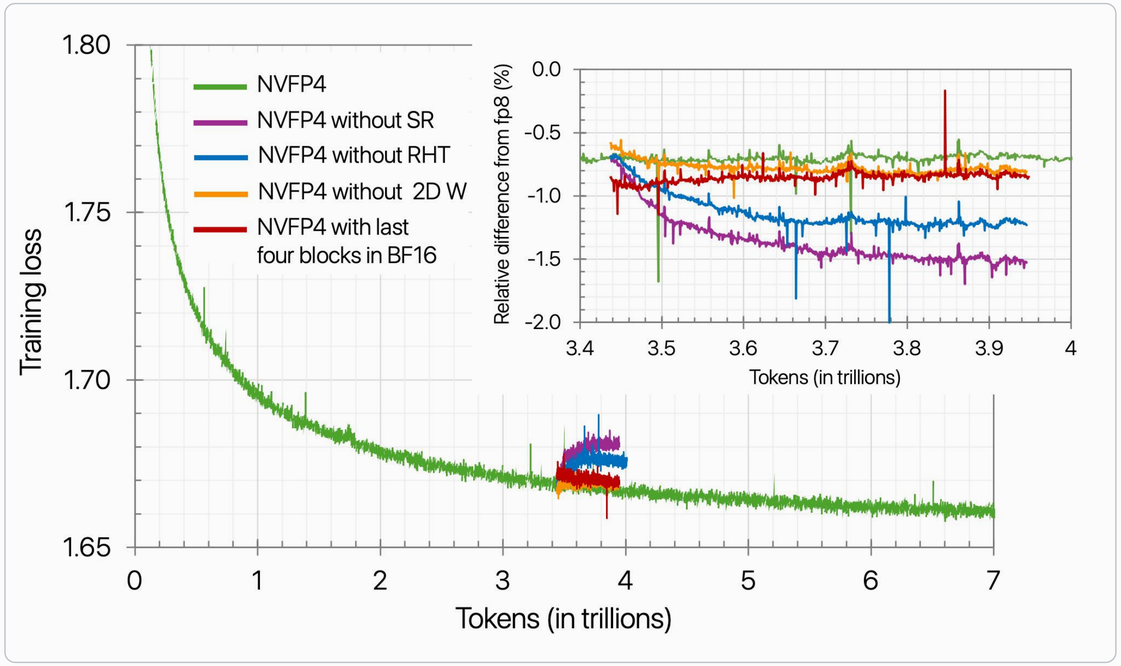
NVIDIA NVFP4 Recipe



Source: Pretraining Large Language Models with NVFP4 - arxiv:2509.25149

NVIDIA NVFP4 Results





- **NVFP4 ≈ FP8:**
  Loss curves nearly identical throughout 10T token training run
- **Throughput : 2x on GB200 / 3x on GB300** vs. FP8
- **Memory: ~2x** (half the memory)

Conclusions

Use NVFP4 when:

- You have Blackwell hardware
  (B300, GB300, B200, GB200, RTX PRO 6000)
- Serving large models where memory is the bottleneck
- Training frontier models and achieving 2-3x throughput

Things to keep in mind:

- Still an active area of research
- Requires complex training recipes
- Attention kernels still in development

Summary of NVFP4 recipe:

GEMM	RHT	Scaling	Rounding
Fprop	N	x: 1D, W^T: 2D	RTN
Dgrad	N	∂y: 1D, W: 2D	∂y: SR
Wgrad	Y	x^T: 1D, ∂y: 1D	x: RTN, ∂y: SR

verda

Thanks for listening!

Use the code:

MLOPS-COMMUNITY-FEB26

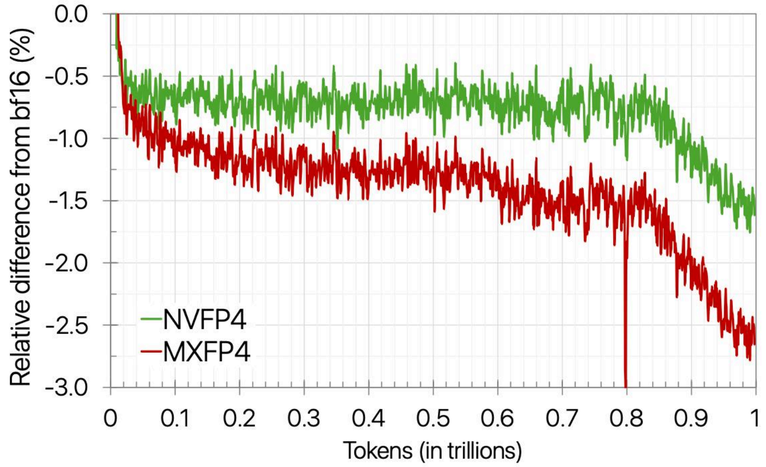
(200€ free credits)

Contacts:

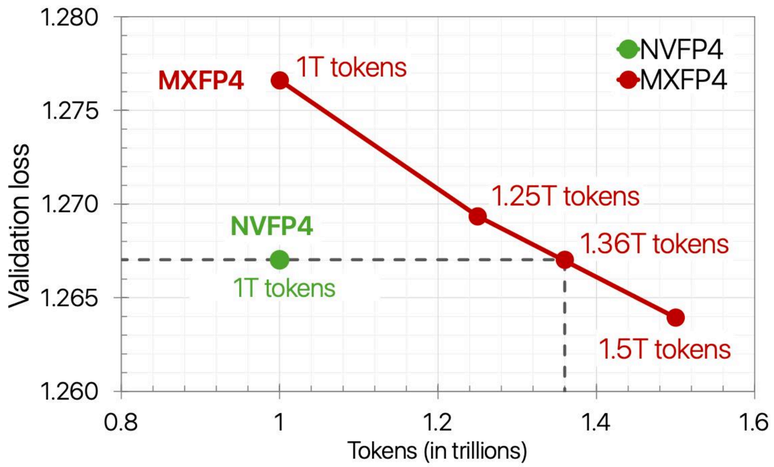
Riccardo Mereu
riccardo@verda.com

NVIDIA NVFP4 Results pt. 2

Format	Element	Scale	Block	Speedup vs. BF16	
				GB200	GB300
MXFP8	E5M2/E4M3	UE8M0	32	2×	2×
MXFP6	E3M2/E2M3	UE8M0	32	2×	2×
MXFP4	E2M1	UE8M0	32	4×	6×
NVFP4	E2M1	E4M3	16	4×	6×



(a) Relative difference between training loss of BF16 (baseline) and NVFP4 and MXFP4 pretraining.

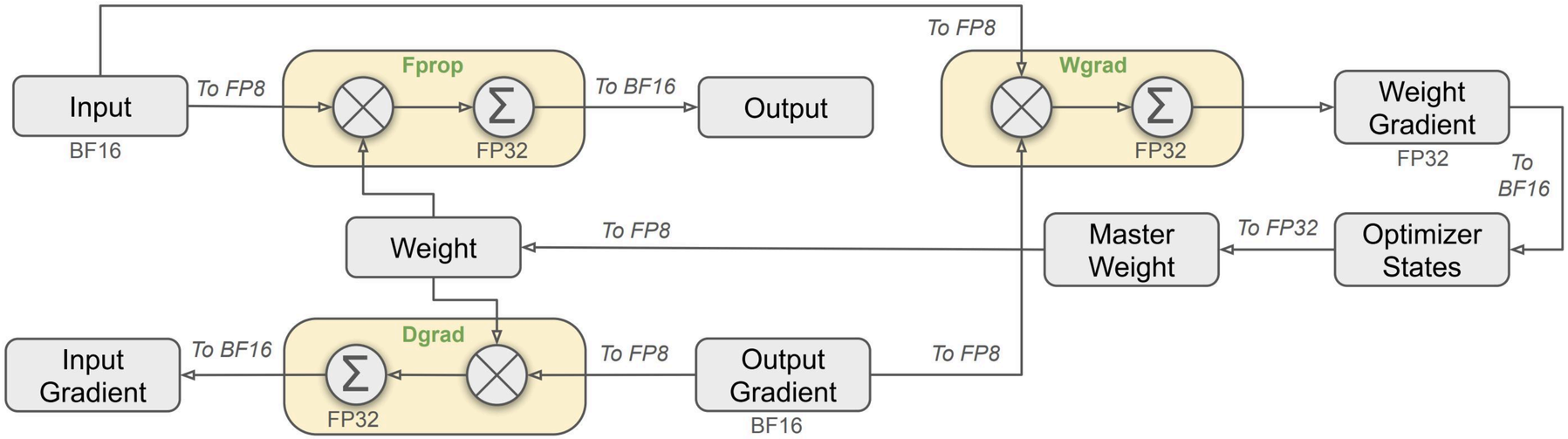


(b) Final validation loss for NVFP4 and MXFP4 pretraining with different number of tokens.

Figure 6 | NVFP4 vs MXFP4 comparisons: (a) training-loss difference; (b) validation perplexity across token budgets.

verda

DeepSeek-V3 Mixed Precision



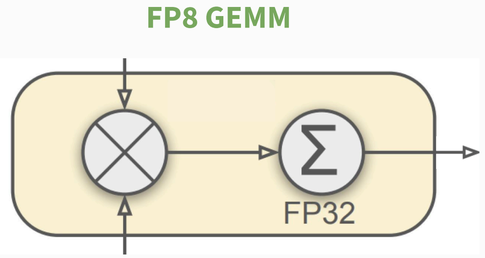
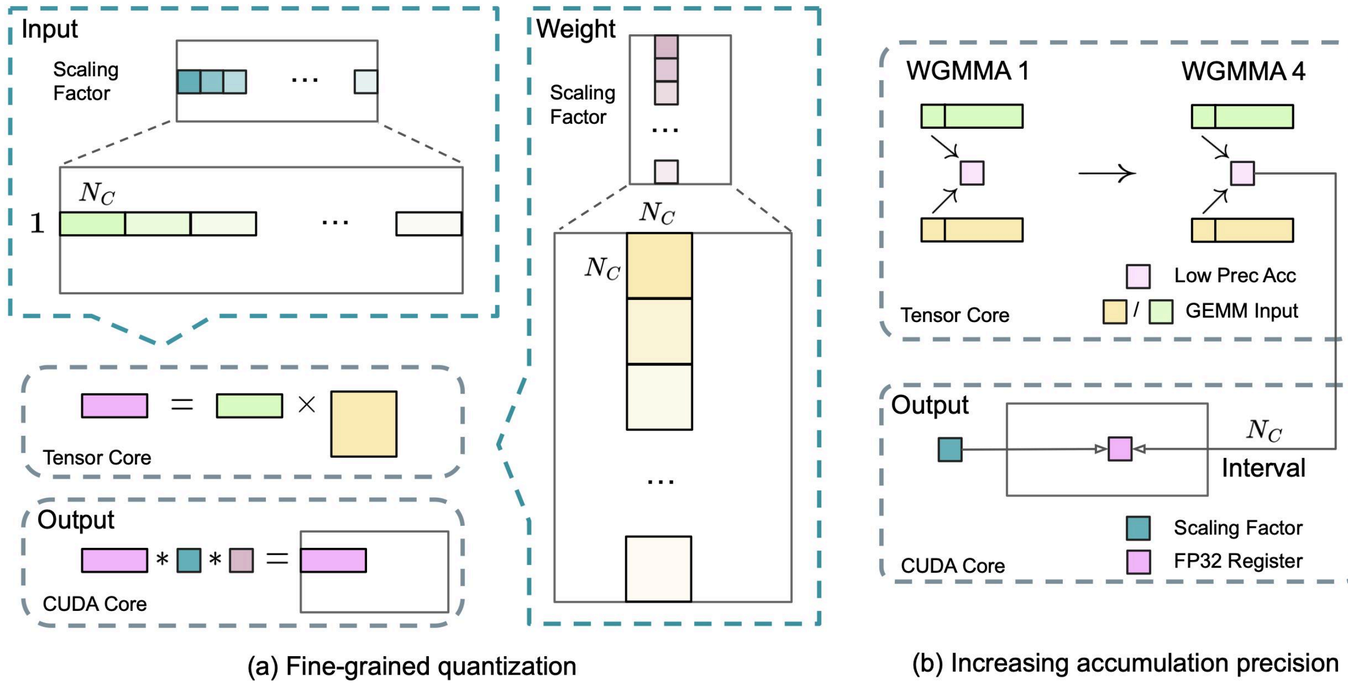


Figure 6 | The overall mixed precision framework with FP8 data format. For clarification, only the `Linear` operator is illustrated.

Source: DeepSeekv3 Technical Report, https://arxiv.org/abs/2412.19437v2

DeepSeek V3



(a) Fine-grained quantization

(b) Increasing accumulation precision

Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only `Fprop` is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

Source: DeepSeekv3 Technical Report, https://arxiv.org/abs/2412.19437v2