Section 4: Quantization

CSE 493G1, Spring 2025

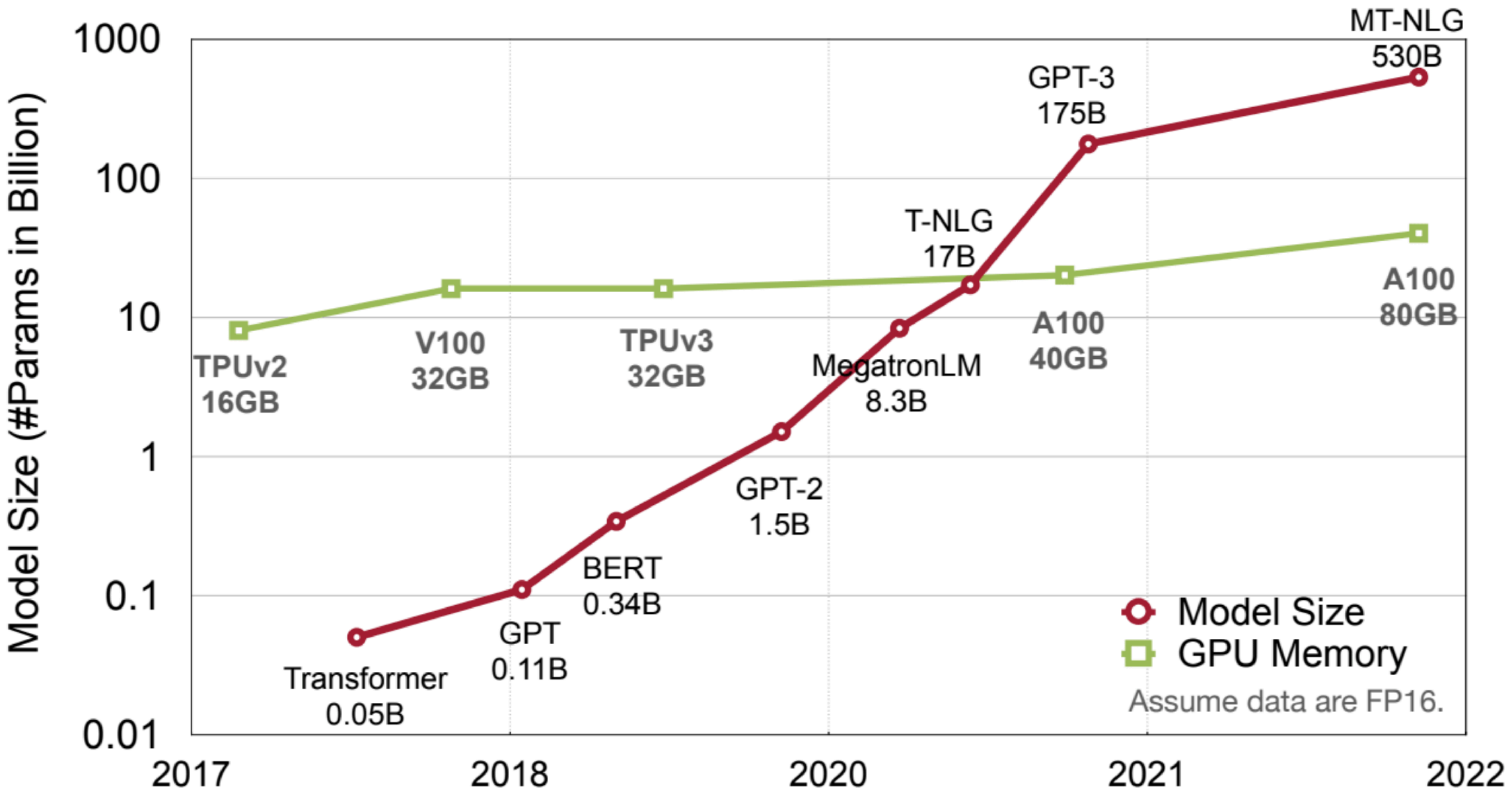
Tanush Yadav

Course Logistics

- A2 due Sunday (4/27)
- Project Proposal due Tuesday (4/29)
- W2 due 5/9
- A3 due 5/11

Motivation & Background

DL models are outgrowing hardware

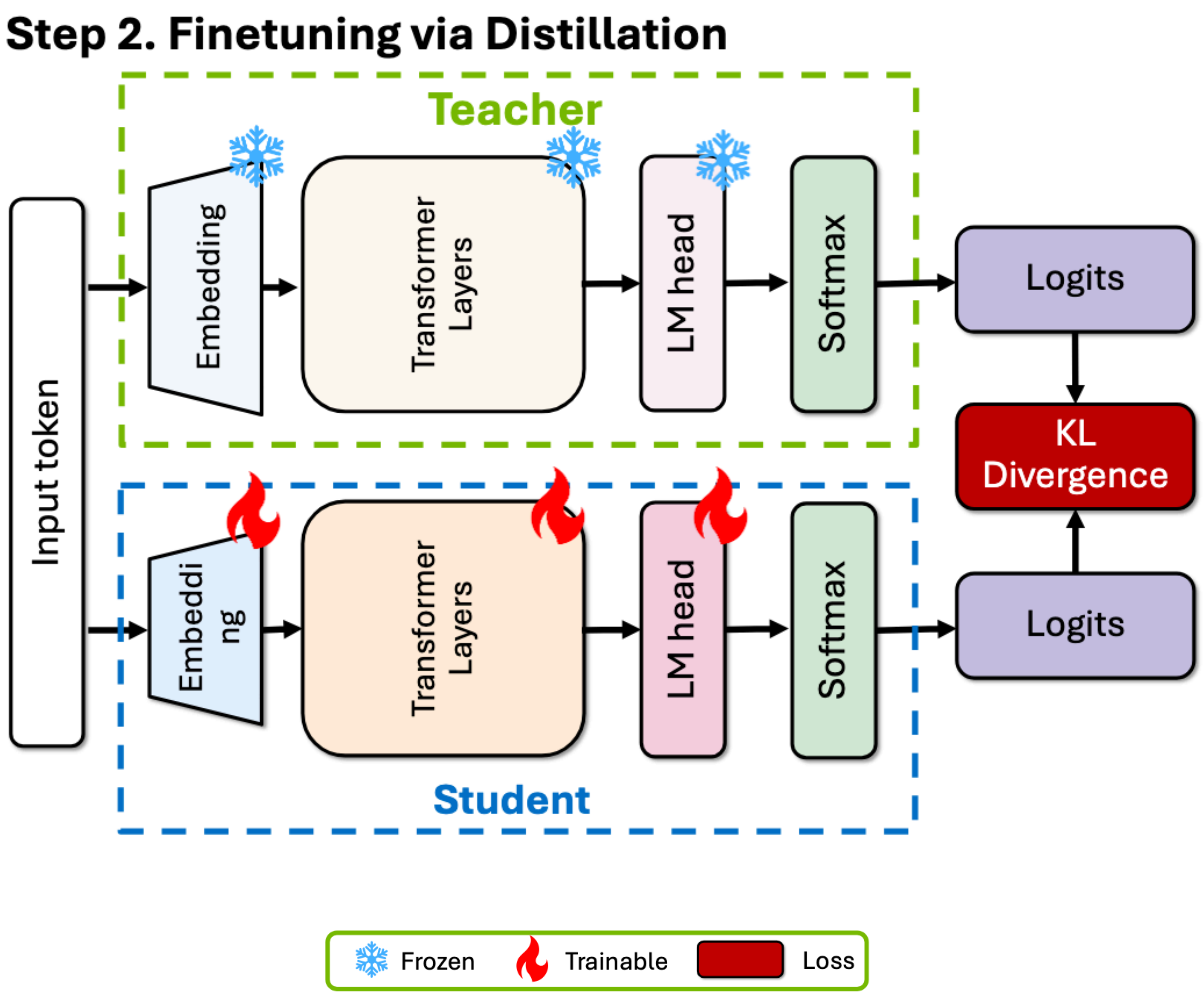


Decreasing model size* is a worthy goal.

*while preserving performance

Distillation

A core tenet of the DeepSeek paper



Sreenivas et al. 2024, “Minitron”

$$\text{model size} = (\text{\# of parameters}) \times (\text{parameter size})$$

$$\text{model size} = (\text{\# of parameters}) \times (\text{parameter size})$$

pruning quantization

Parameter Size

How many bits does each parameter occupy? (“bit width”)

- FP32 ⇒ 4 bytes per parameter
- FP16, BF16 ⇒ 2 bytes per parameter
- FP8, int8 ⇒ 1 byte per parameter
- int4 ⇒ 0.5 bytes per parameter

Parameter Size

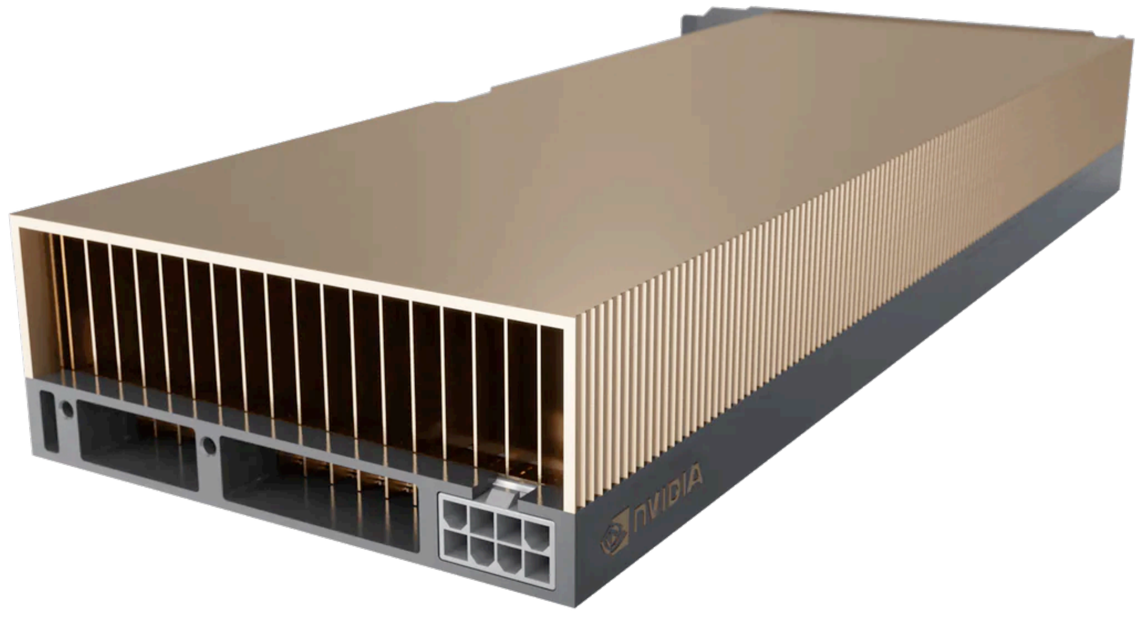
Qwen2.5-VL has 3B, 7B, and 72B versions

NVIDIA 2080 Ti

11GB memory

NVIDIA RTX 6000

24GB memory



NVIDIA A40

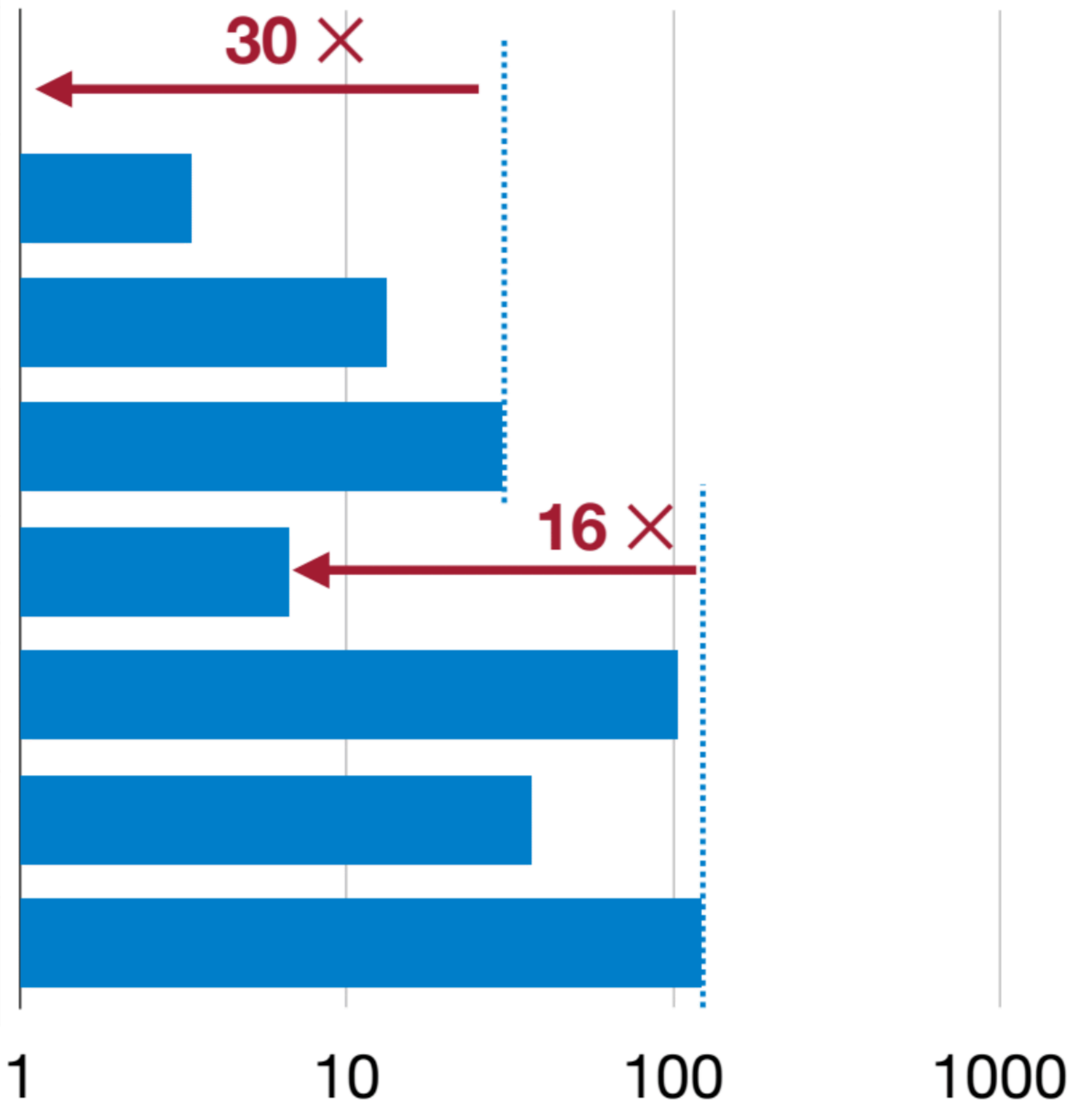
40GB memory

Parameter Size

Operations are cheaper on lower bit widths

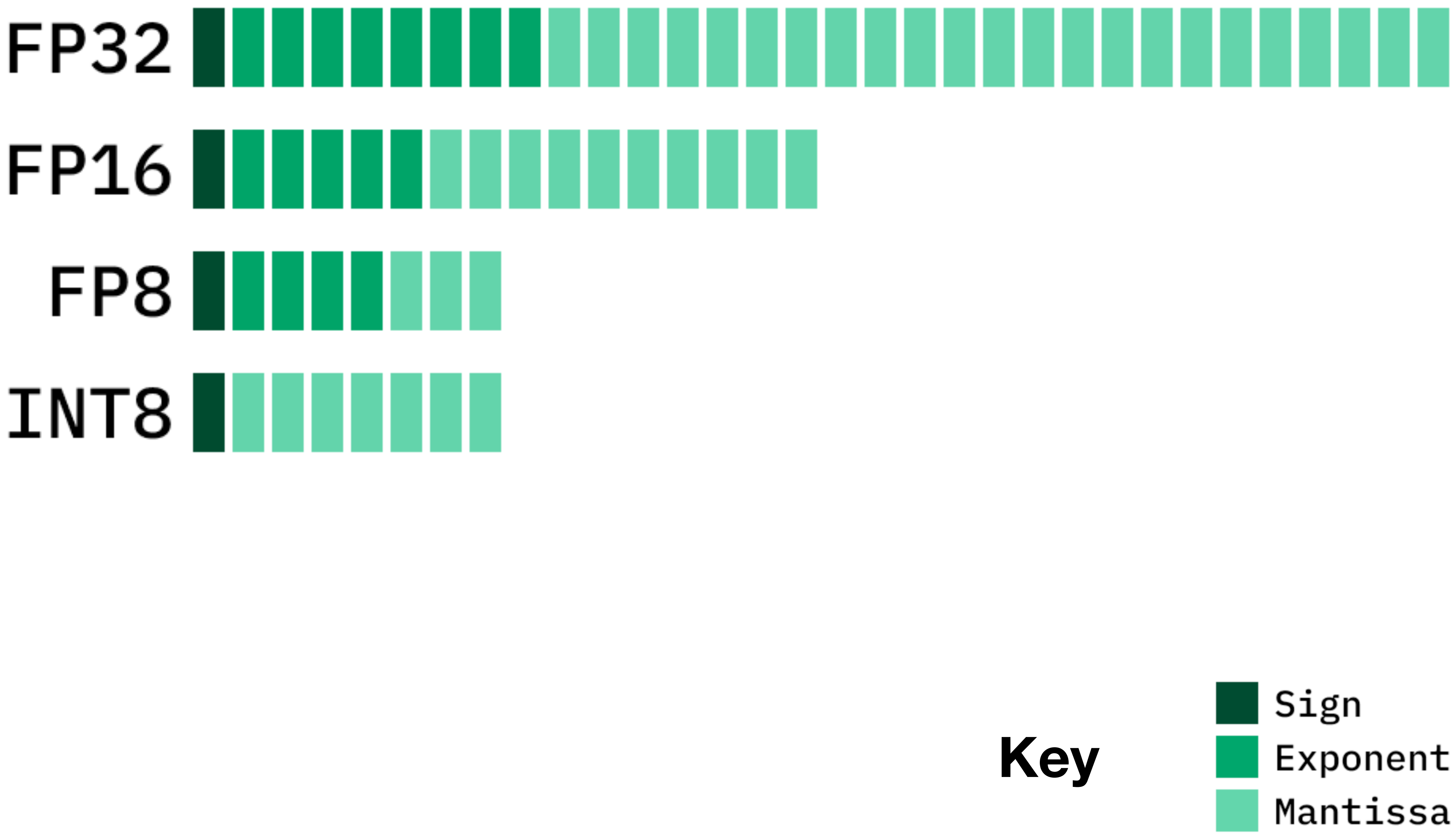
Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7

Rough Energy Cost For Various Operations in 45nm 0.9V



Parameter Size

What do the underlying bits represent?



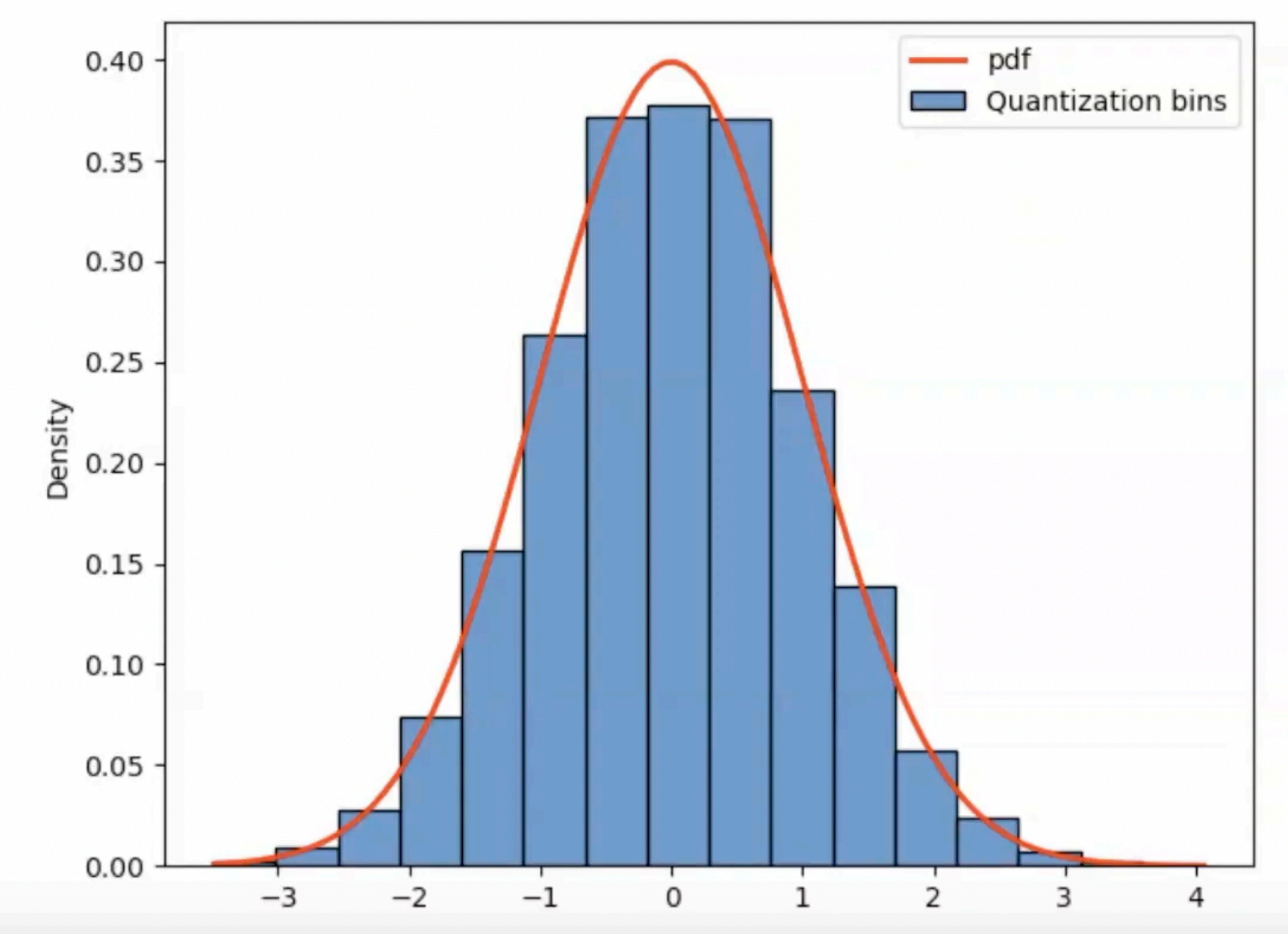
	Min	Max
FP32	3.4×10^{38}	-3.4×10^{38}
FP16	-65504	65504
FP8	-448	448
INT8	-128	127

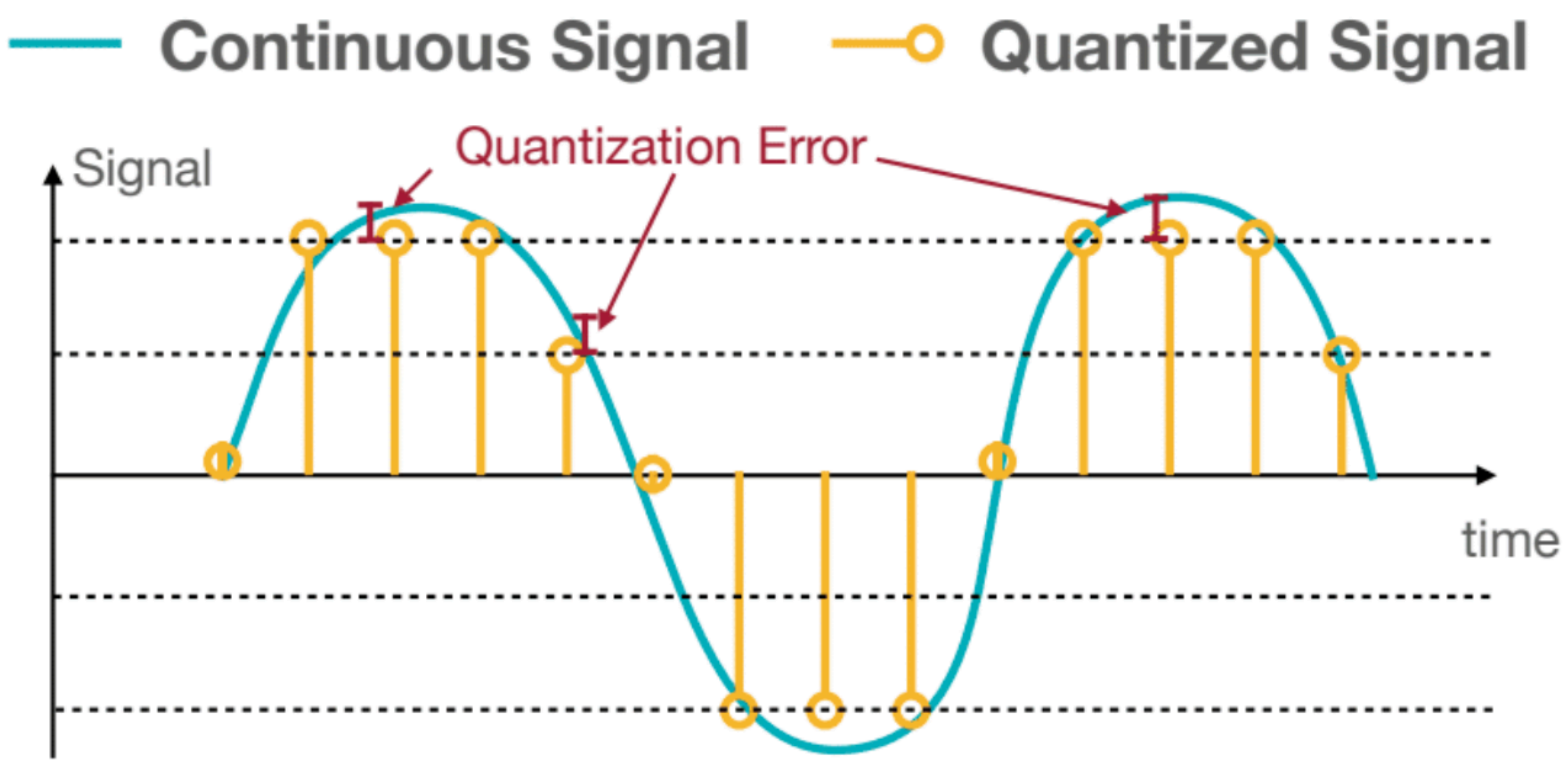
Baseten, "FP8"

Quantization

A well-defined tool in math & signal processing

- process of constraining an input from a continuous or otherwise large set of values to a smaller discrete set





Tim Dettmers, "8-bit Methods"
Song Han, MIT 6.5940, Lecture 5

Quantization

In the context of deep learning

- process of constraining an input from a continuous or otherwise large set of values to a smaller discrete set

fp16 → int8

while minimizing quantization error

Tim Dettmers, "8-bit Methods"
Song Han, MIT 6.5940, Lecture 5

Mathematical Quantization Techniques

Quantization

Formalizing our problem

- Given a matrix of floating point numbers, we wish to…
  - store it as an integer matrix from which we can re-construct a float matrix
  - minimize the difference between the original matrix and the reconstructed matrix (“quantization error”)

Quantization

Formalizing our problem

- Let q be an integer. We wish to map it to a real number r
  - Intuitively, we want to **shift** and **scale** our integer
  - Mathematically, we want to apply an **affine mapping**
    (that is, a transformation that preserves collinearity and ratio of distances)

$$r = S(q - Z) = Sq - Sz$$

S **is the scaling factor (float)**
Z **is the zero-point (int)**

Jacobs et al. 2018, "Efficient Integer-Arithmetic-Only Inference"

Zeropoint Quantization

$$r = S(q - Z)$$

Also known as "asymmetric quantization"

- Assume $S = 1.5 \in \mathbb{R}$ and $Z = 1 \in \mathbb{Z}$, then
  - for $q = 0$ we have $r = 1.5(0 - 1) = 1.5(-1) = -1.5 \in \mathbb{R}$
  - for $q = 1$ we have $r = 1.5(1 - 1) = 1.5(0) = 0 \in \mathbb{R}$
  - for $q = 2$ we have $r = 1.5(2 - 1) = 1.5(1) = 1.5 \in \mathbb{R}$
  - for $q = 3$ we have $r = 1.5(3 - 1) = 1.5(2) = 3 \in \mathbb{R}$

Zeropoint Quantization

We can go the other way too

$$r = S(q - Z) \quad \Rightarrow \quad q = \left\lfloor \frac{r}{S} + Z \right\rceil$$

- Assume $S = 1.5 \in \mathbb{R}$ and $Z = 1 \in \mathbb{Z}$, then
  - for $r = 1$ we have $q = \left\lfloor \frac{1}{1.5} + 1 \right\rceil = \lfloor 0.\bar{6} + 1 \rceil = \lfloor 1.\bar{6} \rceil = 2 \in \mathbb{R}$
  - for $r = 0$ we have $q = \left\lfloor \frac{0}{1.5} + 1 \right\rceil = \lfloor 0 + 1 \rceil = \lfloor 1 \rceil = 1 \in \mathbb{Z}$

Zeropoint Quantization

We can go the other way too

$$r = S(q - Z) \quad \Rightarrow \quad q = \left\lfloor \frac{r}{S} + Z \right\rceil$$

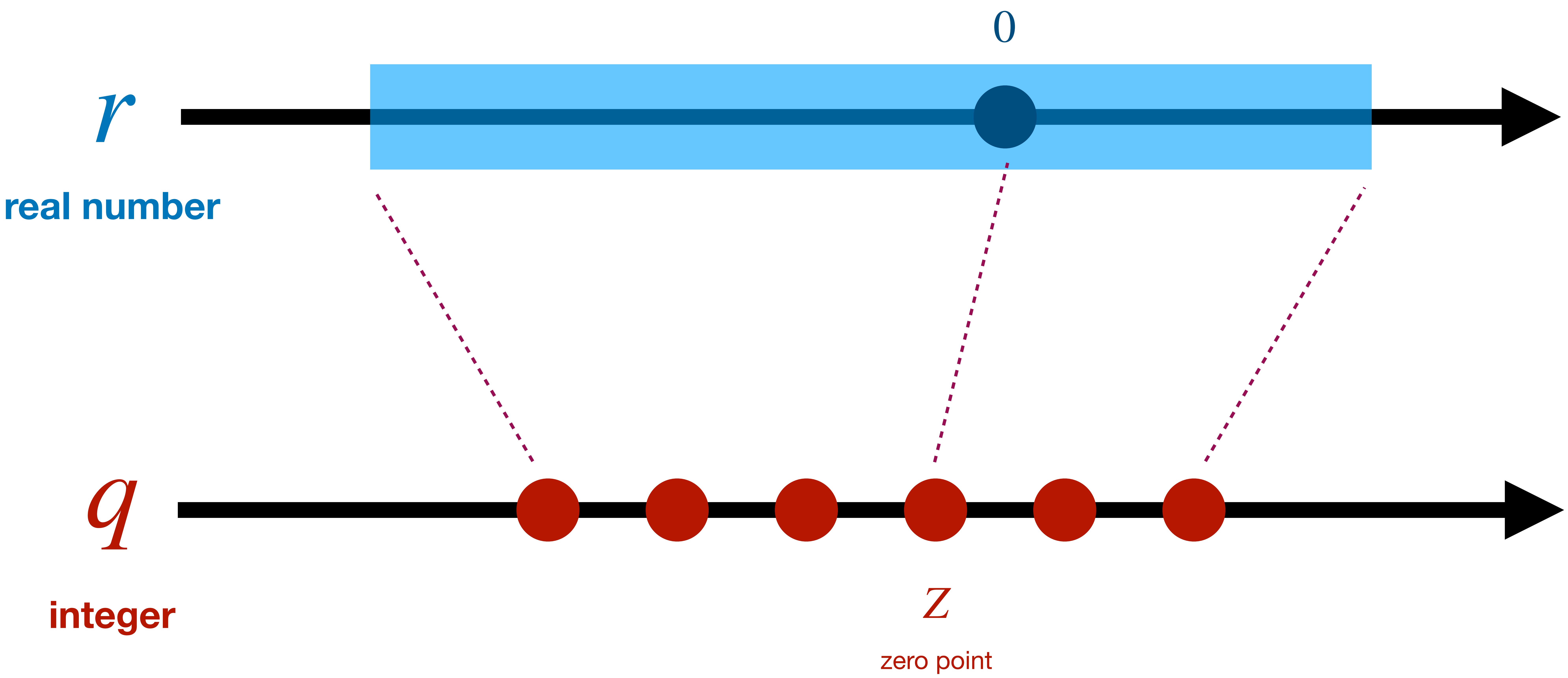
- Regardless of which S and Z we choose, $r = 0$ maps **exactly** to $q = Z$

$$q = \left\lfloor \frac{0}{S} + Z \right\rceil = \lfloor Z \rceil = Z$$

Zeropoint Quantization

$$r = S(q - Z)$$

How do we determine S and Z ?



Quantization

Formalizing our problem

- Given a **specific** matrix,

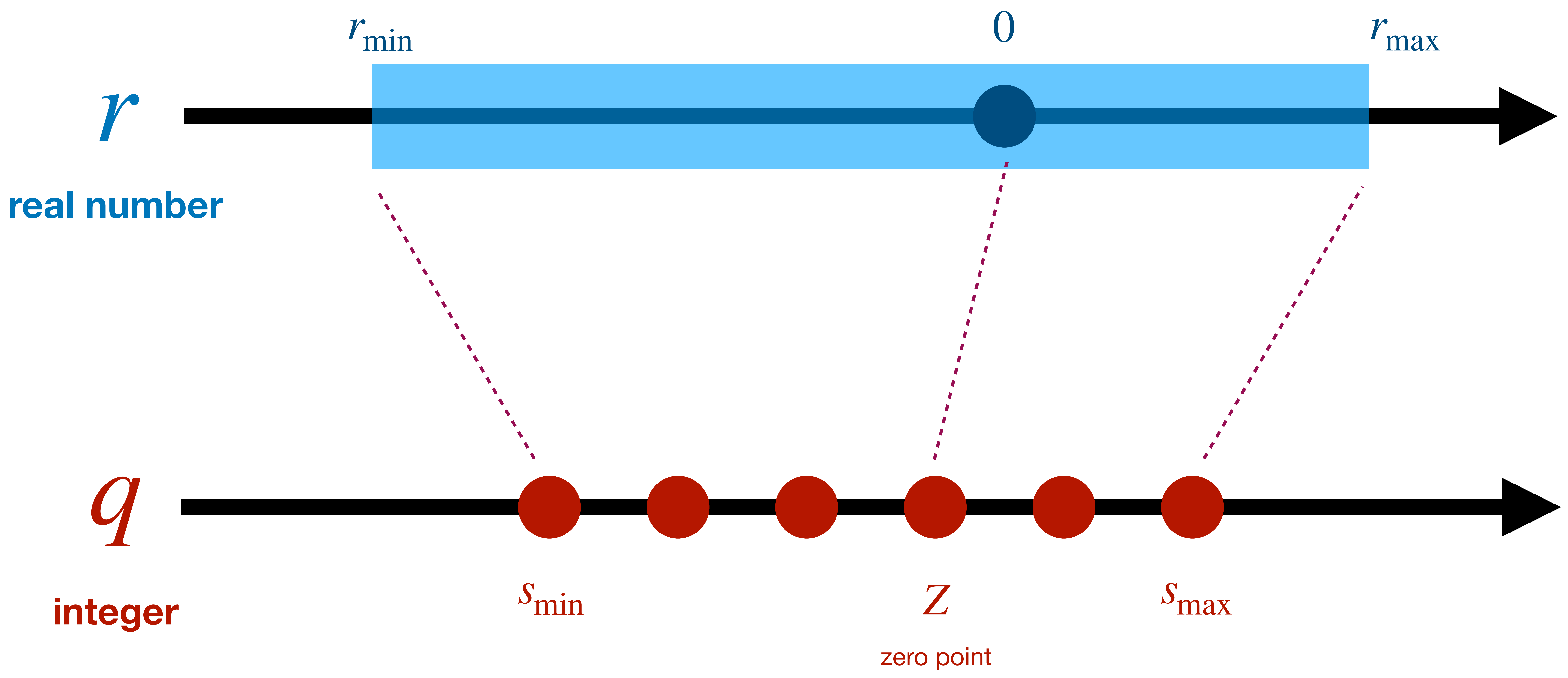
fp16 → int8

while minimizing quantization error

Zeropoint Quantization

$$r = S(q - Z)$$

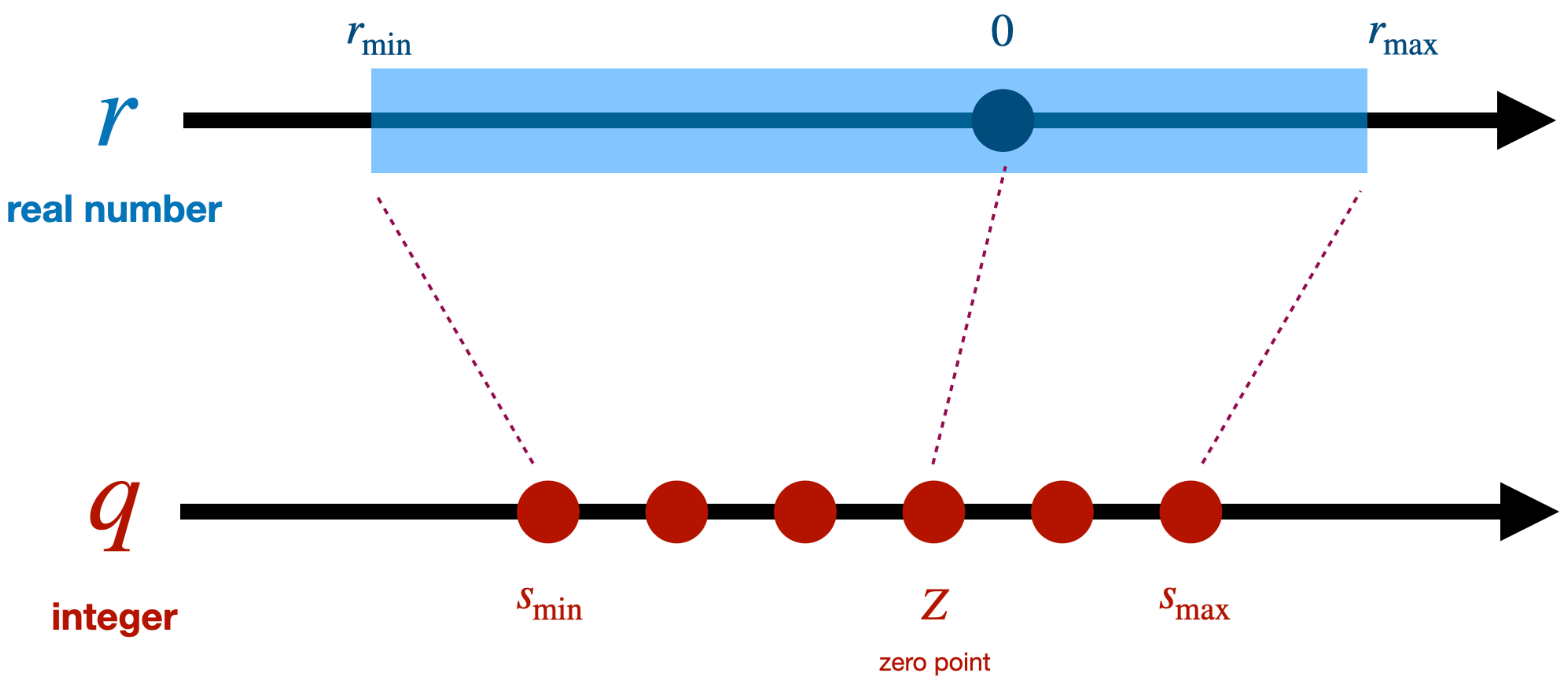
How do we determine S and Z ?



Zeropoint Quantization

$$r = S(q - Z)$$

How do we determine S and Z ?



$$r_{\min} = S\left(q_{\min} - Z\right) \qquad r_{\max} = S\left(q_{\max} - Z\right)$$

Zeropoint Quantization

How do we determine S and Z ?

$$r_{\min} = S\left(q_{\min} - Z\right) \qquad (1)$$

$$r_{\max} = S\left(q_{\max} - Z\right) \qquad (2)$$

Subtracting (1) from (2),

$$r_{\max} - r_{\min} = S\left(q_{\max} - Z\right) - S\left(q_{\min} - Z\right)$$

Simplifying,

$$r_{\max} - r_{\min} = S\left(q_{\max} - q_{\min}\right)$$

Zeropoint Quantization

How do we determine S and Z ?

$$r_{\min} = S\left(q_{\min} - Z\right) \qquad (1)$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \qquad (3)$$

We now have a quantity for S, so we can solve (1) for Z,

$$r_{\min} = S\left(q_{\min} - Z\right) \quad \Rightarrow \quad Z = \left\lfloor q_{\min} - \frac{r_{\min}}{S} \right\rceil$$

Zeropoint Quantization

Quantize the following matrix to 2-bit signed ints

Recall from CSE 351 that a N-bit signed integer has the range $[-2^{N-1}, 2^{N-1}-1]$

2.09	-0.98	1.48	0.09
0.05	-0.14	**-1.08**	**2.12**
0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$r_{\min} = -1.08 \qquad r_{\max} = 2.12$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} = \frac{2.12 - (-1.08)}{1 - (-2)} = 1.07 \in \mathbb{R}$$

$$Z = \left\lfloor q_{\min} - \frac{r_{\min}}{S} \right\rceil = \left\lfloor -2 - \frac{-1.08}{1.07} \right\rceil = -1 \in \mathbb{Z}$$

Zeropoint Quantization

Quantize the following matrix to 2-bit signed ints

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$S = 1.07$ $\quad Z = -1$

$$r = S(q - Z) \quad \Rightarrow \quad q = \left\lfloor \frac{r}{S} + Z \right\rceil$$

$$q = \left\lfloor \frac{r}{1.07} - 1 \right\rceil$$

Zeropoint Quantization

Quantize the following matrix to 2-bit signed ints

Original Weights (float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$$q = \left\lfloor \frac{r}{1.07} - 1 \right\rceil$$

Quantized Weights (int2)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

Examples

$$q = \left\lfloor \frac{r}{1.07} - 1 \right\rceil = \left\lfloor \frac{1.48}{1.07} - 1 \right\rceil = \lfloor 1.38 - 1 \rceil = \lfloor 0.38 \rceil = 0$$

$$q = \left\lfloor \frac{r}{1.07} - 1 \right\rceil = \left\lfloor \frac{1.87}{1.07} - 1 \right\rceil = \lfloor 1.75 - 1 \rceil = \lfloor 0.75 \rceil = 1$$

Zeropoint Quantization

Determine the quantization error

Quantized Weights (int2)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

$$r = 1.07(q + 1)$$

Reconstructed Weights (float)

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

Examples $r = 1.07(q + 1) = 1.07(0 + 1) = 1.07$ $\quad r = 1.07(q + 1) = 1.07(1 + 1) = 2.14$ $\quad r = 1.07(q + 1) = 1.07(-1 + 1) = 0$

Zeropoint Quantization

Determine the quantization error

Original Weights (float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Reconstructed Weights (float)

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

Quantization Error

-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

Zeropoint Quantization

Practice!

- Quantize the following matrix into 3-bit signed integers
- Reconstruct the weights from the quantized weights
- Calculate the quantization error

18	7	8
3	6	12
0	-3	-2

Recall from CSE 351 that a N-bit signed integer has the range $[-2^{N-1}, 2^{N-1} - 1]$

Zeropoint Quantization

Practice!

Original Weights (float)

15	6	9
3	-3	12
0	-6	0

$$q = \left\lfloor \frac{r}{3} - 1 \right\rceil$$

Quantized Weights (int3)

4	1	2
0	-2	3
-1	-3	-1

$r_{\min} = -6$ $\quad$ $r_{\max} = 15$

$S = 3$ $\quad$ $Z = -1$

Zeropoint Quantization

Practice!

Quantized Weights (int3)

4	1	2
0	-2	3
-1	-3	1

$$r = 3(q + 1)$$

Reconstructed Weights (float)

15	6	9
3	-3	12
0	-6	0

Zeropoint Quantization

Practice!

Original Weights (float)

15	6	9
3	-3	12
0	-6	0

Reconstructed Weights (float)

15	6	9
3	-3	12
0	-6	0

Zeropoint Quantization

Matmul as an integer-arithmetic-only operation

- We mostly care about the first equation
- S is still stored as a float 😔

$$r = S(q - Z)$$

$$q = \left\lfloor \frac{r}{S} + Z \right\rceil$$

Zeropoint Quantization

$$r = S(q - Z)$$

Matmul as an integer-arithmetic-only operation

Let $A, B \in \mathbb{R}^{n \times n}$. Let $\hat{A}, \hat{B} \in \mathbb{Z}^{n \times n}$ be their quantized versions. Let $C = AB$. Find $\hat{C}$.

Notation: for some matrix M, we will refer to the element in its i-th row and j-th column as m_{ij}.

By the definition of matrix multiplication,

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}$$

Applying our quantization schemes,

$$S_C \left(\hat{c}_{ij} - Z_C \right) = \sum_{k=1}^{N} S_A \left(\hat{a}_{ij} - Z_A \right) S_B \left(\hat{b}_{ij} - Z_B \right)$$

Zeropoint Quantization

Matmul as an integer-arithmetic-only operation

$$S_C \left(\hat{c}_{ij} - Z_C \right) = \sum_{k=1}^{N} S_A \left(\hat{a}_{ij} - Z_A \right) S_B \left(\hat{b}_{ij} - Z_B \right)$$

Factoring constants out of the sum,

$$S_C \left(\hat{c}_{ij} - Z_C \right) = S_A S_B \sum_{k=1}^{N} \left(\hat{a}_{ij} - Z_A \right) \left(\hat{b}_{ij} - Z_B \right)$$

Isolating $\hat{c}_{ij}$ on the left,

$$\hat{c}_{ij} = Z_C + \frac{S_A S_B}{S_C} \sum_{k=1}^{N} \left(\hat{a}_{ij} - Z_A \right) \left(\hat{b}_{ij} - Z_B \right)$$

Zeropoint Quantization

Matmul as an integer-arithmetic-only operation

$$\hat{c}_{ij} = Z_C + \frac{S_A S_B}{S_C} \sum_{k=1}^{N} \left(\hat{a}_{ij} - Z_A\right)\left(\hat{b}_{ij} - Z_B\right)$$

Define $M := \frac{S_A S_B}{S_C}$**. Then,**

$$\hat{c}_{ij} = Z_C + M \sum_{k=1}^{N} \left(\hat{a}_{ij} - Z_A\right)\left(\hat{b}_{ij} - Z_B\right)$$

Empirically, $M \in (0,1)$**. Normalizing for some** $M_0 \in [0.5,1)$**,**

bit shift

$$M = \boxed{2^{-n}} \cdot \boxed{M_0}$$

fixed-point multiplier

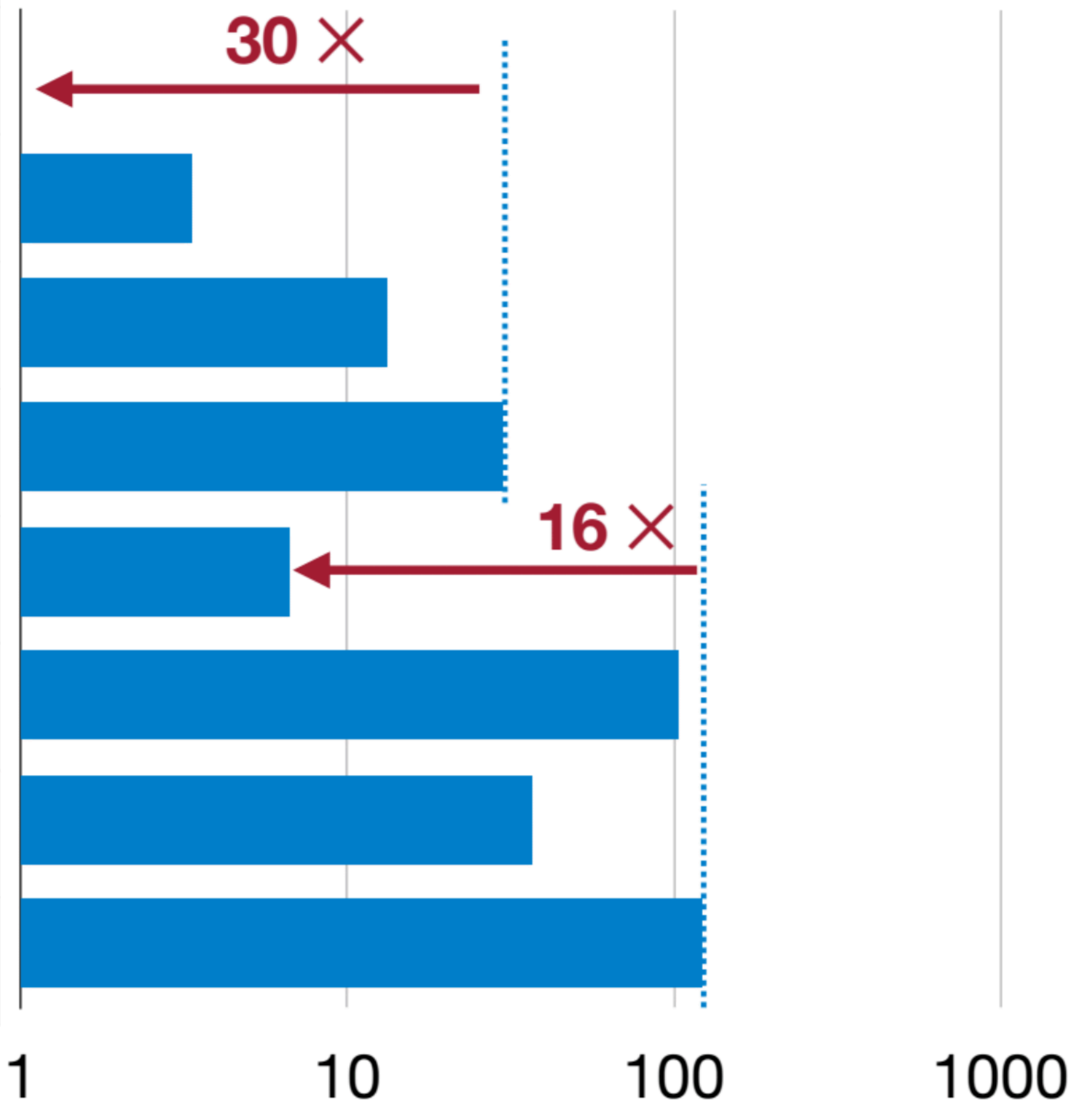
Assuming int32, this will always have 30 bits of relative accuracy. Why?

Zeropoint Quantization

We can now do integer-only matmul!

Operation	Energy [pJ]
8 bit int ADD	0.03
32 bit int ADD	0.1
16 bit float ADD	0.4
32 bit float ADD	0.9
8 bit int MULT	0.2
32 bit int MULT	3.1
16 bit float MULT	1.1
32 bit float MULT	3.7

Rough Energy Cost For Various Operations in 45nm 0.9V



Zeropoint Quantization

Practical Considerations

$$\hat{c}_{ij} = Z_C + M \sum_{k=1}^{N} \left(\hat{a}_{ij} - Z_A\right)\left(\hat{b}_{ij} - Z_B\right)$$

The bulk of the computation occurs in the sum:

$$\left(\hat{a}_{ij} - Z_A\right)\left(\hat{b}_{ij} - Z_B\right) = \sum_{k=1}^{N} \hat{a}_{ij}\hat{b}_{ij} - \hat{a}_{ij}Z_B - \hat{b}_{ij}Z_A + Z_A Z_B$$

int8

int16

CPUs contain `PMADDUBSW`, but most GPUs/TPUs don't.

The zeropoint slows us down tremendously!

Dettmers et al. 2022, "LLM.int8()"

Absolute Maximum Quantization

Also known as "symmetric quantization"

- What if get rid of the zeropoint?

$$r = S(q - Z) \quad \Rightarrow \quad r = Sq$$

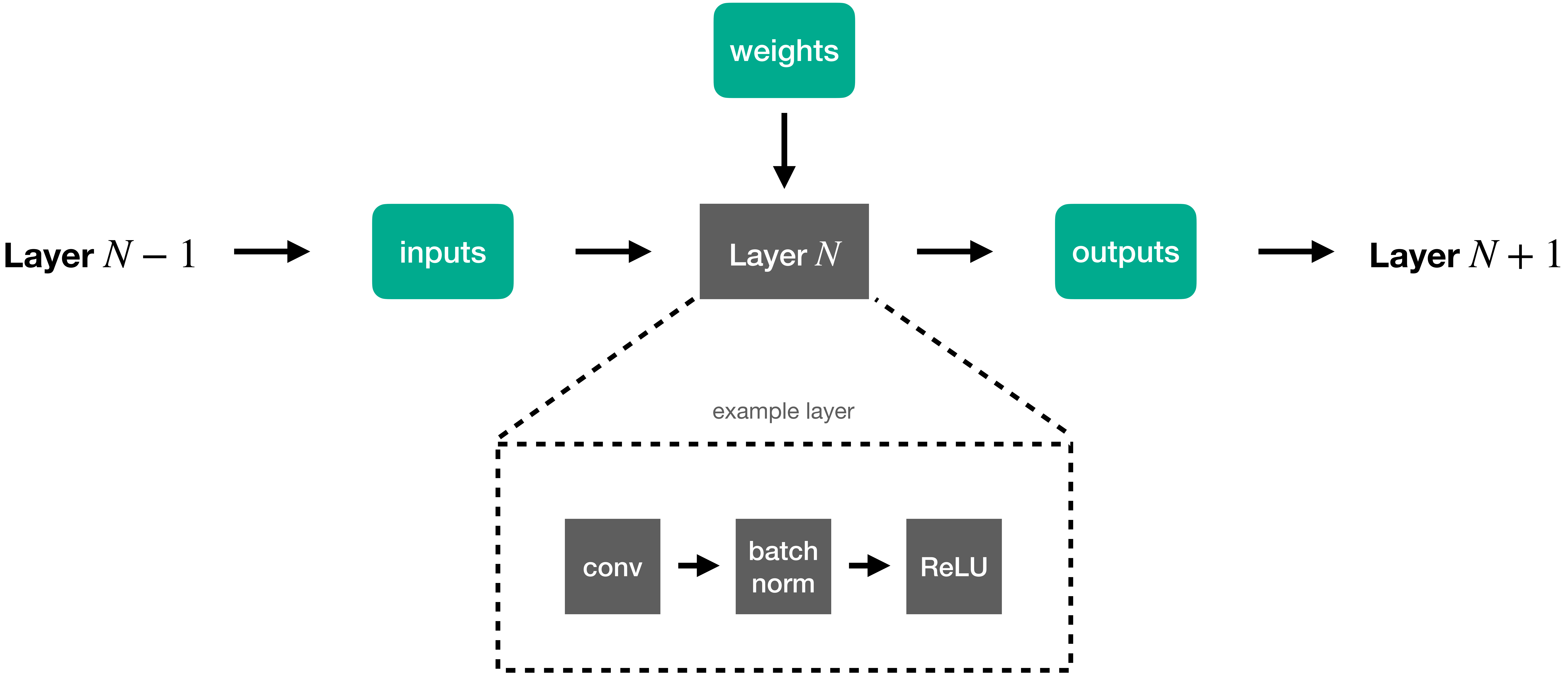
- This forces a symmetric quantization scheme. Why?

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \approx \frac{2 \cdot r_{absmax}}{2 \cdot q_{\max}} = \frac{r_{absmax}}{q_{\max}}$$

Deep Learning Quantization Paradigms

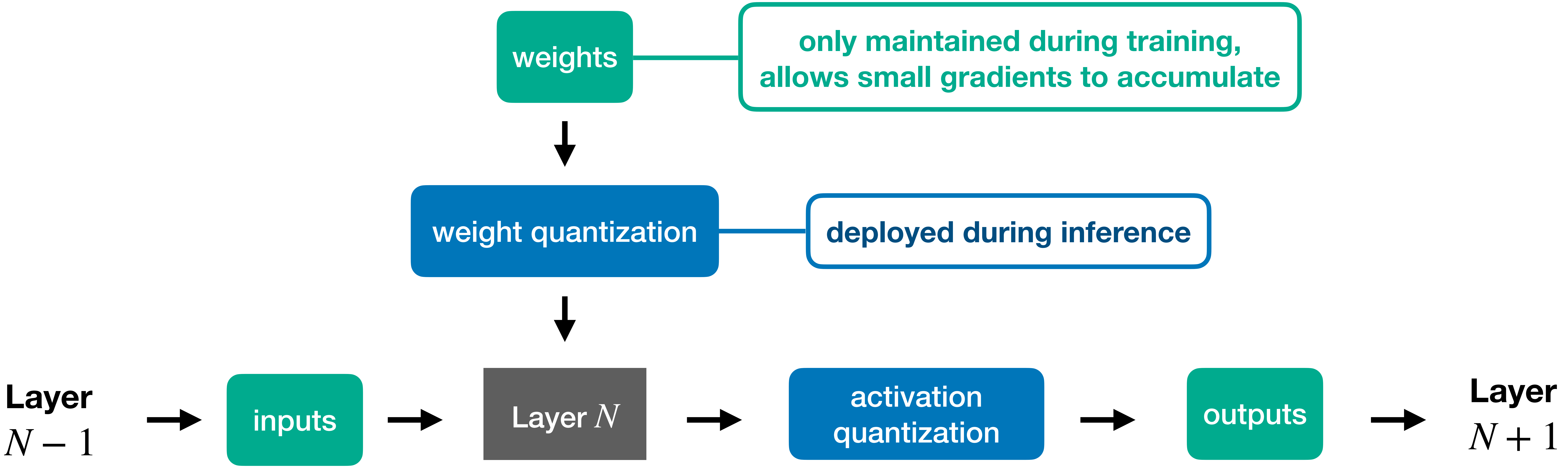
Quantization-Aware Training

Simulated / Fake Quantization



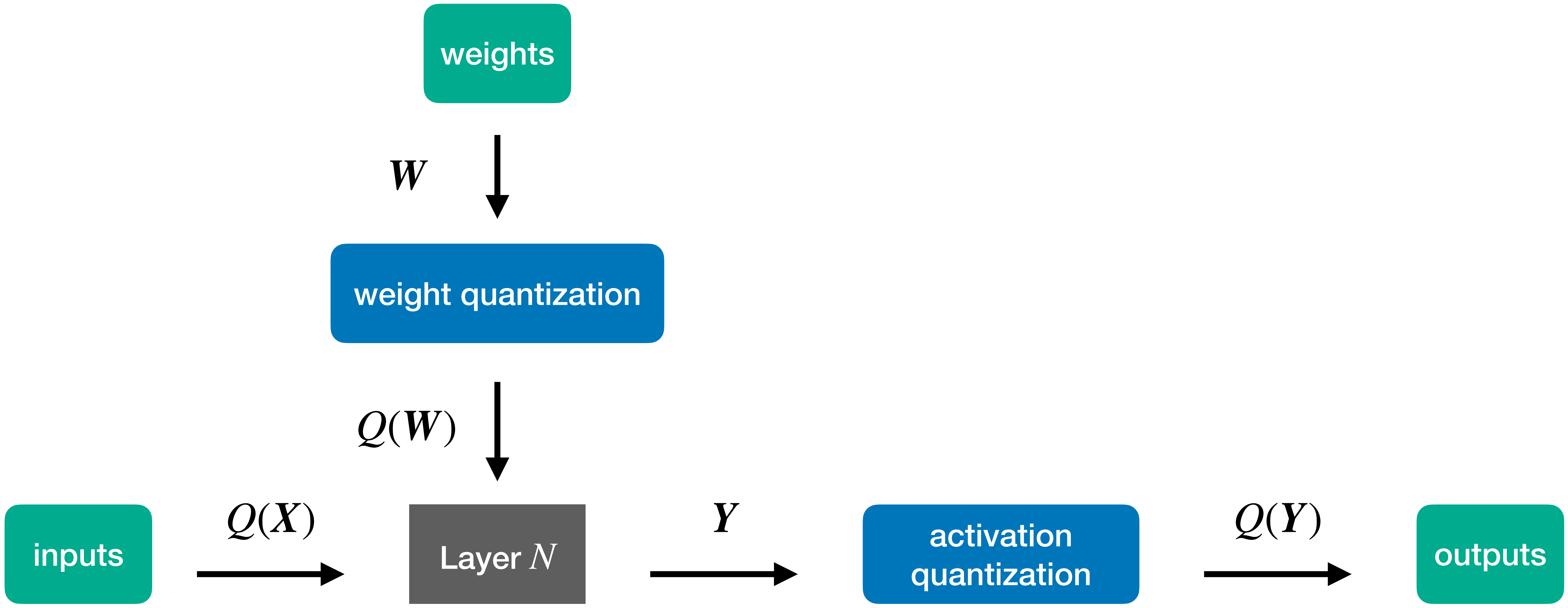
Quantization-Aware Training

Simulated / Fake Quantization



Quantization-Aware Training

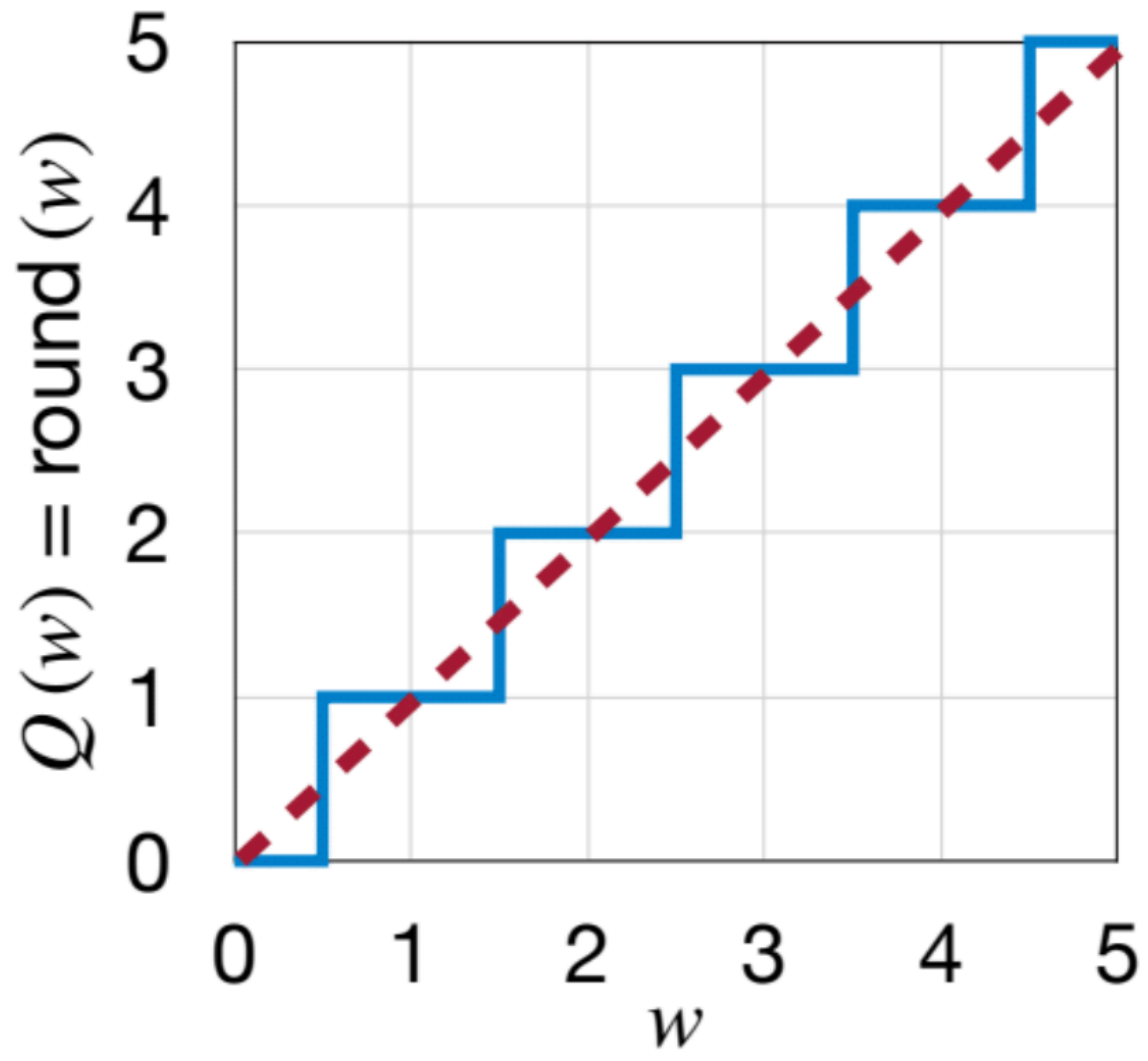
Simulated / Fake Quantization



Quantization-Aware Training

Straight-Through Estimator (STE)

- Quantization is a step function ⇒ gradient is almost always 0



Quantization-Aware Training

Straight-Through Estimator (STE)

- Quantization is a step function ⇒ gradient is almost always 0
- STE passes gradients along “as if it had been the identity function”

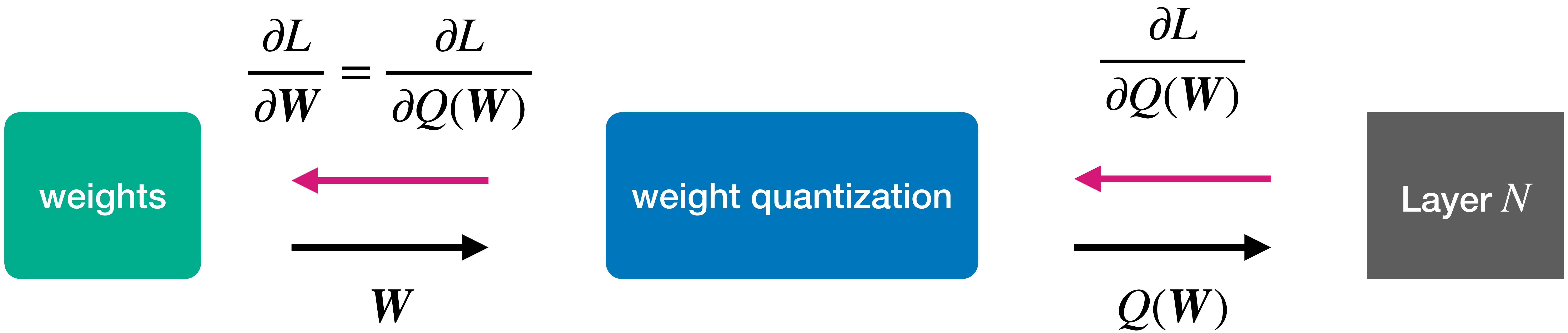
$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial Q(\mathbf{W})} \cdot \underbrace{\frac{\partial Q(\mathbf{W})}{\partial \mathbf{W}}}_{\text{chain rule}} = \frac{\partial L}{\partial Q(\mathbf{W})} \underbrace{\cdot 1}_{\text{straight-through estimator}} = \frac{\partial L}{\partial Q(\mathbf{W})}$$

Bengio et al. 2013, “Estimating or Propagating Gradients”

Quantization-Aware Training

Straight-Through Estimator (STE)

- Quantization is a step function ⇒ gradient is almost always 0
- STE passes gradients along "as if it had been the identity function"



Bengio et al. 2013, "Estimating or Propagating Gradients"

Quantization-Aware Training

Practical Considerations

- Quite effective, especially useful for really large models
  - Requires training (or re-training) the model 😔
- Also called “Mixed Precision Training”

Krishnamoorthi 2018, "Quantizing deep convolutional networks"

Post-Training Quantization

Practical Considerations

Model Size	Model Name	Asymmetric PTQ	Symmetric PTQ	Asymmetric QAT	Symmetric QAT	Floating Point
3.5M	MobileNet-v2	0.1%	69.8%	70.9%	71.1%	71.9%
25M	ResNet-50	75%	75%	75%	75%	75.6%
60M	ResNet-152	76.1%	76%	76%	76%	77.8%

Smaller models suffer from PQT, likely because they have a smaller representational capacity

Quantization Libraries

& their underlying PTQ methods

LLM.int8()

Seminal work in quantization for LLMs

- Implemented in `bitsandbytes`
- Integrated with 🤗 Transformers

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

previously discussed methods

what we're about to discuss

perplexity scores

Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

Absmax Quantization

$$r = S(q - Z) \quad \Rightarrow \quad r = Sq \quad \Rightarrow \quad q = \left\lfloor \frac{r}{S} \right\rceil$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \approx \frac{r_{absmax}}{q_{\max}} = \frac{r_{absmax}}{127}$$

int8 $\Rightarrow q_{\max} = 127$

LLM.int8()

Absmax Quantization

$$r = S(q - Z) \quad \Rightarrow \quad r = Sq \quad \Rightarrow \quad q = \left\lfloor \frac{r}{S} \right\rceil$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \approx \frac{r_{absmax}}{q_{\max}} = \frac{r_{absmax}}{127}$$

quantized → real

$$\frac{r_{absmax}}{127}$$

real → quantized

$$\frac{127}{r_{absmax}}$$

we will refer to this value as s

Dettmers et al. 2022, “LLM.int8()”

LLM.int8()

Absmax Quantization

- $\frac{1}{s}$: quantized → real

- s : real → quantized

quantized → real

$$\frac{r_{absmax}}{127}$$

real → quantized

$$\frac{127}{r_{absmax}}$$

Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

int8 matmul with fp16 inputs and outputs

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{s \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$

$$\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} = \boldsymbol{C}_{f16}$$

from previous slide

LLM.int8()

int8 matmul with fp16 inputs and outputs

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{s \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$

$$
\begin{aligned}
\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} = \boldsymbol{C}_{f16} &\approx \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot \boldsymbol{C}_{\boxed{i32}} \\
&\approx \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot \boldsymbol{X}_{i8}\boldsymbol{W}_{i8}
\end{aligned}
$$

Why int32?

LLM.int8()

int8 matmul with fp16 inputs and outputs

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{s \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$

$$
\begin{aligned}
\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} = \boldsymbol{C}_{f16} &\approx \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot \boldsymbol{C}_{i32} \\
&\approx \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot \boldsymbol{X}_{i8}\boldsymbol{W}_{i8} \\
&= \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot Q(\boldsymbol{X}_{f16})\, Q(\boldsymbol{W}_{f16})
\end{aligned}
$$

Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

Vector-wise Quantization

- More scaling constants ⇒ improved quantization error
- Matrix multiplication is a series of dot products
- Idea: one scaling factor for each row of $\boldsymbol{X}$ and each column of $\boldsymbol{W}$

LLM.int8()

Vector-wise Quantization

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{s \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$
  - Assign a different scaling constant $s_{x_{f16}}$ to each row of $\boldsymbol{X}_{f16}$
  - Assign a different scaling constant $s_{w_{f16}}$ to each col of $\boldsymbol{W}_{f16}$

LLM.int8()

Vector-wise Quantization

$$\text{first entry of } C = (\text{first row of } X) \cdot (\text{first col of } W)$$

$$\approx \frac{1}{s \text{ for first row of } X \cdot s \text{ for first col of } W} \cdot Q\,(\text{first row of } X) \cdot Q\,(\text{first col of } W)$$

$$\text{first entry of } C = \begin{bmatrix} 2 & 6 & -15 \end{bmatrix} \bullet \begin{bmatrix} 7 \\ 2 \\ -4 \end{bmatrix}$$

$$\approx \frac{1}{8.46\bar{6} \cdot 18.143} \bullet Q(\begin{bmatrix} 2 & 6 & -15 \end{bmatrix}) \bullet Q(\begin{bmatrix} 7 \\ 2 \\ -4 \end{bmatrix})$$

derived via $\frac{127}{absmax}$ formula

LLM.int8()

Vector-wise Quantization

$$= \frac{1}{8.46\bar{6} \cdot 18.143} \bullet Q\left(\begin{bmatrix} 2 & 6 & -15 \end{bmatrix}\right) \bullet Q\left(\begin{bmatrix} 7 \\ 2 \\ -4 \end{bmatrix}\right)$$

$$= \frac{1}{8.46\bar{6} \cdot 18.143} \bullet \begin{bmatrix} \lfloor 2 \cdot 8.46\bar{6} \rceil & \lfloor 6 \cdot 8.46\bar{6} \rceil & \lfloor -15 \cdot 8.46\bar{6} \rceil \end{bmatrix} \bullet \begin{bmatrix} \lfloor 18.143 \cdot 7 \rceil \\ \lfloor 18.143 \cdot 2 \rceil \\ \lfloor 18.143 \cdot -4 \rceil \end{bmatrix}$$

$$= \frac{1}{153.61} \bullet \begin{bmatrix} 17 & 51 & -127 \end{bmatrix} \bullet \begin{bmatrix} 127 \\ 36 \\ -73 \end{bmatrix}$$

$$= \frac{1}{153.61} \cdot \big(17(127) + 51(36) - 127(-73)\big) = \frac{1}{153.61} \cdot \big(13266\big) = 86.36$$

for reference,
the correct result is 86

LLM.int8()

Vector-wise Quantization

$\boldsymbol{X}_{f16}$

2	6	-15
-7	9	12
3	-10	8
8	1	3
-1	11	9

$\boldsymbol{W}_{f16}$

7	-3
2	8
-4	0

LLM.int8()

Vector-wise Quantization

real → quantized

$$s_{f16} = \frac{127}{r_{absmax}}$$

$$\boldsymbol{X}_{f16} \quad \begin{bmatrix} 2 & 6 & -15 \\ -7 & 9 & 12 \\ 3 & -10 & 8 \\ 8 & 1 & 3 \\ -1 & 11 & 9 \end{bmatrix} \xrightarrow[\text{absmax}]{\text{row-wise}} \begin{bmatrix} 15 \\ 12 \\ 10 \\ 8 \\ 11 \end{bmatrix} \xrightarrow{\text{compute } s} \begin{bmatrix} 8.466 \\ 10.583 \\ 12.7 \\ 15.875 \\ 11.545 \end{bmatrix}$$

$$\boldsymbol{W}_{f16} \quad \begin{bmatrix} 7 & -3 \\ 2 & -8 \\ -4 & 0 \end{bmatrix} \xrightarrow[\text{absmax}]{\text{col-wise}} \begin{bmatrix} 7 & 8 \end{bmatrix} \xrightarrow{\text{compute } s} \begin{bmatrix} 18.143 & 15.875 \end{bmatrix}$$

LLM.int8()

Vector-wise Quantization

X_{f16}

2	6	-15
-7	9	12
3	-10	8
8	1	3
-1	11	9

s_{f16}

8.466
10.583
12.7
15.875
11.545

$$Q(X_{f16}) = \lfloor s_{f16} X_{f16} \rceil$$

17	51	-127
-74	95	127
38	-127	102
127	16	3
-12	127	104

W_{f16}

7	-3
2	-8
-4	0

s_{f16}

18.143	15.875

$$Q(W_{f16}) = \lfloor s_{f16} W_{f16} \rceil$$

127	-48
36	-127
-73	0

LLM.int8()

Vector-wise Quantization

- Recall our dequantization equation

$$\boldsymbol{C}_{f16} \approx \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot Q(\boldsymbol{X}_{f16})\, Q(\boldsymbol{W}_{f16})$$

- We now have multiple s
- How can we efficiently dequantize?

LLM.int8()

Vector-wise Quantization

- How can we efficiently dequantize?
  - ij-th entry of C is the dot product of the i-th row of $\boldsymbol{X}$ & the j-th col of $\boldsymbol{W}$
  - i-th row of $\boldsymbol{X}$ was quantized with i-th entry of $\boldsymbol{s}_{f16}$
  - j-th col of $\boldsymbol{W}$ was quantized with j-th entry of $\boldsymbol{s}_{f16}$

What should the ij-th entry of C be dequantized by?

LLM.int8()

Vector-wise Quantization

- How can we efficiently dequantize?
  - ij-th entry of $\boldsymbol{C}$ is the dot product of the i-th row of $\boldsymbol{X}$ & the j-th col of $\boldsymbol{W}$
  - i-th row of $\boldsymbol{X}$ was quantized with i-th entry of $\boldsymbol{s}_{f16}$
  - j-th col of $\boldsymbol{W}$ was quantized with j-th entry of $\boldsymbol{s}_{f16}$

What should C be dequantized by?

$$\boldsymbol{s}_{f16} \otimes \boldsymbol{s}_{f16}$$

LLM.int8()

Vector-wise Quantization

$$
\mathbf{s}_{f16} \otimes \mathbf{s}_{f16} = \begin{bmatrix} 8.466 \\ 10.583 \\ 12.7 \\ 15.875 \\ 11.545 \end{bmatrix} \otimes \begin{bmatrix} 18.143 & 15.875 \end{bmatrix} = \begin{bmatrix} 153.610 & 134.408 \\ 192.012 & 168.010 \\ 230.414 & 201.613 \\ 288.018 & 252.016 \\ 209.468 & 183.284 \end{bmatrix}
$$

LLM.int8()
Vector-wise Quantization

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{b \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$

$$\underbrace{\boldsymbol{C}_{f16}}_{b \times o \textbf{ matrix}} = \underbrace{\frac{1}{\boldsymbol{s}_{f16} \otimes \boldsymbol{s}_{f16}}}_{b \times o \textbf{ matrix}} \cdot \underbrace{Q\left(\boldsymbol{X}_{f16}\right) Q\left(\boldsymbol{W}_{f16}\right)}_{b \times o \textbf{ matrix}}$$

elementwise multiplication

LLM.int8()

Vector-wise Quantization

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{b \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$

$$\boldsymbol{C}_{f16} = \frac{1}{\boldsymbol{s}_{f16} \otimes \boldsymbol{s}_{f16}} \cdot \overset{\lfloor s_{f16}\boldsymbol{X}_{f16} \rceil}{\overset{\shortparallel}{Q\left(\boldsymbol{X}_{f16}\right)}} \; \overset{\lfloor s_{f16}\boldsymbol{W}_{f16} \rceil}{\overset{\shortparallel}{Q\left(\boldsymbol{W}_{f16}\right)}}$$

LLM.int8()

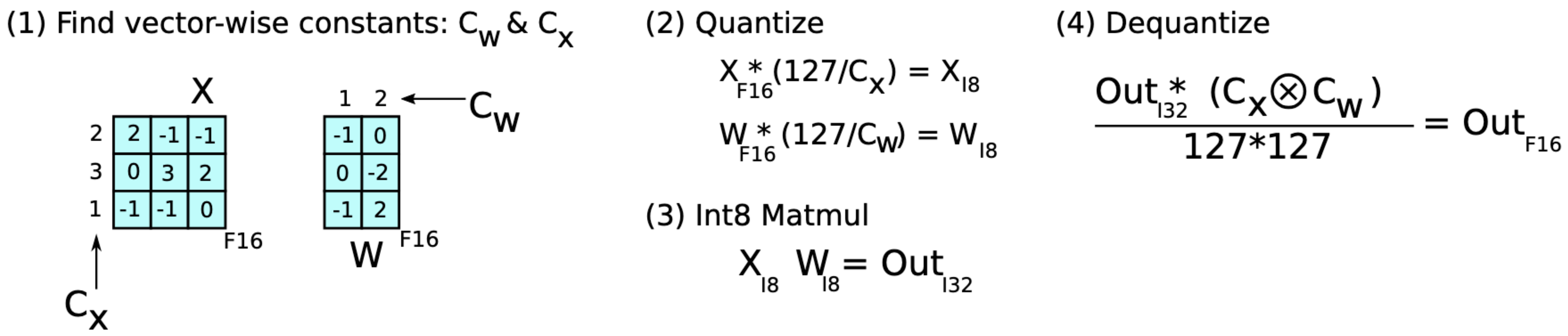
Vector-wise Quantization

- Given inputs $\boldsymbol{X}_{f16} \in \mathbb{R}^{b \times h}$ and $\boldsymbol{W}_{f16} \in \mathbb{R}^{h \times o}$, we compute $\boldsymbol{C}_{f16} \in \mathbb{R}^{s \times o}$

$$\boldsymbol{C}_{f16} = \boldsymbol{S} \cdot \overset{\left\lfloor s_{f16}\boldsymbol{X}_{f16} \right\rceil}{\overset{\shortparallel}{Q\left(\boldsymbol{X}_{f16}\right)}} \; \overset{\left\lfloor s_{f16}\boldsymbol{W}_{f16} \right\rceil}{\overset{\shortparallel}{Q\left(\boldsymbol{W}_{f16}\right)}}$$

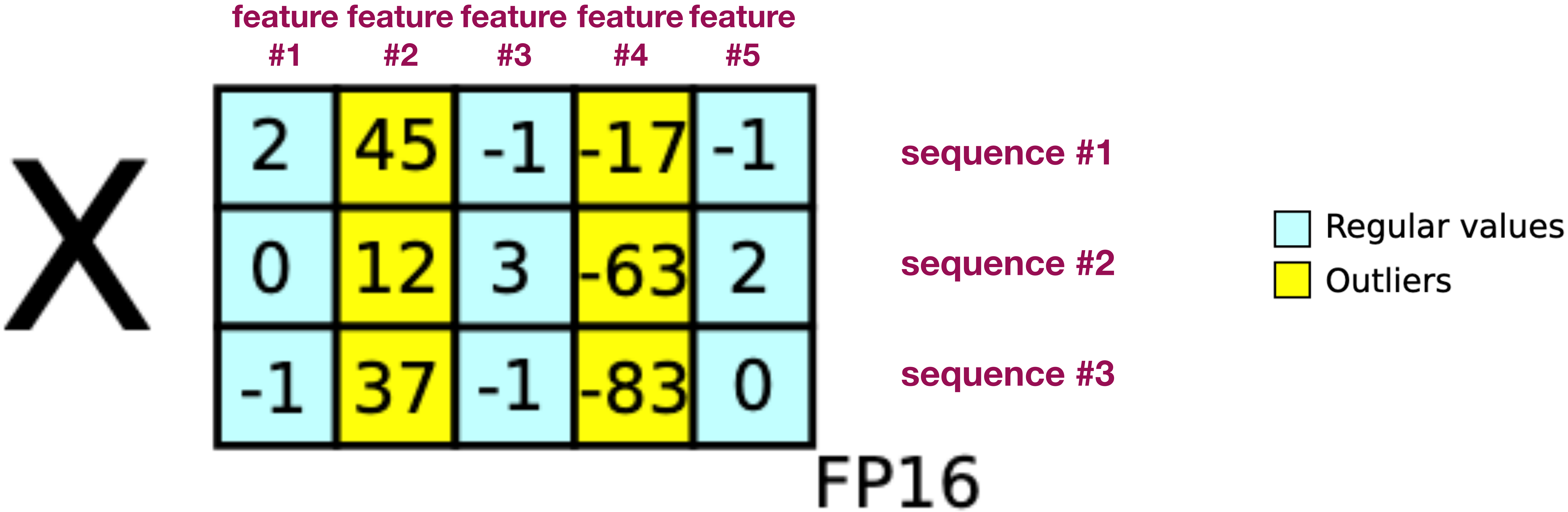
LLM.int8()

Vector-wise Quantization



Dettmers et al. 2022, “LLM.int8()”

LLM.int8()
Outliers

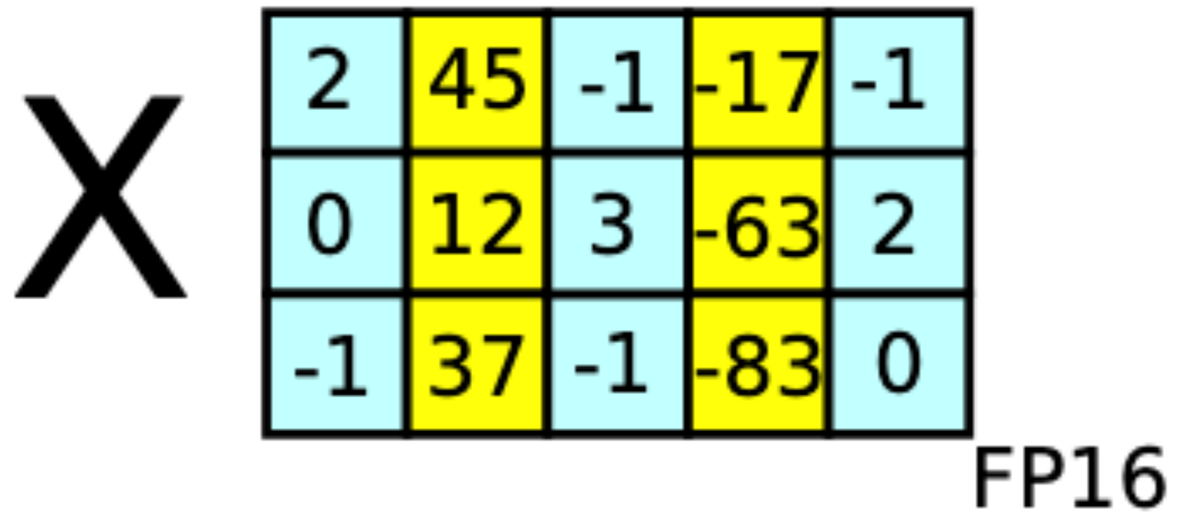


Dettmers et al. 2022, “LLM.int8()”

LLM.int8()

Outliers

- Empirically, large (>6B) transformer models have **outliers**:
  - Large magnitude features (columns)
  - Extremely important for performance
  - Require high quantization precision



Dettmers et al. 2022, "LLM.int8()"

LLM.int8()
Outliers

- Empirically, large transformer models have…
  - 99.9% regular features — medium quantization precision OK
  - 0.01% outlier features — require high quantization precision

Great, we already employ vector-wise quantization, right?

Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

Outliers

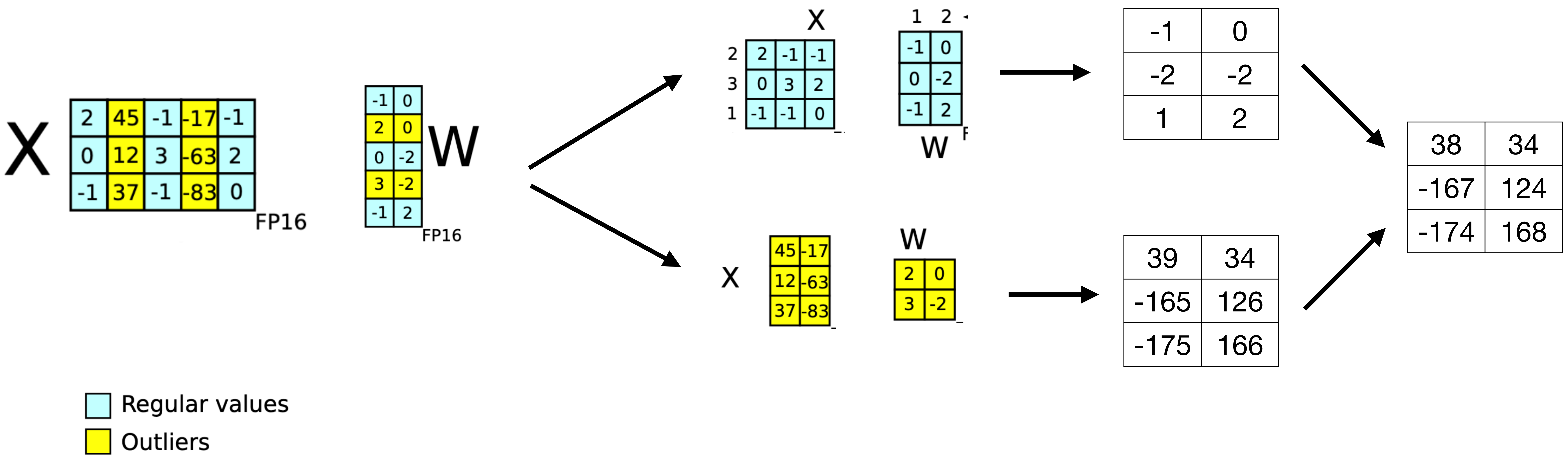
- Vector-wise quantization assigns different scaling factors for rows in X
- Outliers occur in columns in X

Vector-wise quantization doesn't help

LLM.int8()

Mixed-precision Decomposition

- What if we handled the normal features and outlier features separately?



Dettmers et al. 2022, "LLM.int8()"

Aside: Einstein Notation

This is non-standard

$$X^2W^2 + X^4W^4$$

LLM.int8()

Mixed-precision Decomposition

- Notation for handling outliers O “separately” (still with normal fp16 matmul)

$$\boldsymbol{C}_{f16} = \sum_{h \in O} \boldsymbol{X}^h_{f16} \boldsymbol{W}^h_{f16} + \sum_{h \notin O} \boldsymbol{X}^h_{f16} \boldsymbol{W}^h_{f16}$$

LLM.int8()

Mixed-precision Decomposition

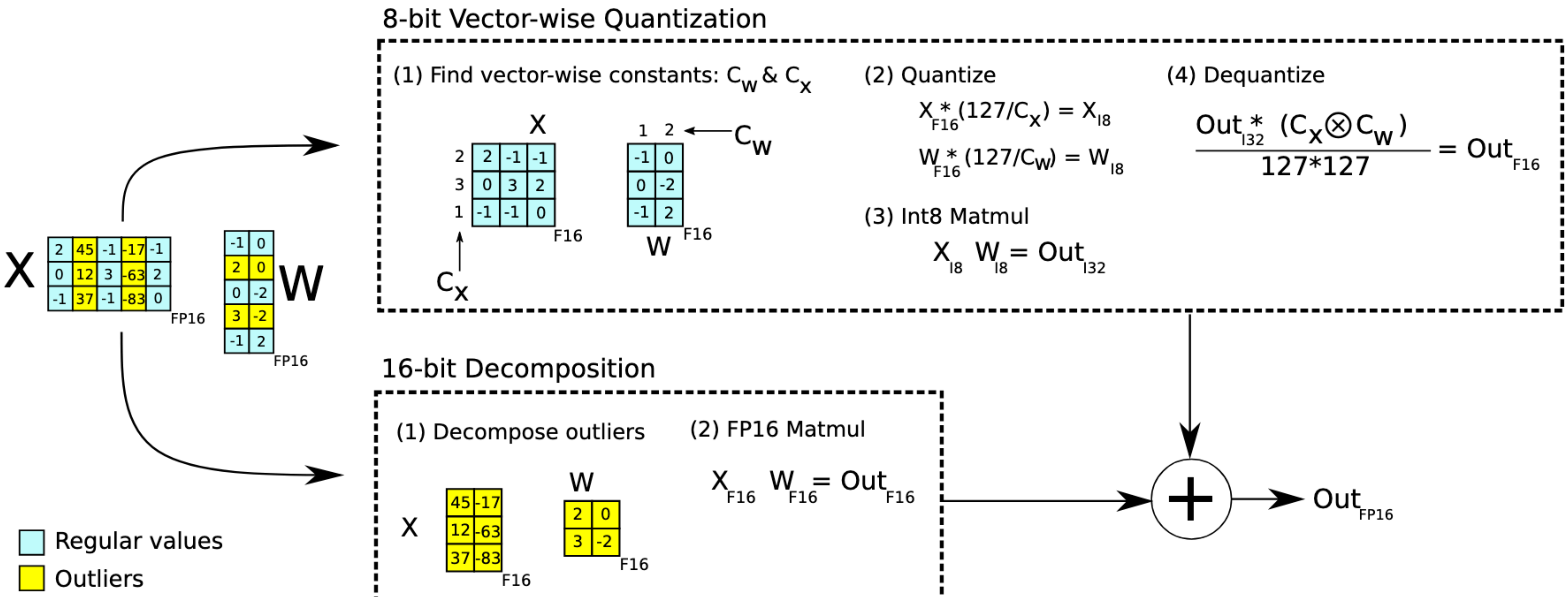
$$C_{f16} = S \cdot Q\left(X_{f16}\right) Q\left(W_{f16}\right)$$

- What if we handled the normal features and outlier features separately?

$$C_{f16} = \sum_{h \in O} X^h_{f16} W^h_{f16} + S \cdot \sum_{h \notin O} X^h_{i8} W^h_{i8}$$

Dettmers et al. 2022, "LLM.int8()"

LLM.int8()



Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

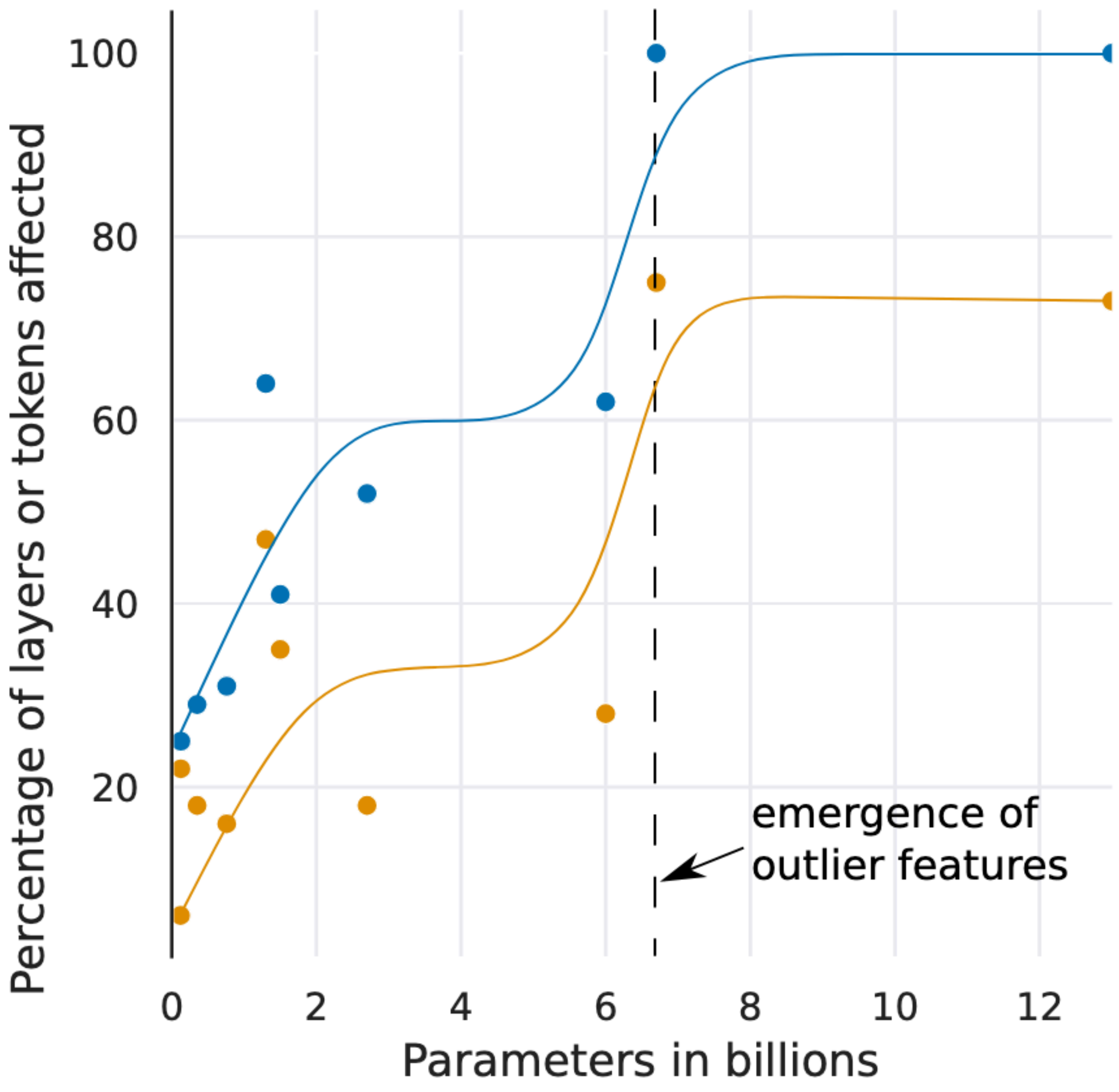
Outliers are critical to transformer performance

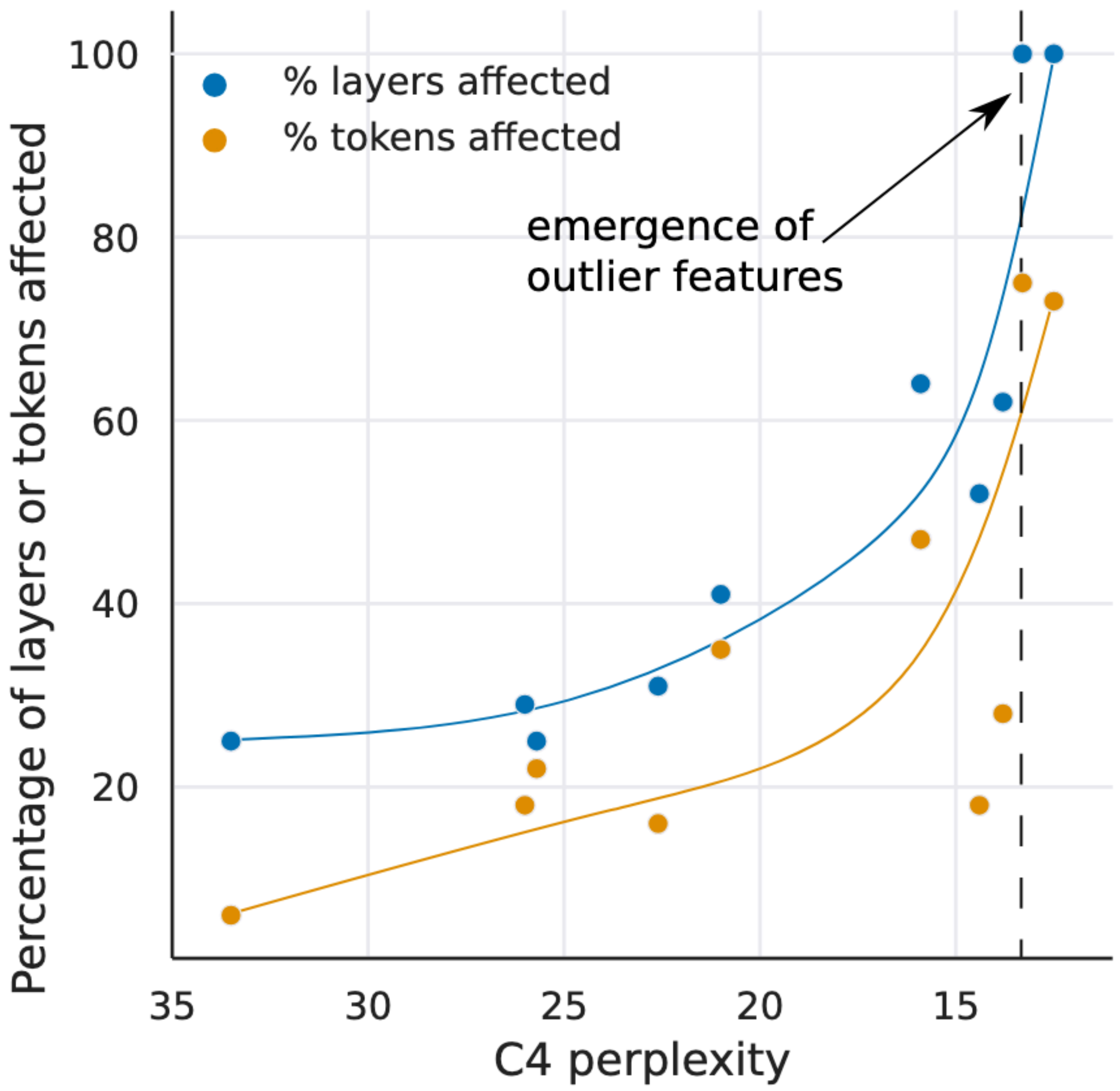
- Removing **<7 outlier features** causes…
  - top-1 softmax probability cut in half
  - validation perplexity increases 6-10x
- Removing **7 random features** causes…
  - top-1 softmax probability decreases by 0.3%
  - validation perplexity increases by 0.1%

Dettmers et al. 2022, “LLM.int8()”

LLM.int8()

Phase Shift

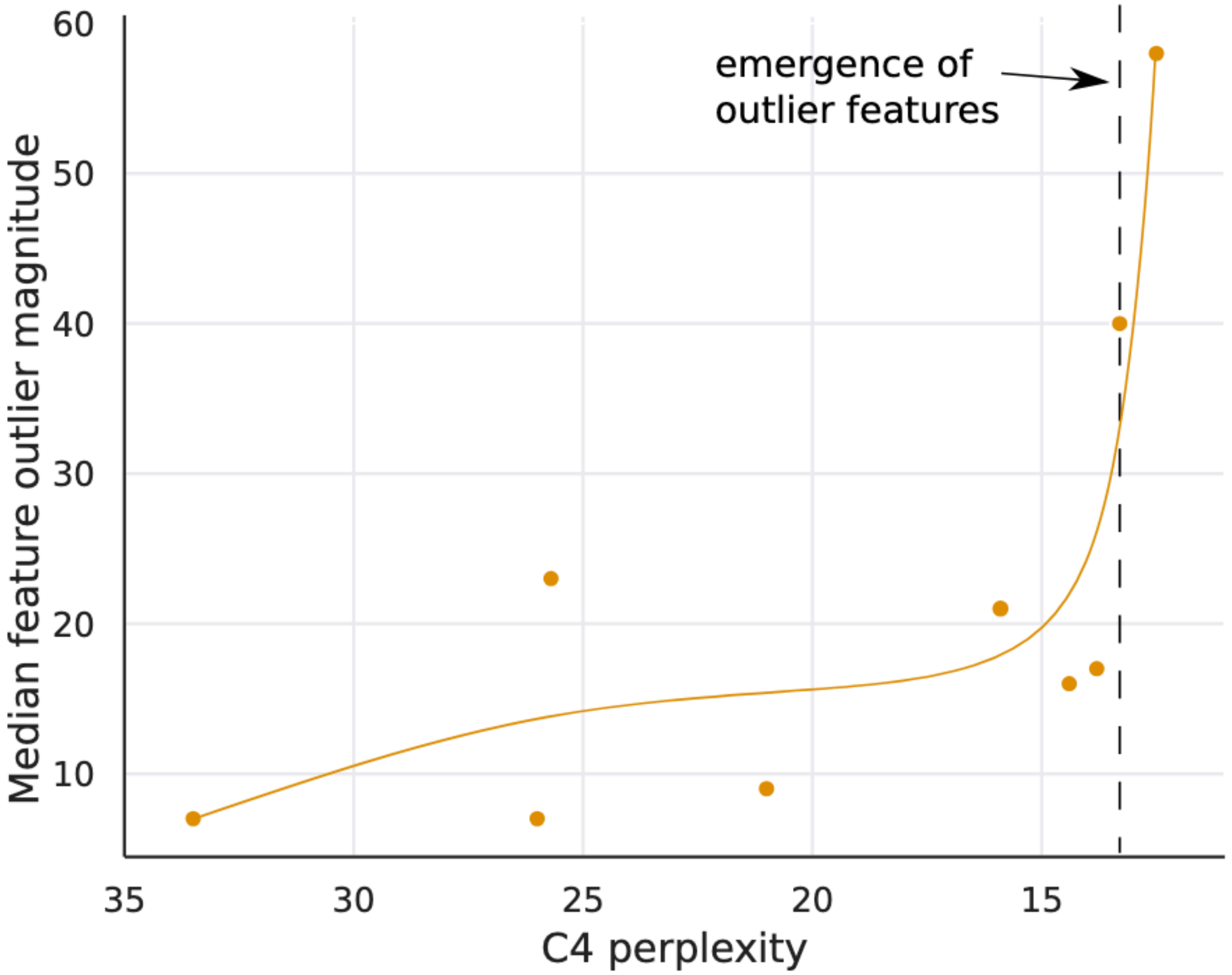


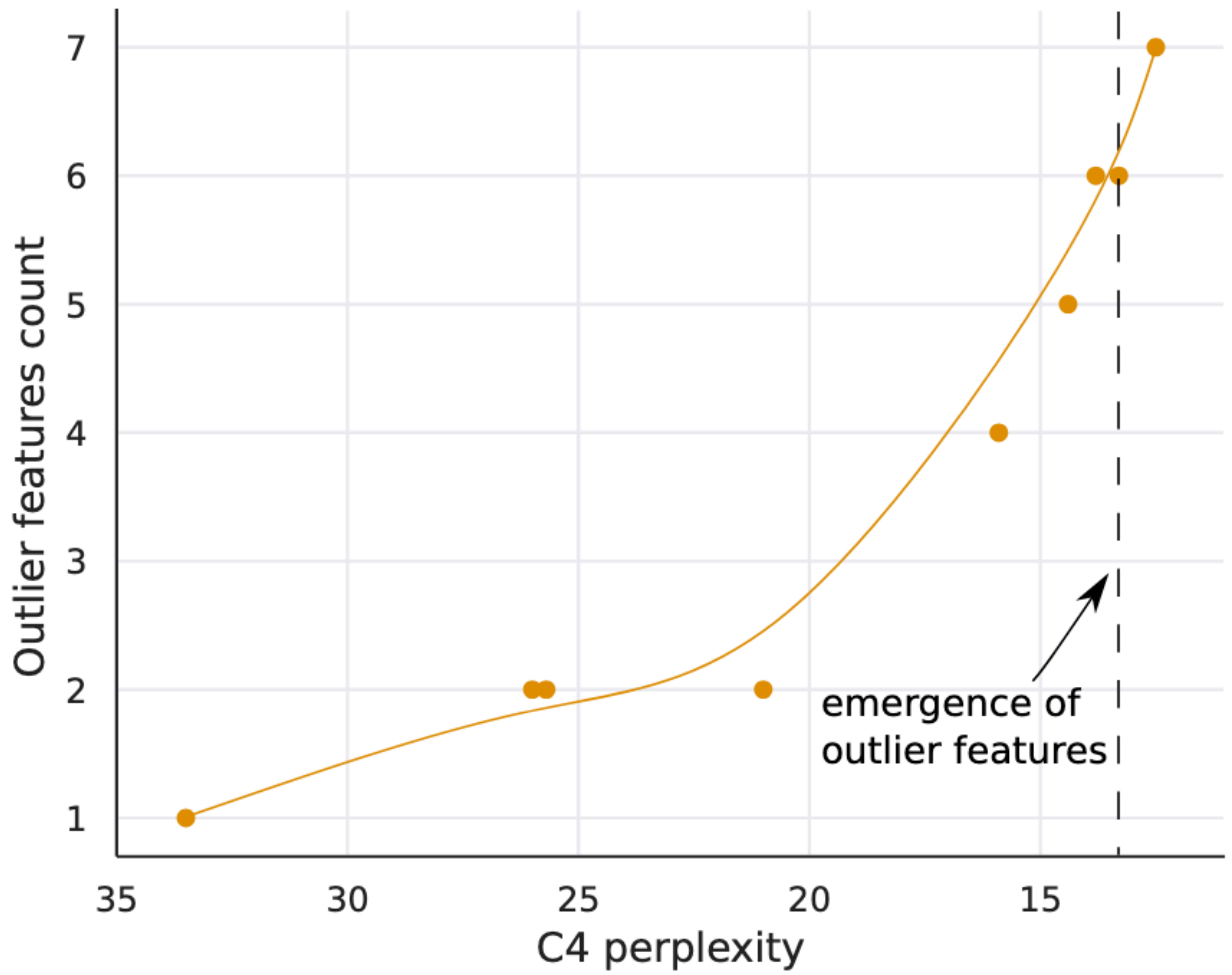


Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

Phase Shift

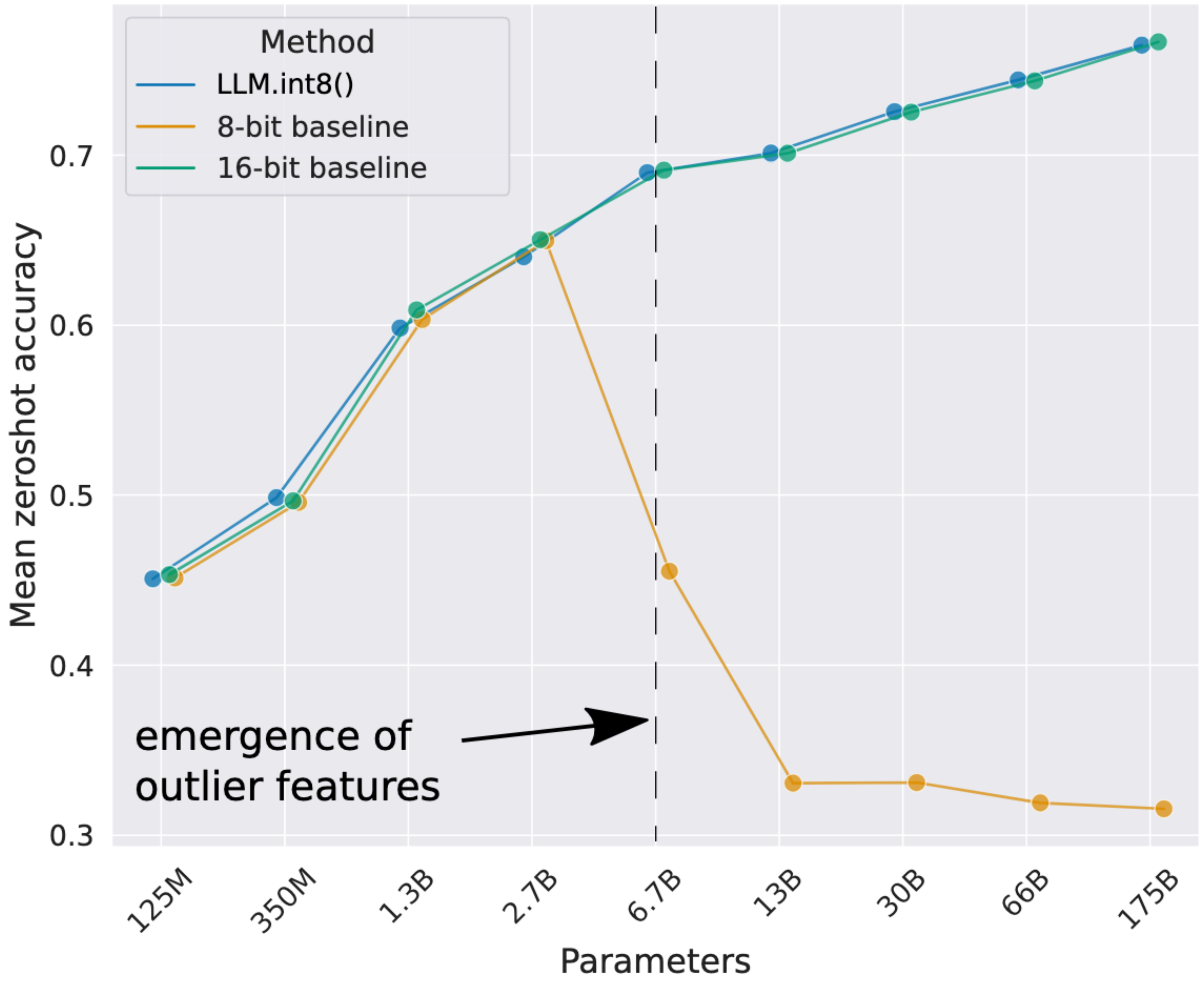




Dettmers et al. 2022, "LLM.int8()"

LLM.int8()

Phase Shift

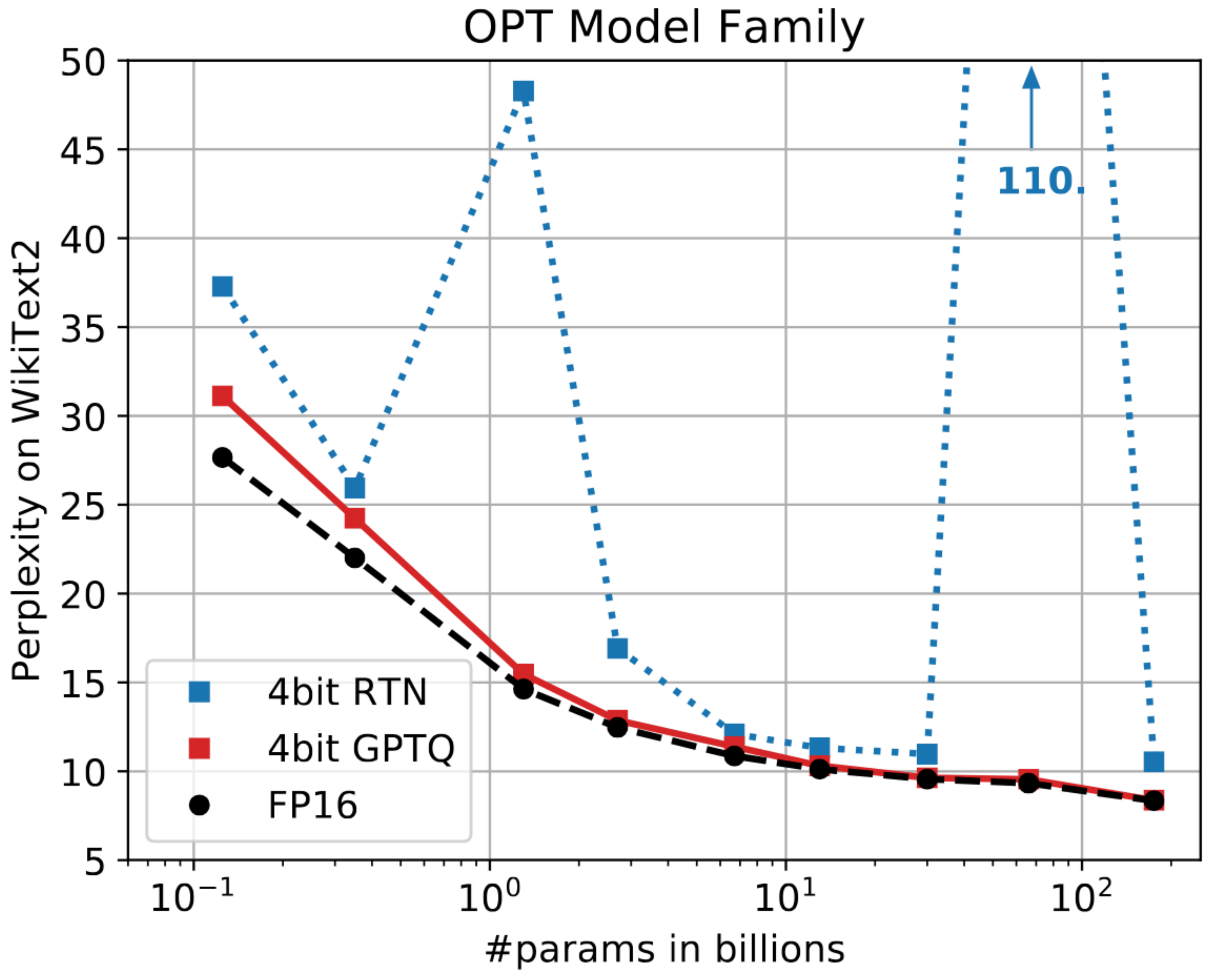


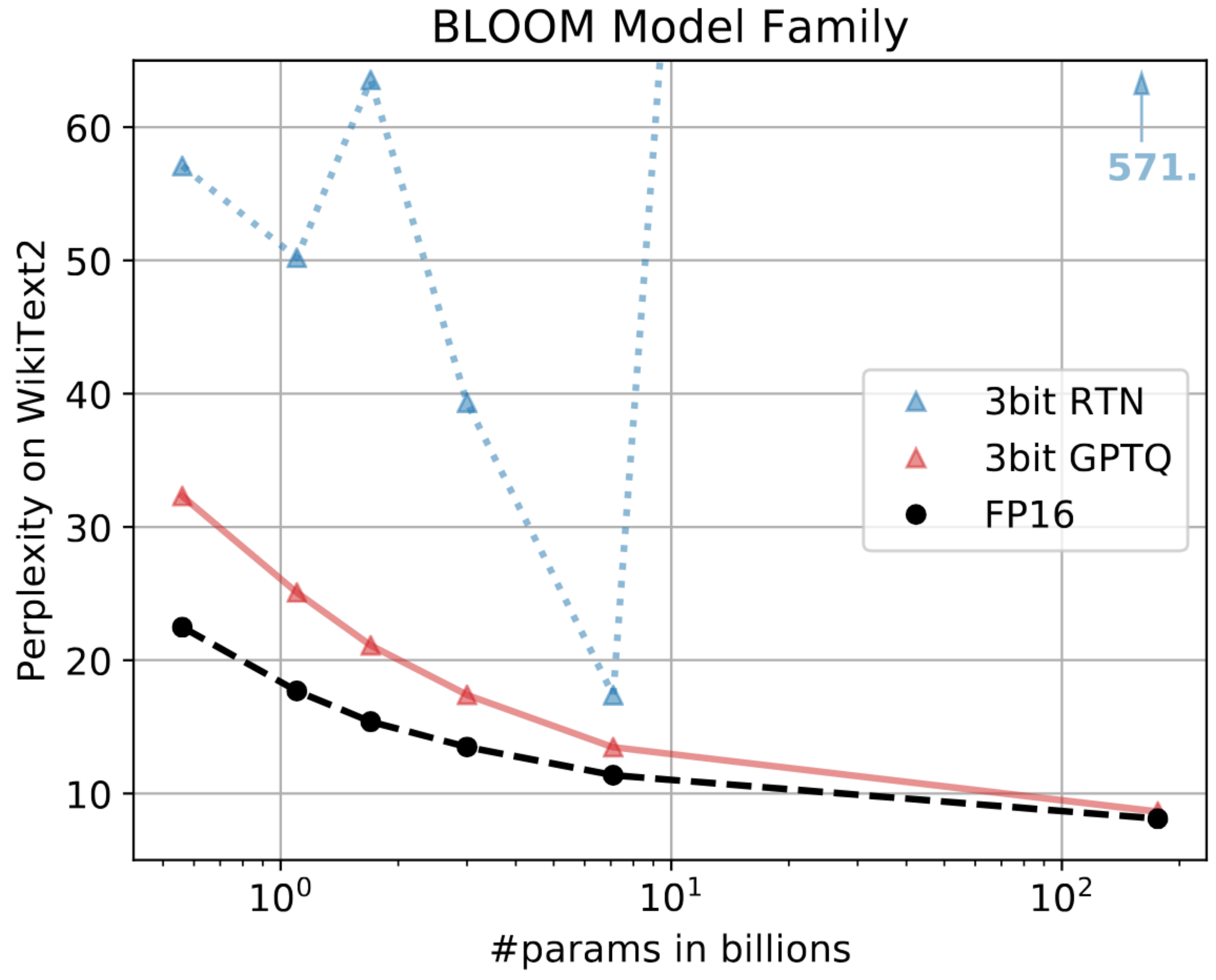
Dettmers et al. 2022, “LLM.int8()”

GPTQ

4 Bit Quantization

- Integrated with 🤗 Transformers
- First paper to break the 8 bit quantization barrier





Frantar et al. 2022, "GPTQ"

AWQ

4 Bit Quantization… but simpler :)

- 1% of weights are “salient weights”
  - Similar insight to LLM.int8()
- Can be found via activation distribution instead of weights
  - Intuition: input features with large magnitude are important
- Neat trick: multiply weight by s, input by s^{-1}

Lin et al. 2024, “AWQ”

Conclusion

Summary

Quantization is like printers.

- Powerful tool to reduce memory footprint of models with many use-cases
  - Frontier labs use quantization to train very large models
  - Academics & home users use quantization to fit large models on their GPUs
- Awe-inspiring results

References

Recommended further reading is bolded

- Song Han, MIT 6.5940 TinyML and Efficient Deep Learning Computing, 2024, https://hanlab.mit.edu/courses/2024-fall-65940
- Sreenivas et al., “LLM Pruning and Distillation in Practice: The Minitron Approach,” December 10, 2024, https://arxiv.org/pdf/2408.11796
- Pankaj Gupta and Philip Kiely, “FP8: Efficient model inference with 8-bit floating point numbers,” https://www.baseten.co/blog/fp8-efficient-model-inference-with-8-bit-floating-point-numbers/
- Tim Dettmers, “8-bit Methods for Efficient Deep Learning,” March 2023, https://www.youtube.com/watch?v=jyOqtw4ry2w
- **Jacob et al., “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” CPVR 2018, https://arxiv.org/pdf/1712.05877**
- Raghuraman Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” June 2018, https://arxiv.org/pdf/1806.08342
- **Dettmers et al., “LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale,” NeurIPS 2022, https://arxiv.org/pdf/2208.07339**
- Yoshua Bengio, Nicholas L´eonard and Aaron Courville, “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation,” August 15, 2013, https://arxiv.org/pdf/1308.3432
- Frantar et al., “GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers,” ICLR 2023, https://arxiv.org/pdf/2210.17323
- **Lin et al., “AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration,” MLSys 2024, https://arxiv.org/pdf/2306.00978**