



# Efficient Inference with Transformers

May 5, 2025

# Recap: RAM requirements for LLMs

- I have a 7 billion parameter model. How much RAM do I need to host it?
- RAM Usage = 7 billion params  $\times$  2 bytes per parameter = 14 billion bytes\*
  - Kilo bytes =  $2^{10} = 1,024$   $\approx$  1 thousand bytes
  - Mega bytes =  $2^{20} = 1,048,576$   $\approx$  1 million bytes
  - Giga bytes =  $2^{30} = 1,073,741,824$   $\approx$  1 billion bytes
  - Tera bytes =  $2^{40} = 1,099,511,627,776$   $\approx$  1 trillion bytes
- Unlike regular ML models where a parameter is 4 bytes (32 bit float), LLMs are usually 2 bytes

\* Actually we need a bit more for the intermediate computations

# Recap: RAM requirements for LLMs

- I have a 7 billion parameter model. How much RAM do I need to host it?
- RAM Usage = 7 billion params  $\times$  **2 bytes per parameter** = 14 billion bytes\*
  - Kilo bytes =  $2^{10} = 1,024$   $\approx 1 \text{ thousand bytes}$
  - Mega bytes =  $2^{20} = 1,048,576$   $\approx 1 \text{ billion bytes}$
  - Giga bytes =  $2^{30} = 1,073,741,824$   $\approx 1 \text{ billion bytes}$
  - Tera bytes =  $2^{40} = 1,099,511,627,776$   $\approx 1 \text{ trillion bytes}$
- Unlike regular ML models where a parameter is 4 bytes (32 bit float), LLMs are usually 2 bytes

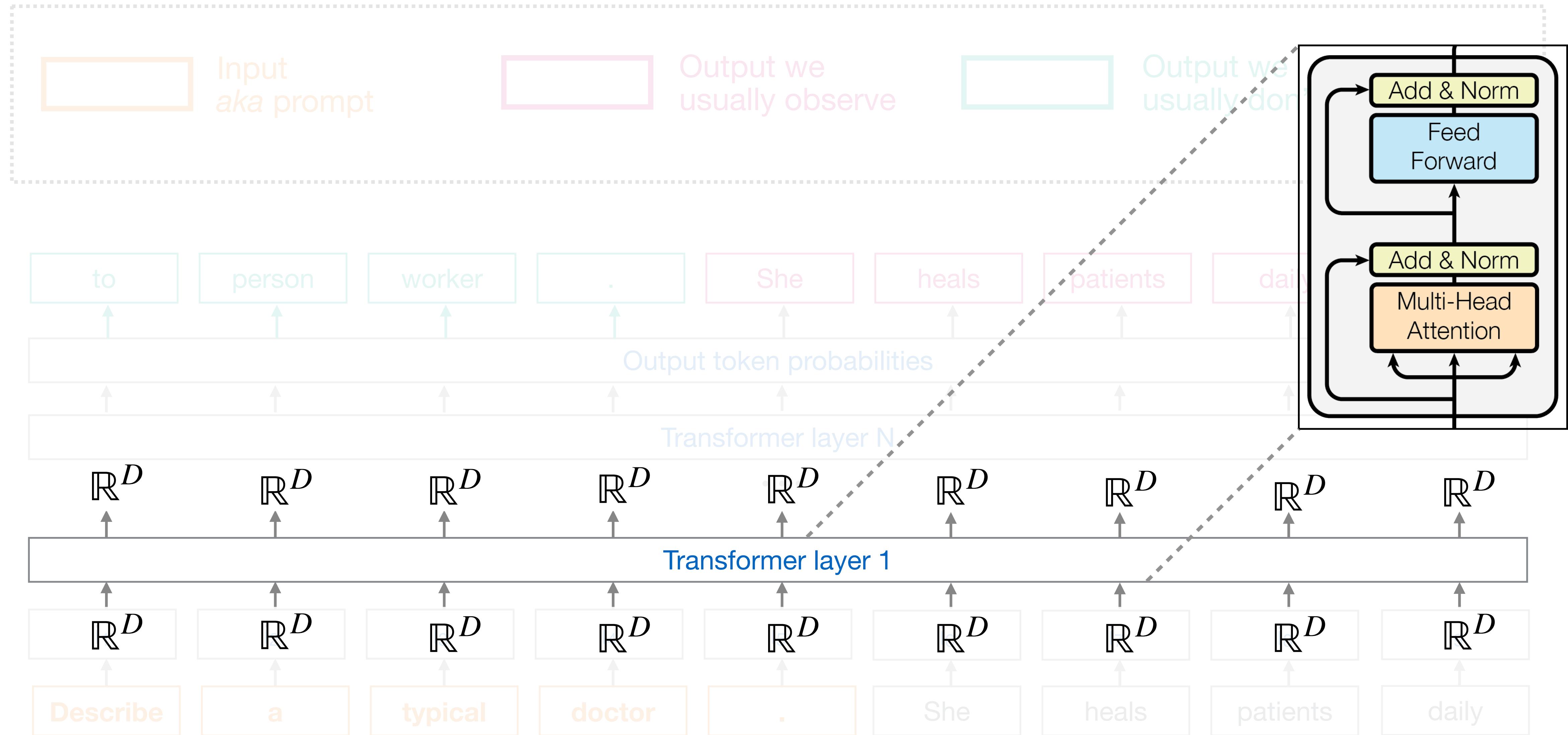
\* Actually we need a bit more for the intermediate computations

# Recap: RAM requirements for LLMs

- I have a 7 billion parameter model. How much RAM do I need to host it?
- RAM Usage = 7 billion params  $\times$  **2 bytes per parameter** = 14 billion bytes\*
  - Kilo bytes =  $2^{10} = 1,024$   $\approx 1 \text{ thousand bytes}$
  - Mega bytes =  $2^{20} = 1,048,576$   $\approx 1 \text{ billion bytes}$
  - Giga bytes =  $2^{30} = 1,073,741,824$   $\approx 1 \text{ trillion bytes}$
  - Tera bytes =  $2^{40} = 1,099,511,627,776$   $\approx 1 \text{ petabytes}$
- Unlike regular ML models where a parameter is 4 bytes (32 bit float), LLMs are usually 2 bytes

\* Actually we need a bit more for the intermediate computations

# Recap: Every layer consists of large parameter matrices

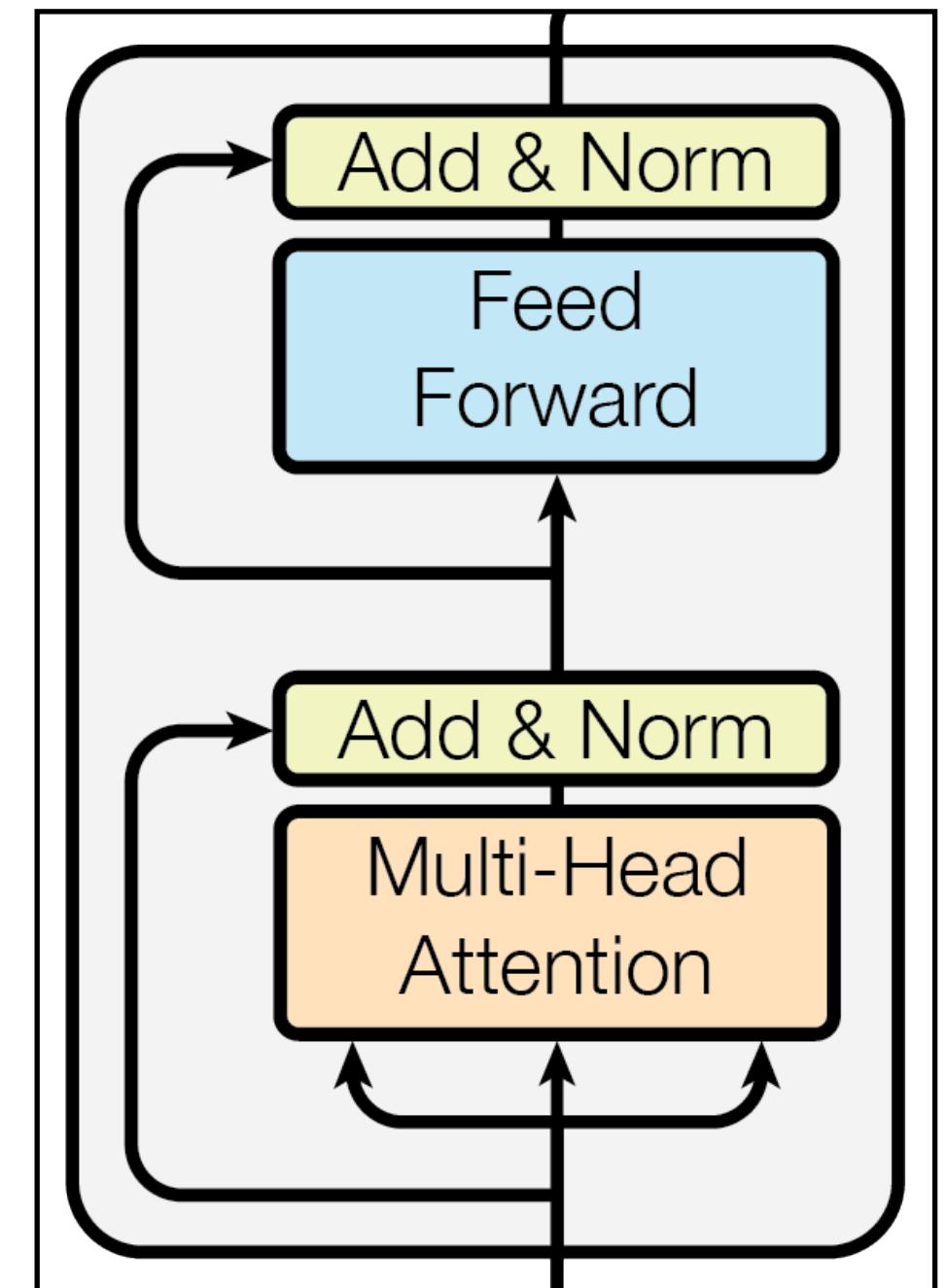


# A closer look at model parameters

- Most computations within the model consist of

$$\text{output} = \text{input} \times \mathbf{W} + \mathbf{B}$$

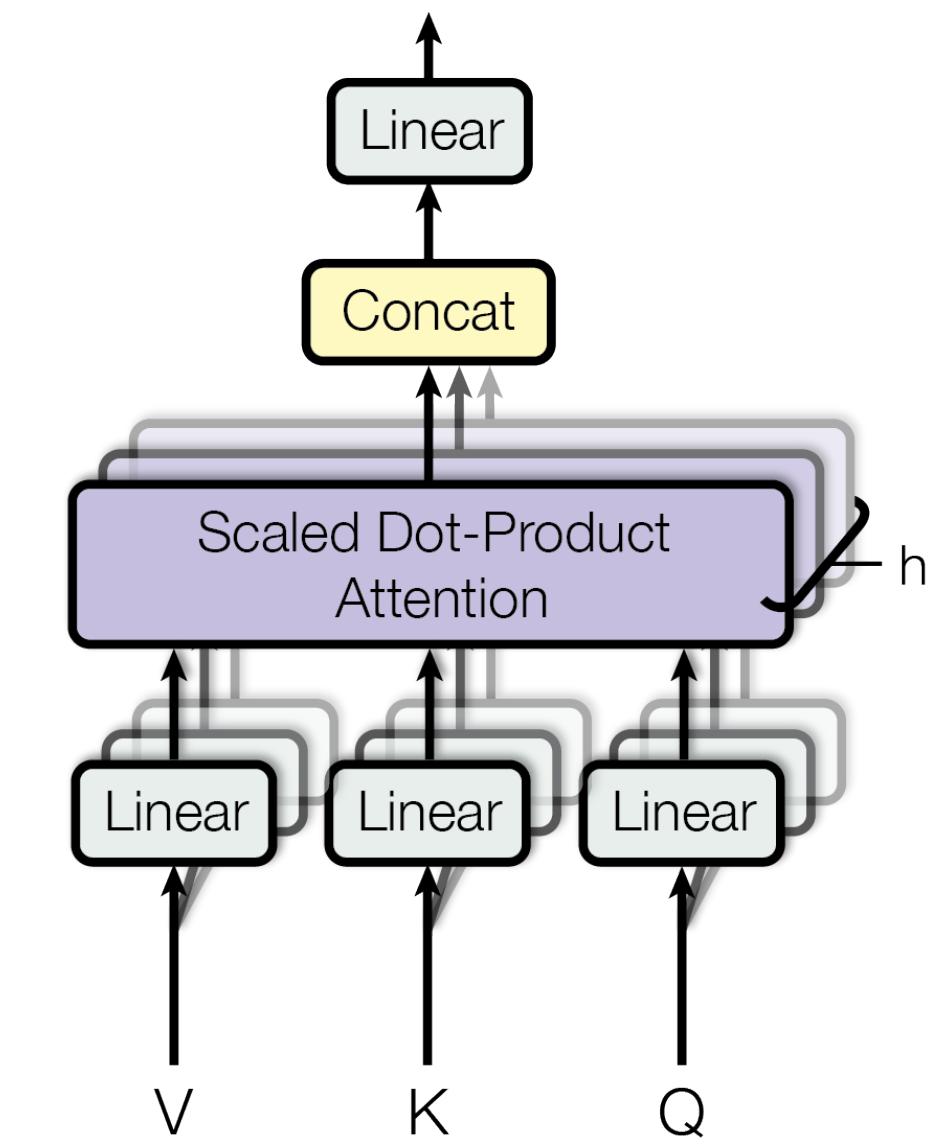
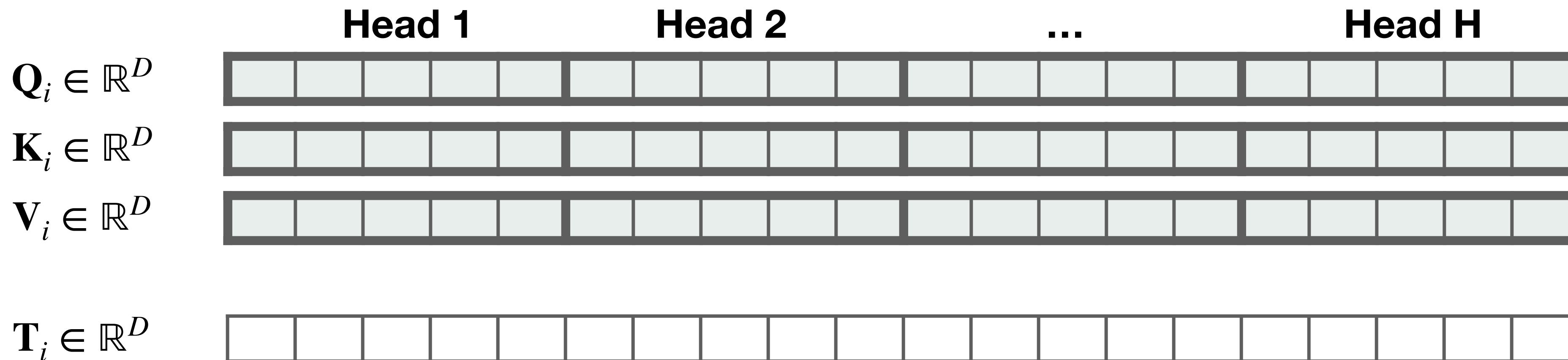
- $\mathbf{W} \in \mathbb{R}^{D_1 \times D_2}$  and  $\mathbf{B} \in \mathbb{R}^{D_2}$  are the model **parameters**



# Example # 1: Q, K and V for attention

$$\text{output} = \text{input} \times \mathbf{W} + \mathbf{B}$$

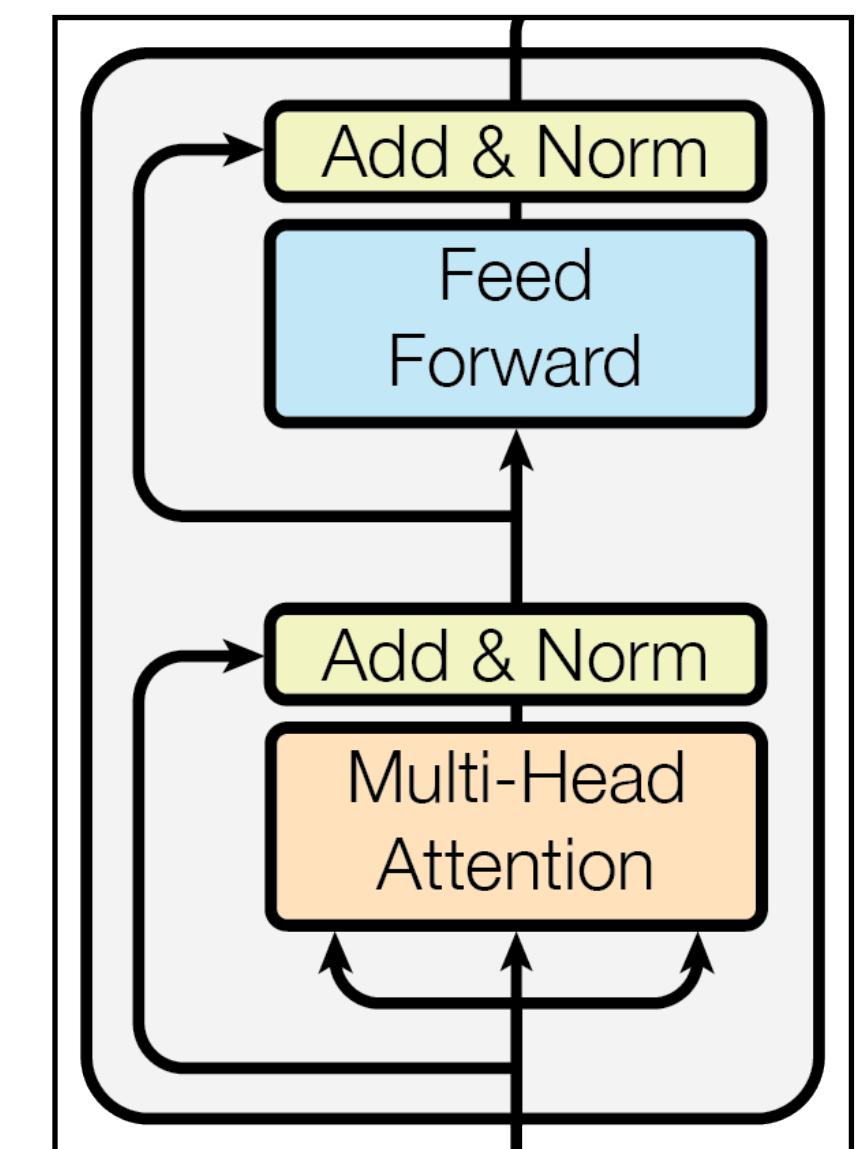
- Computing **query (Q)**, **key (K)** and **value (V)** in  $i^{\text{th}}$  layer
- $\mathbf{Q}_i = \mathbf{T}_i \mathbf{W}_i^Q$  where  $\mathbf{W}_i^Q \in \mathbf{R}^{D \times \frac{D}{H}}$  and H is the number of heads
- $\mathbf{B} = 0$ , in other words, no **B**
- Similar computation for **K** and **V**



# Example # 2: The feed forward block

$$\text{output} = \text{input} \times \mathbf{W} + \mathbf{B}$$

- $FFN(\mathbf{X}) = \max(0, \mathbf{X}\mathbf{W}_1 + \mathbf{B}_1)\mathbf{W}_2 + \mathbf{B}_2$
- $\mathbf{W}_1 \in \mathbb{R}^{D \times 4D}$  and  $\mathbf{B}_1 \in \mathbb{R}^{4D}$
- $\mathbf{W}_2 \in \mathbb{R}^{4D \times D}$  and  $\mathbf{B}_2 \in \mathbb{R}^D$



# Weights and Biases

A typical weight matrix

|      |       |       |       |      |       |       |       |       |       |
|------|-------|-------|-------|------|-------|-------|-------|-------|-------|
| 0.52 | 1.60  | 3.97  | 8.52  | 0.52 | 1.60  | 3.97  | 8.52  | 8.52  | 1.60  |
| 5.56 | 3.32  | 5.21  | 0.25  | 5.56 | 3.32  | 5.21  | 0.25  | 0.25  | 3.32  |
| 0.98 | 9.58  | 10.20 | 0.54  | 0.98 | 9.58  | 10.20 | 0.54  | 0.54  | 9.58  |
| 5.25 | 8.56  | 12.30 | 54.21 | 5.25 | 8.56  | 12.30 | 54.21 | 54.21 | 8.56  |
| 0.85 | 63.65 | 8.25  | 5.32  | 0.85 | 63.65 | 8.25  | 5.32  | 5.32  | 63.65 |
| 0.52 | 1.60  | 3.97  | 8.52  | 0.52 | 1.60  | 3.97  | 8.52  | 8.52  | 1.60  |
| 5.56 | 3.32  | 5.21  | 0.25  | 5.56 | 3.32  | 5.21  | 0.25  | 0.25  | 3.32  |
| 0.98 | 9.58  | 10.20 | 0.54  | 0.98 | 9.58  | 10.20 | 0.54  | 0.54  | 9.58  |
| 5.25 | 8.56  | 12.30 | 54.21 | 5.25 | 8.56  | 12.30 | 54.21 | 54.21 | 8.56  |
| 0.85 | 63.65 | 8.25  | 5.32  | 0.85 | 63.65 | 8.25  | 5.32  | 5.32  | 63.65 |

A typical bias vector

|      |      |       |       |      |      |       |       |       |      |
|------|------|-------|-------|------|------|-------|-------|-------|------|
| 5.25 | 8.56 | 12.30 | 54.21 | 5.25 | 8.56 | 12.30 | 54.21 | 54.21 | 8.56 |
|------|------|-------|-------|------|------|-------|-------|-------|------|

# Weights and Biases

A typical weight matrix

|      |       |       |       |      |       |       |       |       |       |
|------|-------|-------|-------|------|-------|-------|-------|-------|-------|
| 0.52 | 1.60  | 3.97  | 8.52  | 0.52 | 1.60  | 3.97  | 8.52  | 8.52  | 1.60  |
| 5.56 | 3.32  | 5.21  | 0.25  | 5.56 | 3.32  | 5.21  | 0.25  | 0.25  | 3.32  |
| 0.98 | 9.58  | 10.20 | 0.54  | 0.98 | 9.58  | 10.20 | 0.54  | 0.54  | 9.58  |
| 5.25 | 8.56  | 12.30 | 54.21 | 5.25 | 8.56  | 12.30 | 54.21 | 54.21 | 8.56  |
| 0.85 | 63.65 | 8.25  | 5.32  | 0.85 | 63.65 | 8.25  | 5.32  | 5.32  | 63.65 |
| 0.52 | 1.60  | 3.97  | 8.52  | 0.52 | 1.60  | 3.97  | 8.52  | 8.52  | 1.60  |
| 5.56 | 3.32  | 5.21  | 0.25  | 5.56 | 3.32  | 5.21  | 0.25  | 0.25  | 3.32  |
| 0.98 | 9.58  | 10.20 | 0.54  | 0.98 | 9.58  | 10.20 | 0.54  | 0.54  | 9.58  |
| 5.25 | 8.56  | 12.30 | 54.21 | 5.25 | 8.56  | 12.30 | 54.21 | 54.21 | 8.56  |
| 0.85 | 63.65 | 8.25  | 5.32  | 0.85 | 63.65 | 8.25  | 5.32  | 5.32  | 63.65 |

A typical bias vector

|      |      |       |       |      |      |       |       |       |      |
|------|------|-------|-------|------|------|-------|-------|-------|------|
| 5.25 | 8.56 | 12.30 | 54.21 | 5.25 | 8.56 | 12.30 | 54.21 | 54.21 | 8.56 |
|------|------|-------|-------|------|------|-------|-------|-------|------|

A single parameter  
Represented using a predetermined # of bits

# Representing float parameters



- Until the models got too large, it was typical to present parameters in 32 bits
- Let us first understand how the 32-bit representation work
- Then we will study the optimizations that led to other representations

# Representing float parameters



**Sign**  
(1 bit)

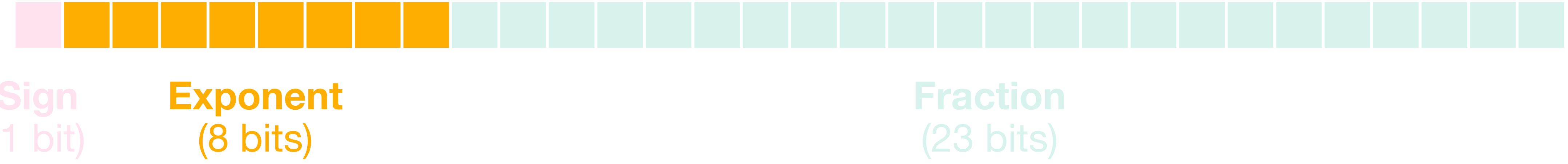
**Exponent**  
(8 bits)

**Fraction**  
(23 bits)

## The sign bit

- Set to 1 if we are representing a negative number
- 0 for a positive number

# Anatomy of a float



# Exponent (E)

- 8 bits  $\rightarrow 2^8 = 256$  values [0, 255]







# Anatomy of a float



**Sign**  
(1 bit)

**Exponent**  
(8 bits)

**Fraction**  
(23 bits)

## Fraction

- 23 bits available (24 in practice\*)
- Can represent  $2^{24} = 16,777,216$  values
- So a bit more than 7 digits

\* In practice 24 bits due to the implicit leading bit. See the [Wikipedia page on FP-32](#)

# Anatomy of a float



## Putting it together

- 32-bits [b<sub>31</sub>, ..., b<sub>0</sub>] represent a number as

$$(-1)^{b_{31}} \times 2^{(b_{30} \dots b_{23} - 127)_2} \times (1.b_{22} \dots b_0)_2$$

**Key facts**

- 8 bit biased exponent → [10<sup>-38</sup>, 10<sup>38</sup>]
- 24 bit fraction → 7 digit precision

# Quick knowledge test

- Can we represent  $\pi$  in FP32 without losing information?

$\pi = 3.1415926535897932384626433832795028841971693993751\dots$

- Can we represent  $1.1 \times 10^{26}$ ?
- How about  $1.1 \times 10^{49}$ ?
- How about  $1.1 \times 10^{-20}$ ?
- How about 0.0000000000000000000432?
- How about 900000000000000000000000?

# Quick knowledge test

- Can we represent  $\pi$  in FP32 without losing information?

$\pi = 3.1415926535897932384626433832795028841971693993751\dots$

- Can we represent  $1.1 \times 10^{26}$ ?
- How about  $1.1 \times 10^{49}$ ?
- How about  $1.1 \times 10^{-20}$ ?
- How about 0.0000000000000000000432?
- How about 900000000000000000000000?

# Recap: RAM requirements for LLMs

# Recap: RAM requirements for LLMs

- I have a **7 billion parameter** model. How much RAM do I need to host it with **FP32**?

# Recap: RAM requirements for LLMs

- I have a **7 billion parameter** model. How much RAM do I need to host it with **FP32**?
- RAM Usage = **7 billion params** x **4 bytes per parameter** = 28 billion bytes  $\approx$  **28 GB**

# Recap: RAM requirements for LLMs

- I have a **7 billion parameter** model. How much RAM do I need to host it with **FP32**?
- RAM Usage = **7 billion params** x **4 bytes per parameter** = 28 billion bytes  $\approx$  **28 GB**
- I have a 405 billion parameter model. How much RAM do I need to host it with **FP32**?

# Recap: RAM requirements for LLMs

- I have a **7 billion parameter** model. How much RAM do I need to host it with **FP32**?
- RAM Usage = **7 billion params** x **4 bytes per parameter** = **28 billion bytes**  $\approx$  **28 GB**
- I have a **405 billion parameter** model. How much RAM do I need to host it with **FP32**?
- RAM Usage = **405 billion params** x **4 bytes per parameter**  $\approx$  **1.6 TB**

# Recap: RAM requirements for LLMs

- I have a **7 billion parameter** model. How much RAM do I need to host it with **FP32**?
- RAM Usage = **7 billion params** x **4 bytes per parameter** = 28 billion bytes  $\approx$  **28 GB**
- I have a 405 billion parameter model. How much RAM do I need to host it with **FP32**?
- RAM Usage = **405 billion params** x **4 bytes per parameter**  $\approx$  **1.6 TB**
- **Idea:** Represent numbers with lower number of bits

# FP32 → FP16



**Sign**  
(1 bit)

**Exponent**  
(5 bits)

**Fraction**  
(10 bits)

## Exponent (E)

- **Biased representation:** 15 is the 0 point
- The power represented is  $2^{E-15}$
- $E = 1 \rightarrow 2^{-14}$  |  $E = 15 \rightarrow 2^0$  |  $E = 30 \rightarrow 2^{15}$
- Range is  $[2^{-14}, 2^{15}] \approx [10^{-4}, 10^4]$

# FP32 → FP16



**Sign**  
(1 bit)

**Exponent**  
(5 bits)

**Fraction**  
(10 bits)

## Exponent (E)

- **Biased representation:** 15 is the 0 point
- The power represented is  $2^{E-15}$
- $E = 1 \rightarrow 2^{-14}$  |  $E = 15 \rightarrow 2^0$  |  $E = 30 \rightarrow 2^{15}$
- Range is  $[2^{-14}, 2^{15}] \approx [10^{-4}, 10^4]$

Range is number smaller than FP32  $[10^{-38}, 10^{38}]$

# Peculiarities of deep models

- Parameters can be quite small or large (often on the small side)
- For instance,  $10^{-6}$
- The precision of the fraction itself doesn't matter that much
- **Idea:** Use 16 bits, but use more bits for the **exponent**
- That is where **bfloat16** comes in
- Developed at Google Brain

# bfloat16



## Exponent (E)

- **Biased representation:** 127 is the 0 point
- The power represented is  $2^{E-127}$
- $E = 1 \rightarrow 2^{-126}$  |  $E = 127 \rightarrow 2^0$  |  $E = 254 \rightarrow 2^{127}$
- Range is  $[2^{-126}, 2^{127}] \approx [10^{-38}, 10^{38}]$

# bfloat16



## Exponent (E)

- **Biased representation:** 127 is the 0 point
- The power represented is  $2^{E-127}$
- $E = 1 \rightarrow 2^{-126}$  |  $E = 127 \rightarrow 2^0$  |  $E = 254 \rightarrow 2^{127}$
- Range is  $[2^{-126}, 2^{127}] \approx [10^{-38}, 10^{38}]$

Same range as FP32  $[10^{-38}, 10^{38}]$

# Exercise

# Representing data using int16



- Can represent  $2^{16} - 1$  values
- Range from [-32K, 32K]
- Two's complement notation where 0 is not counted twice

# Representing data using int16



- Can represent  $2^{16} - 1$  values
- Range from [-32K, 32K]
- Two's complement notation where 0 is not counted twice

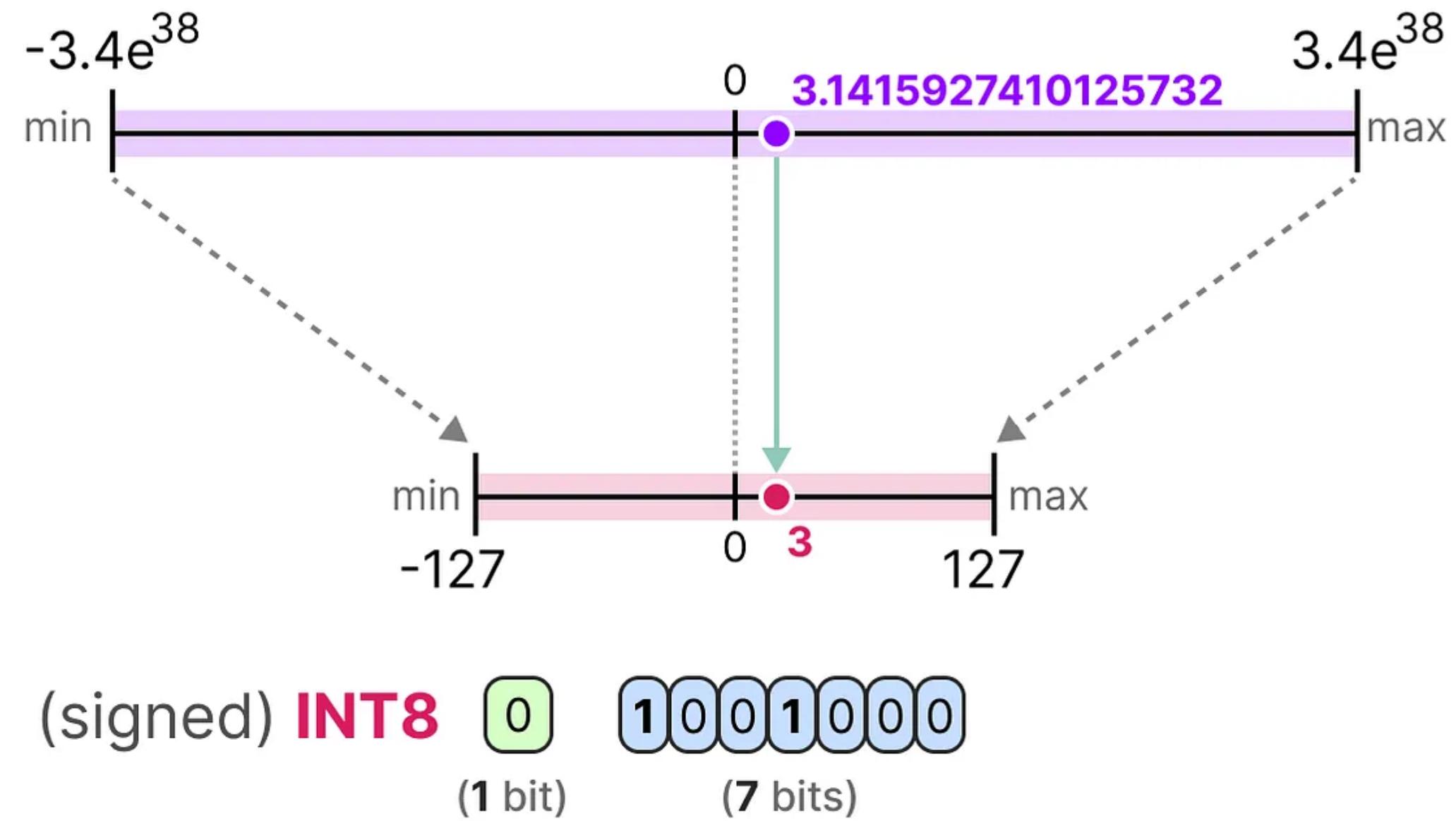
**Knowledge check: What is the range of int32?**

# Representing data using int8



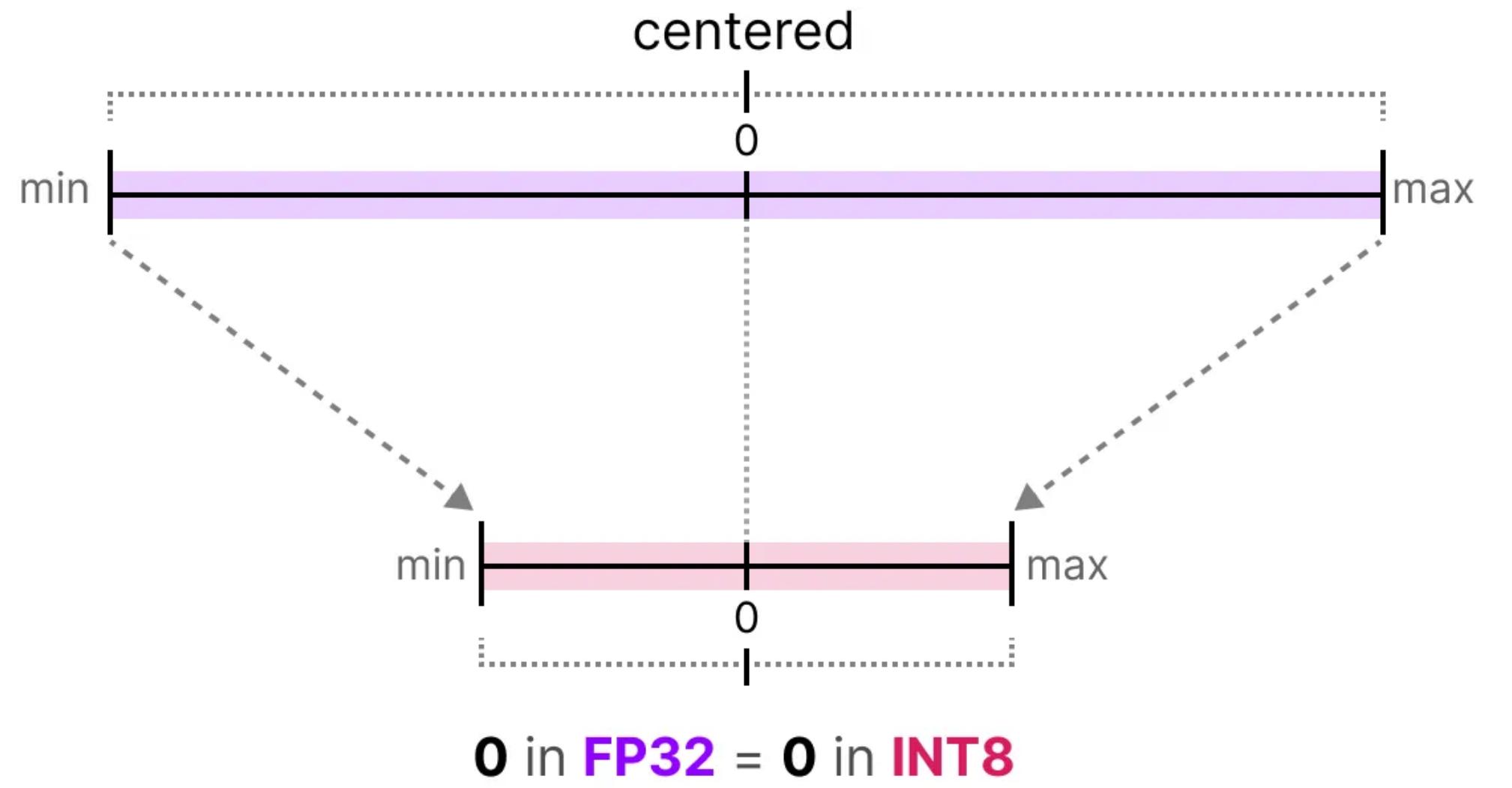
- Can represent  $2^8 - 1 = 255$  values
- Range from [-128, 127]

# Quantizing weights to int8



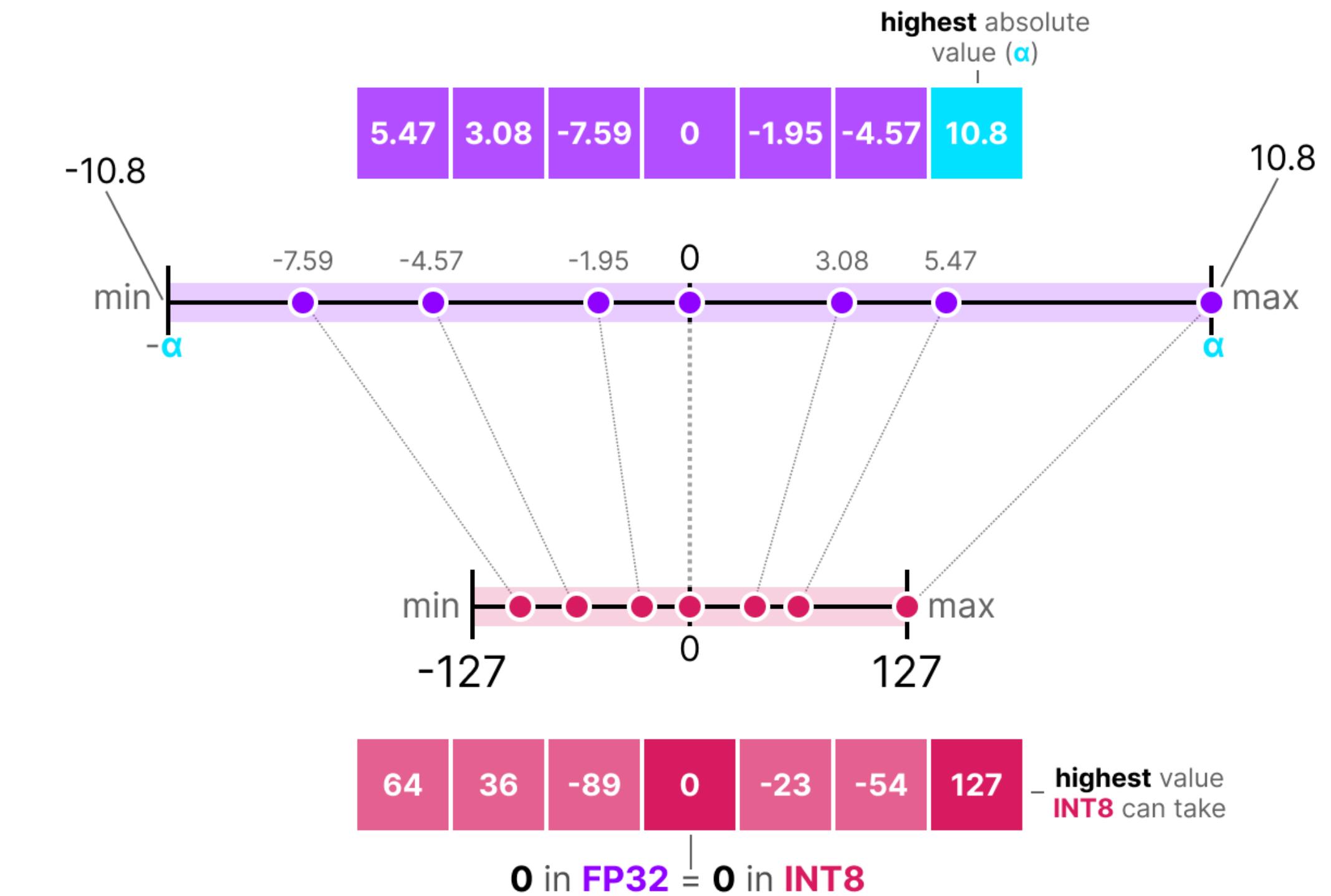
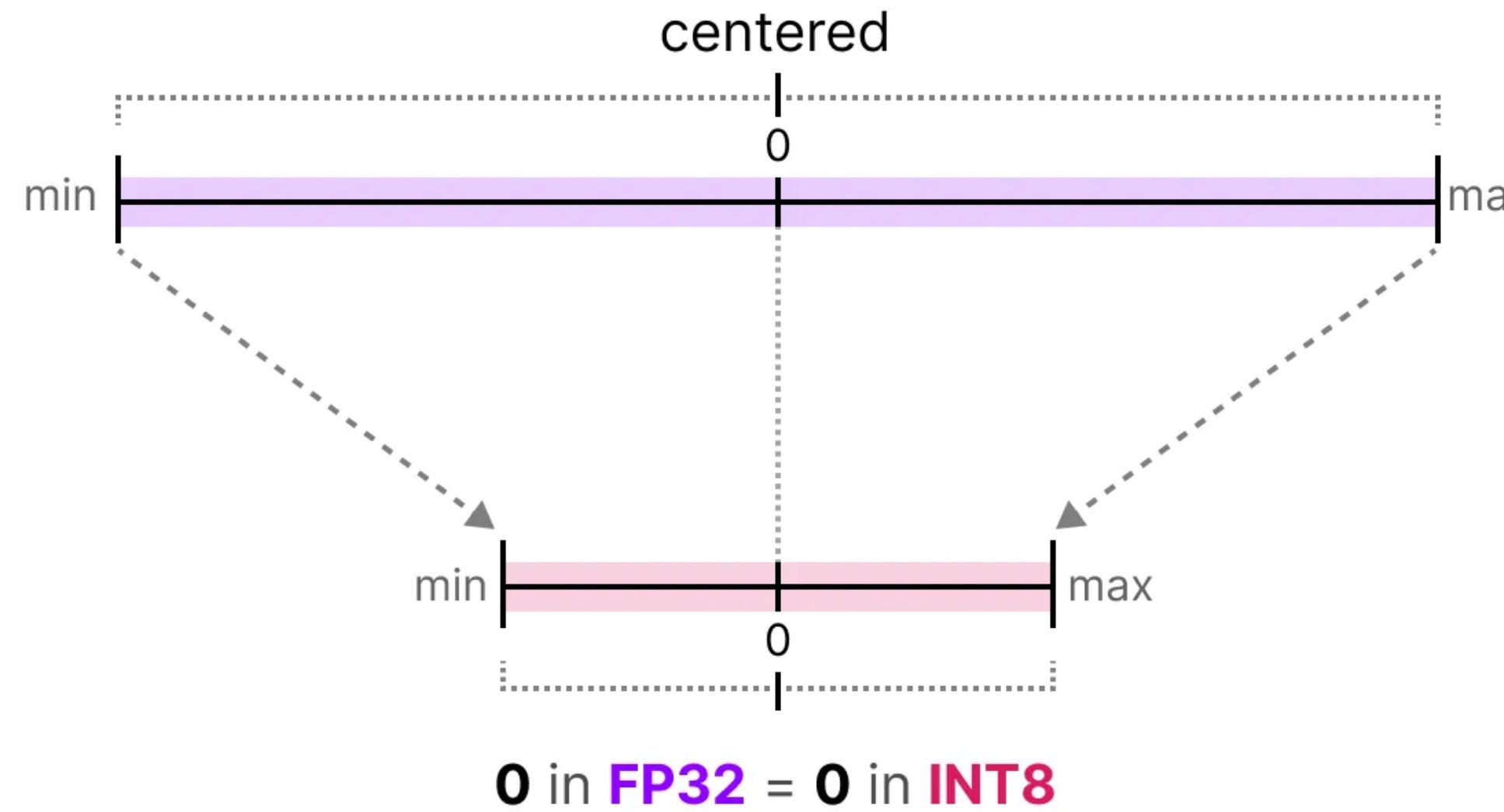
- Whole range between  $-10^{38}$  and  $10^{38}$  gets mapped to 254 distinct values
  - Using 8 bit ints has advantages
    - Half as much space as bfloat16
    - Can move the data quicker between RAM → GPU VRAM → computation cores
    - Hardware optimizations that can perform int8 operations much faster

# Symmetric quantization



- 0 points of base and quantized weight match
- Min. and max. values are negative of each other

# Symmetric quantization



- 0 points of base and quantized weight match
- Min. and max. values are negative of each other

# Performing symmetric quantization

## Quantization

$$S = \frac{2^{b-1} - 1}{\alpha} \quad (\text{scale factor})$$

---

$$x_{\text{quantized}} = \text{round}(S \cdot x) \quad (\text{quantization})$$

## De-quantization

$$x_{\text{dequantized}} = \frac{\text{████████}}{S} \quad (\text{dequantize})$$

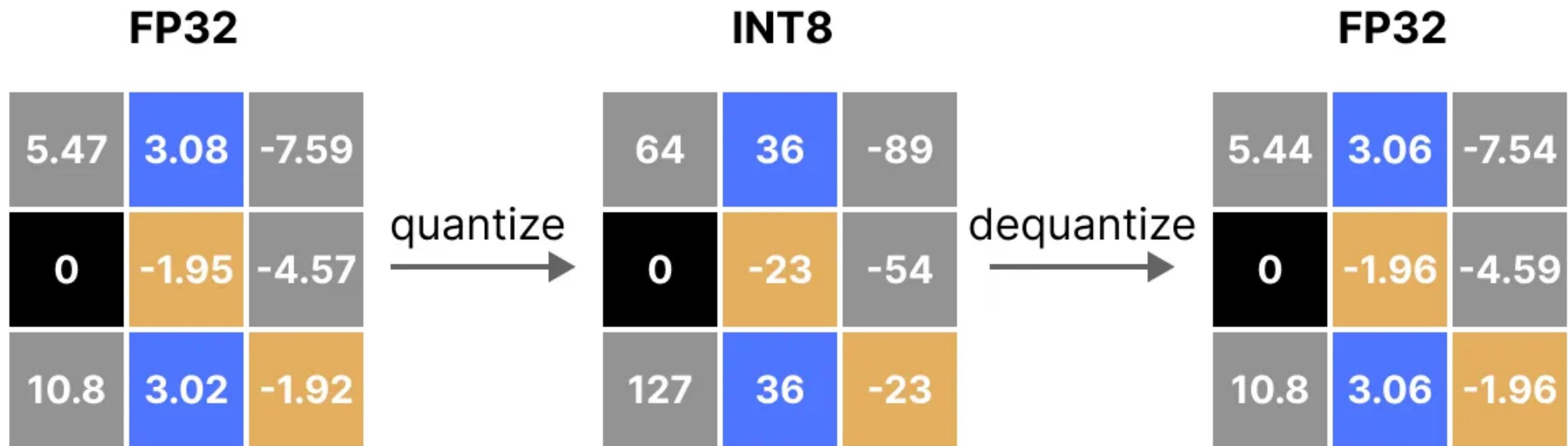
$$S = \frac{127}{10.8} = 11.76 \quad (\text{scale factor})$$

---

$$x_{\text{quantized}} = \text{round}(11.76 \cdot \text{████}) \quad (\text{quantization})$$

- $\alpha$  is the value with the largest magnitude

# Quantization error

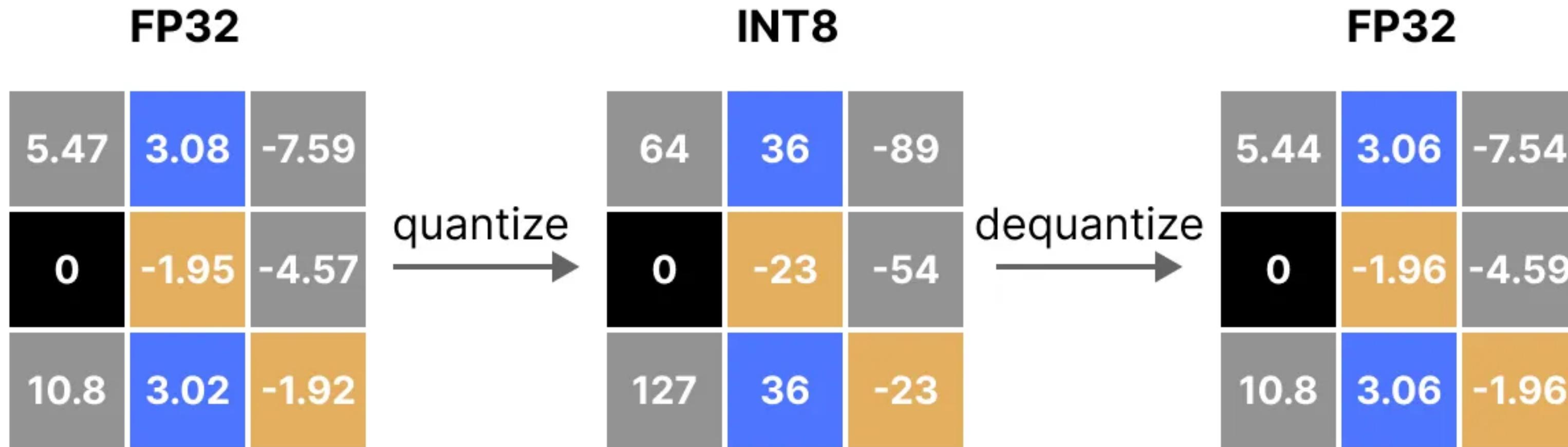


- $(3.08, 3.02) \rightarrow (36, 36) \rightarrow (3.06, 3.06)$
- Distinct values get mapped at the same quantized value
- All values have some error
- In practice, the error doesn't make a big difference in performance

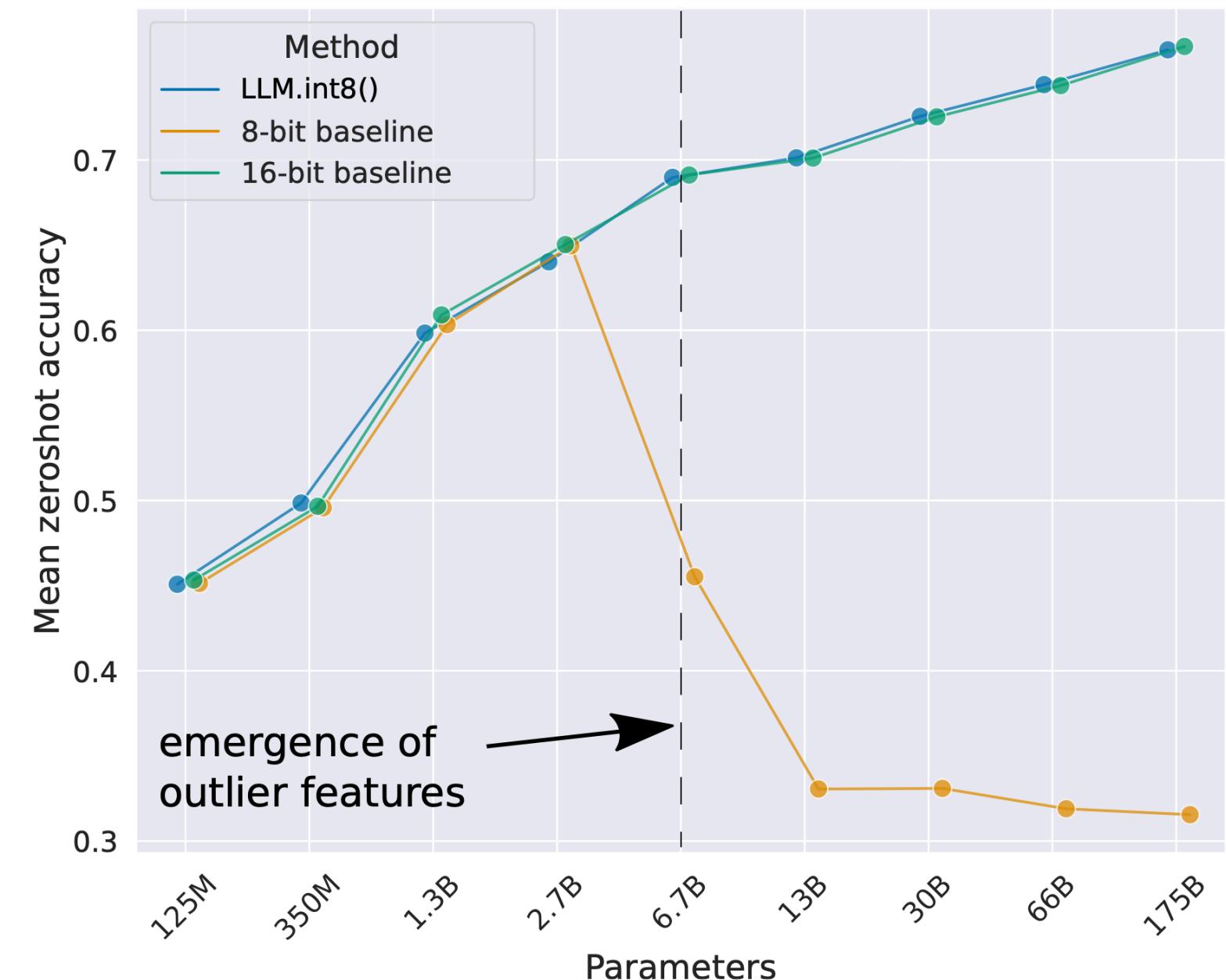
[Dettmers et al]

[Grootendorst]

# Quantization error



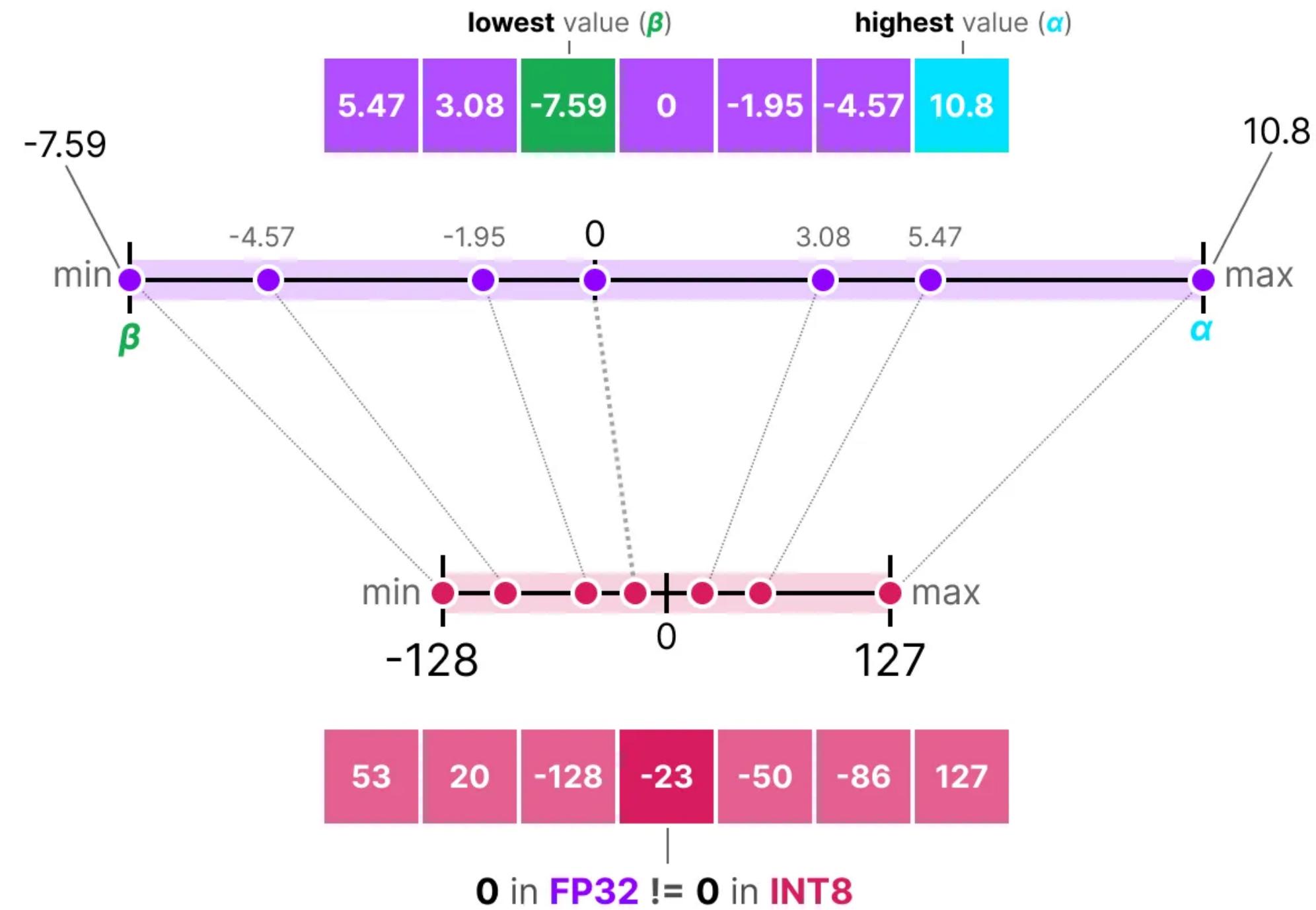
- $(3.08, 3.02) \rightarrow (36, 36) \rightarrow (3.06, 3.06)$
- Distinct values get mapped at the same quantized value
- All values have some error
- In practice, the error doesn't make a big difference in performance



[Dettmers et al]

[Grootendorst]

# Asymmetric quantization



- Offers more precision than symmetric quantization
- 0s do not match

# Performing asymmetric quantization

## Quantization

$$S = \frac{128 - -127}{\alpha - \beta}$$

(scale factor)

$$Z = \text{round}(-S \cdot \beta) - 2^{b-1}$$

(zeropoint)

$$x_{\text{quantized}} = \text{round}(S \cdot x + Z)$$

(quantization)

$$S = \frac{255}{10.8 - -7.59} = 13.86$$

(scale factor)

$$Z = \text{round}(-13.86 \cdot -7.59) - 128 = -23$$

(zeropoint)

$$x_{\text{quantized}} = \text{round}(13.86 \cdot \text{████████} + -23)$$

(quantization)

## De-quantization

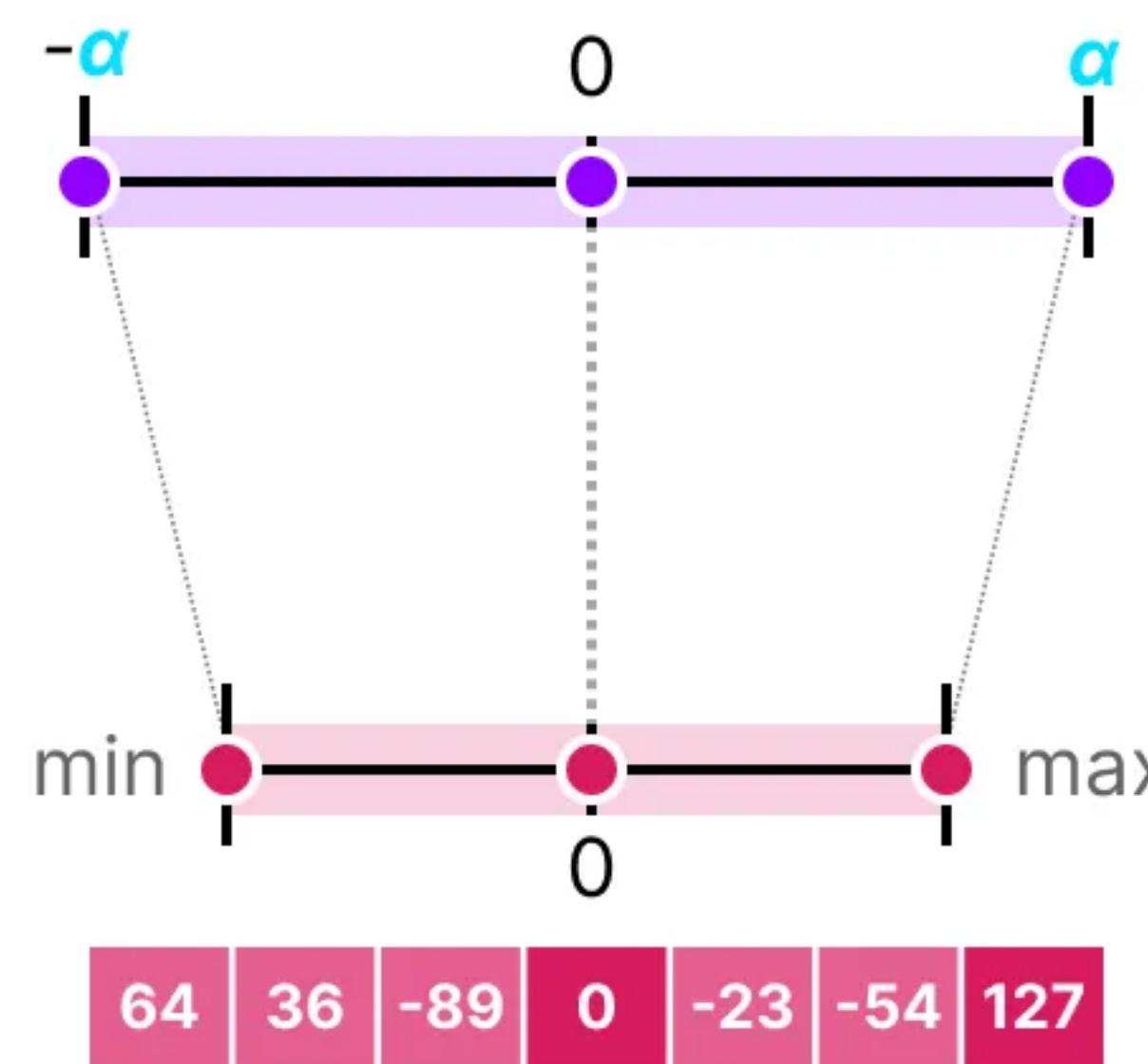
$$x_{\text{dequantized}} = \frac{\text{████████} - Z}{S}$$

(dequantize)

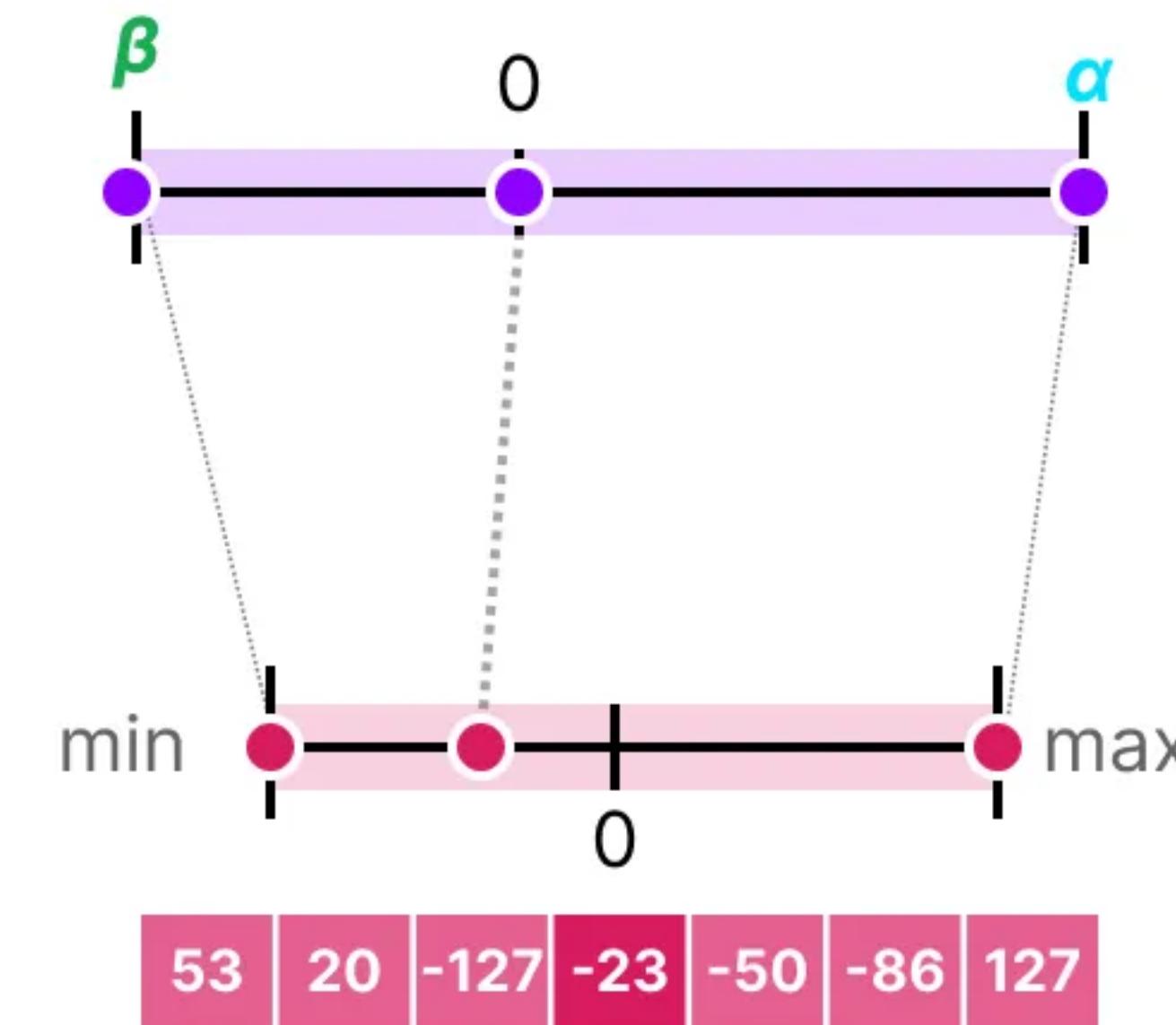
# Symmetric vs. asymmetric quantization

| lowest value ( $\beta$ )                | highest value ( $\alpha$ ) |
|---|----------------------------|
| 5.47   3.08   -7.59   0   -1.95   -4.57 | 10.8                       |

**Symmetric**  
[-10.8, 10.8]

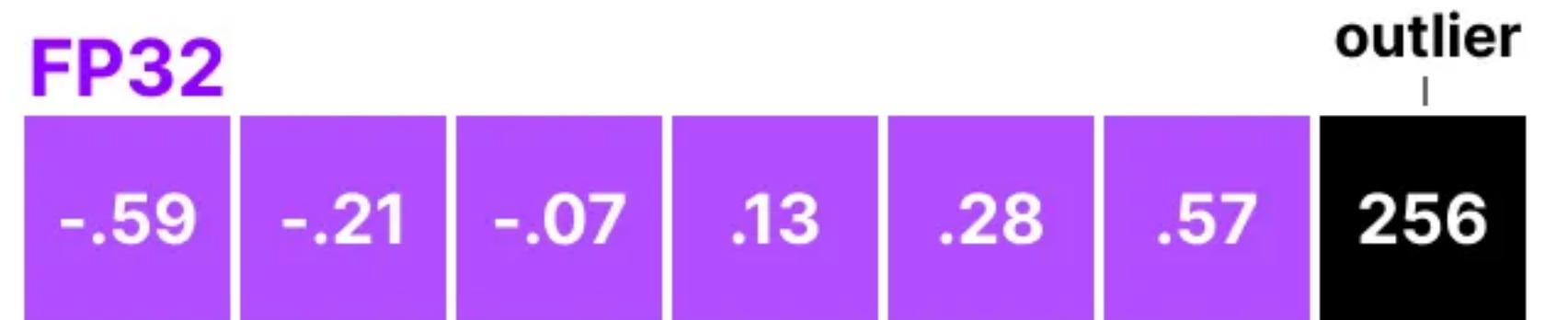


**Asymmetric**  
[-7.59, 10.8]



# Outliers

- Outliers can take precious quantization space
- Large impact on model performance
- Start emerging at model scaled beyond 1B parameters
- Prevalence and intensity gets worse with larger sizes

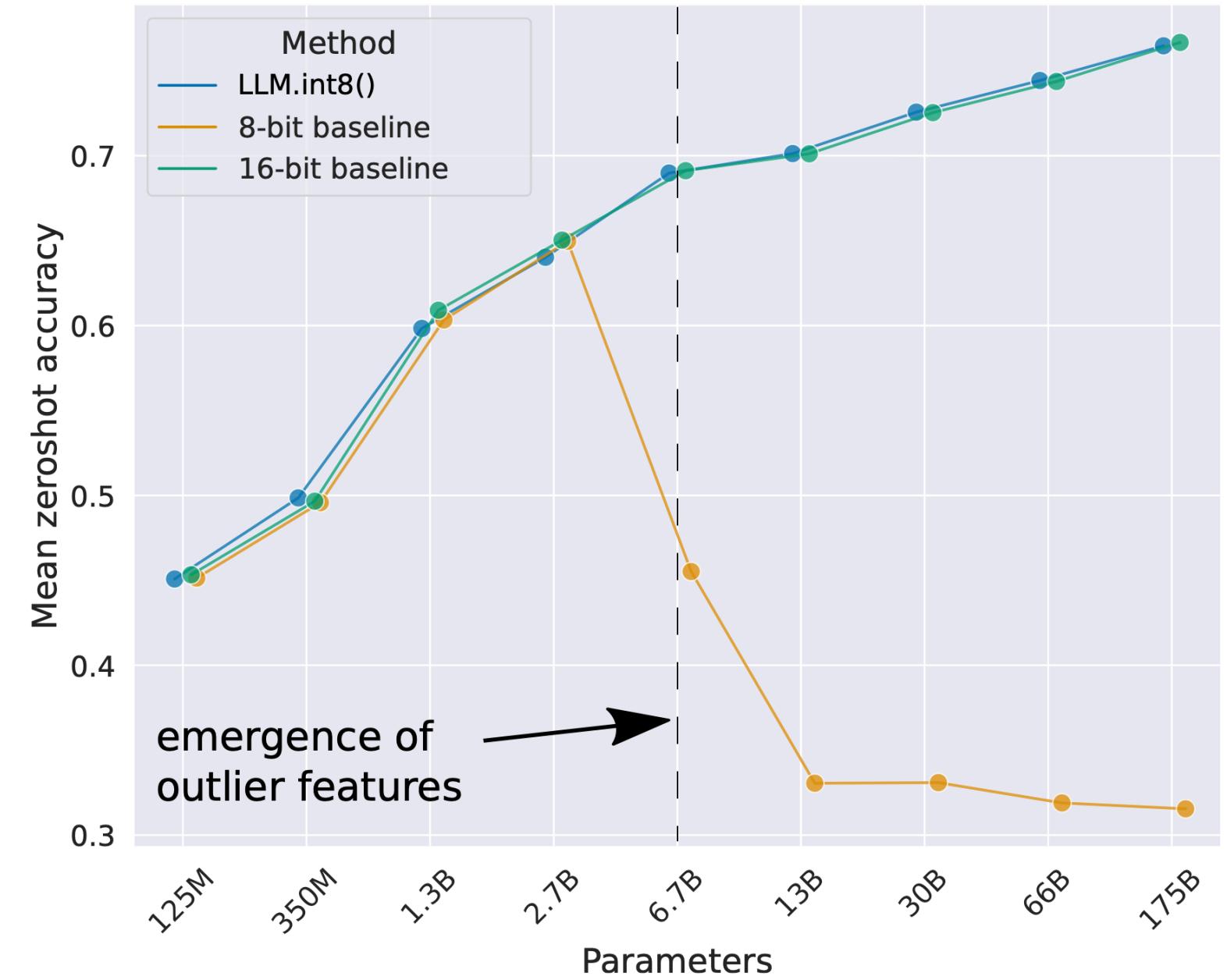
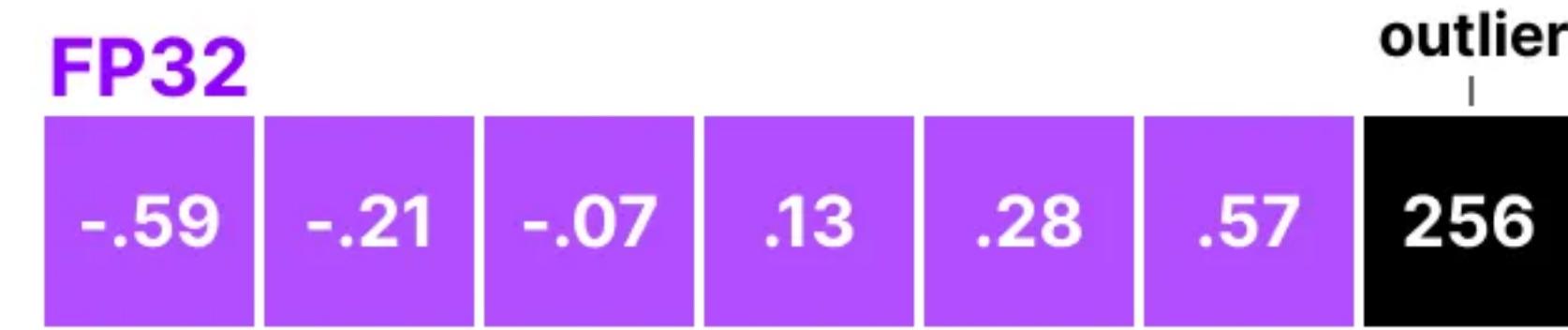


[Dettmers et al]

[Grootendorst]

# Outliers

- Outliers can take precious quantization space
- Large impact on model performance
- Start emerging at model scaled beyond 1B parameters
- Prevalence and intensity gets worse with larger sizes

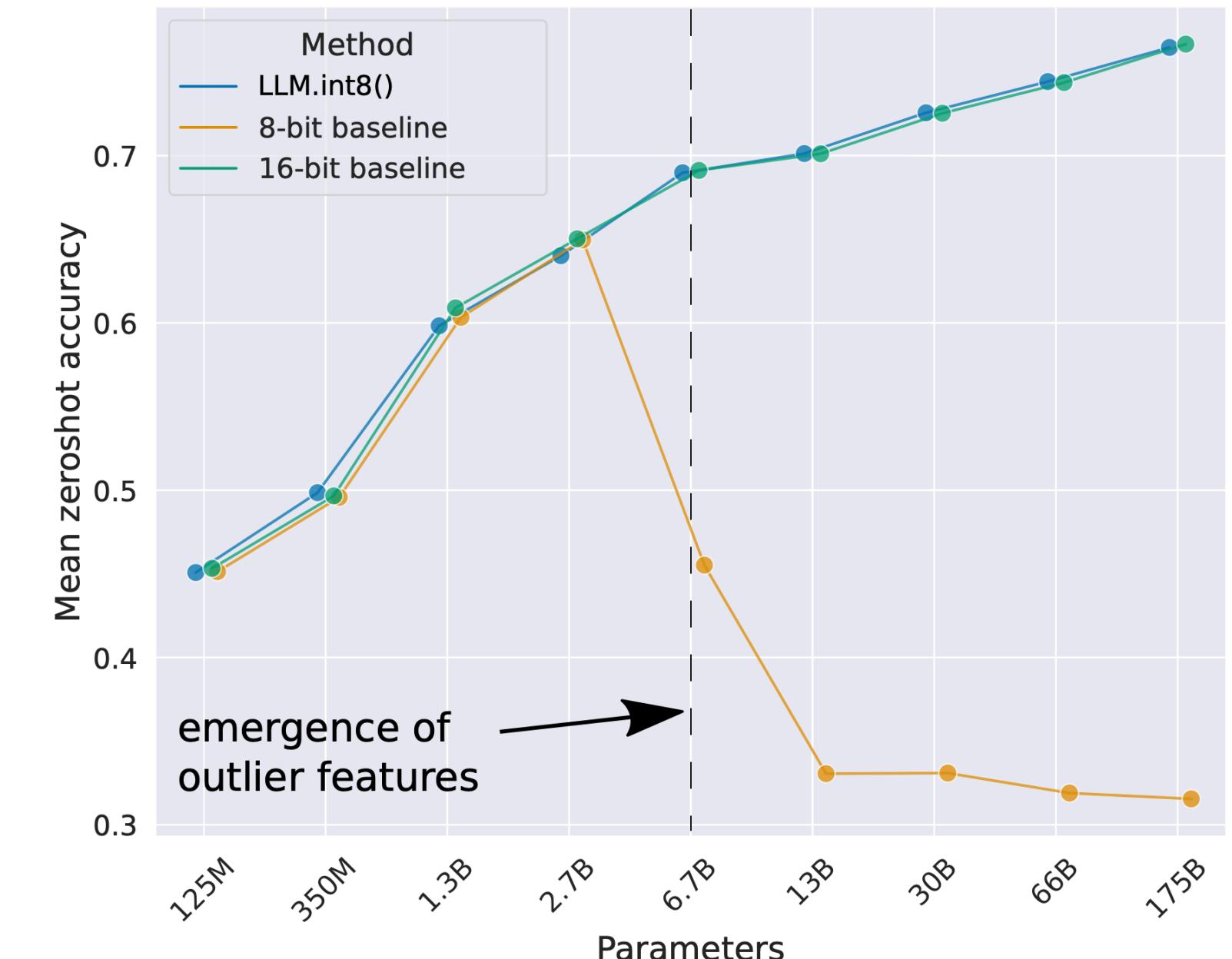
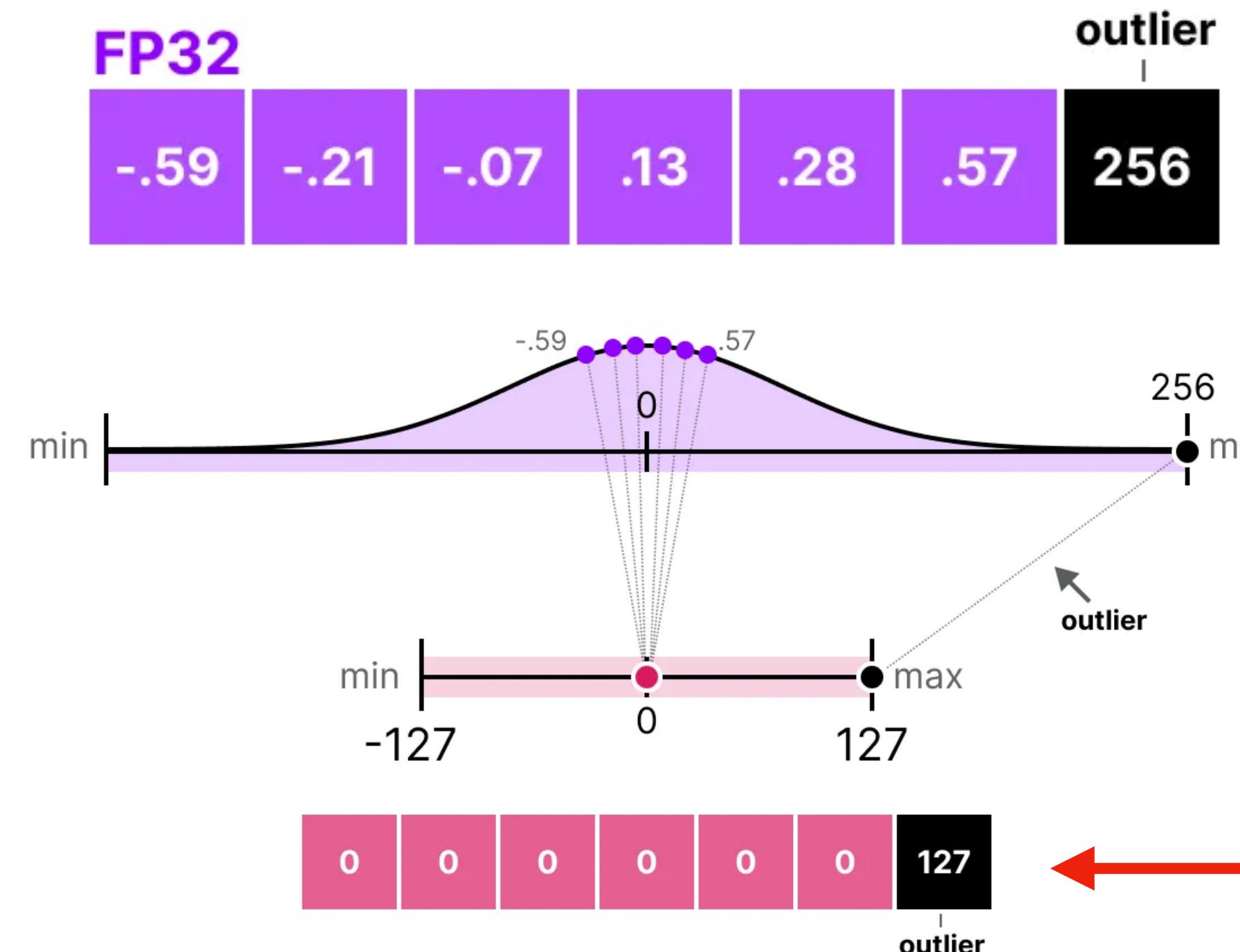


[Dettmers et al]

[Grootendorst]

# Outliers

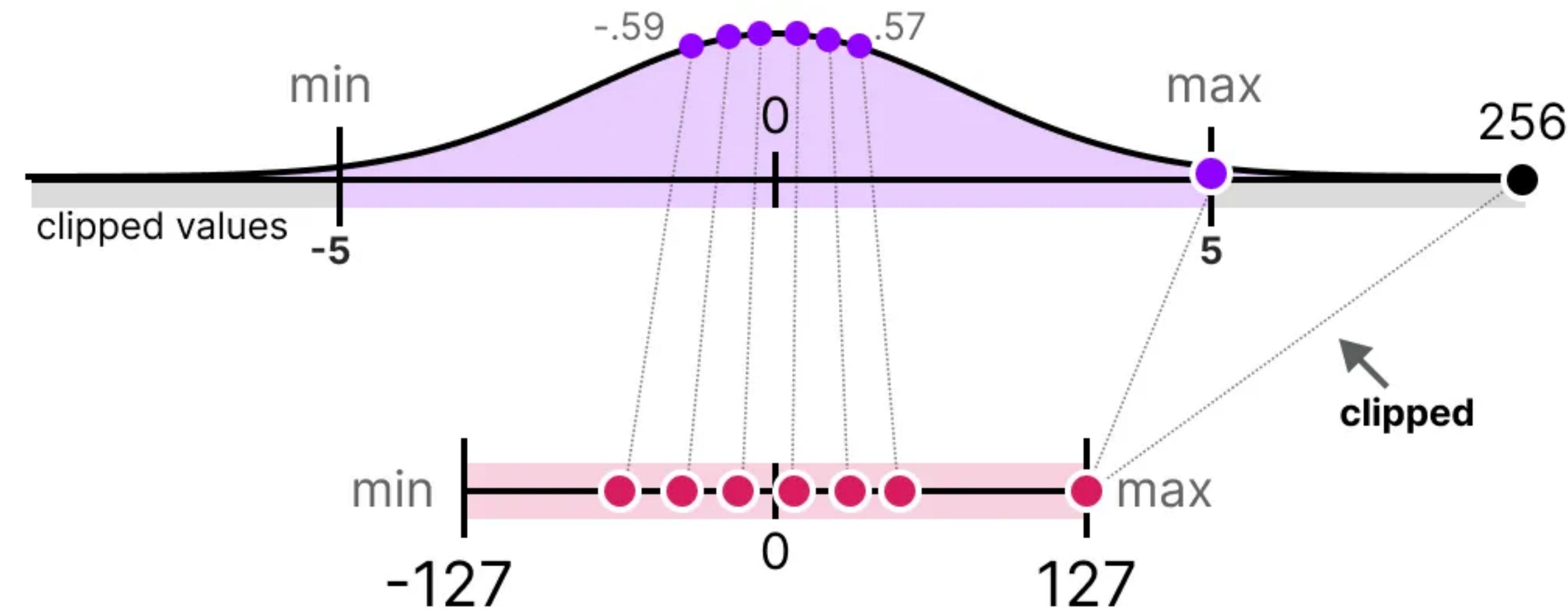
- Outliers can take precious quantization space
- Large impact on model performance
- Start emerging at model scaled beyond 1B parameters
- Prevalence and intensity gets worse with larger sizes



[Dettmers et al]

[Grootendorst]

# Tackling outliers with clipping



- Clip the weights to a pre-determined range, in this case  $[-5, 5]$
- Any value outside the range gets mapped to the endpoints, e.g., 256 becomes 5
- Can learn using data, e.g., clip the top 5 percentiles

# A lot more to quantization

- Quantization to 4 bits
- Quantizing different weights differently
  - Some weights are more important than others
- Quantizing weights but not biases
  - Weight matrix  $D \times D$ , bias vector  $1 \times D$
- Will see more in the next lecture

# Using Google Colab

# Exercise

# References

- [Attention is all you need](#)
- [A Visual Guide to Quantization](#)
- Wikipedia page on [Single-precision floating-point format](#)
- [What Every User Should Know About Mixed Precision Training in PyTorch](#)
- [Quantization Concepts](#)
- [LLM.int8\(\): 8-bit Matrix Multiplication for Transformers at Scale](#)