https://hao-ai-lab.github.io/cse234-w25/

CSE 234: Data Systems for Machine Learning
Winter 2025

LLMSys

Optimizations and Parallelization

MLSys Basics

Which of the following is not one of the main sources of memory consumption?

A. Intermediate activation values
B. Model weights
C. Optimizer states
D. Training code

Which of the following statements is false about gradient checkpointing?

A. Applying gradient checkpointing during model training could save GPU memory
B. Gradient checkpointing applies to both model weights and activations
C. The location of gradient checkpointing affects how much re-computation and memory are needed
D. It is possible to discard some of the activation from memory since they are only needed during backward pass

Given this table of model configurations, what is the activation size of GPT-3 2.7B (using fp16 for all activations and checkpointing at the transformer layer boundary)?

A. 7031.25GB

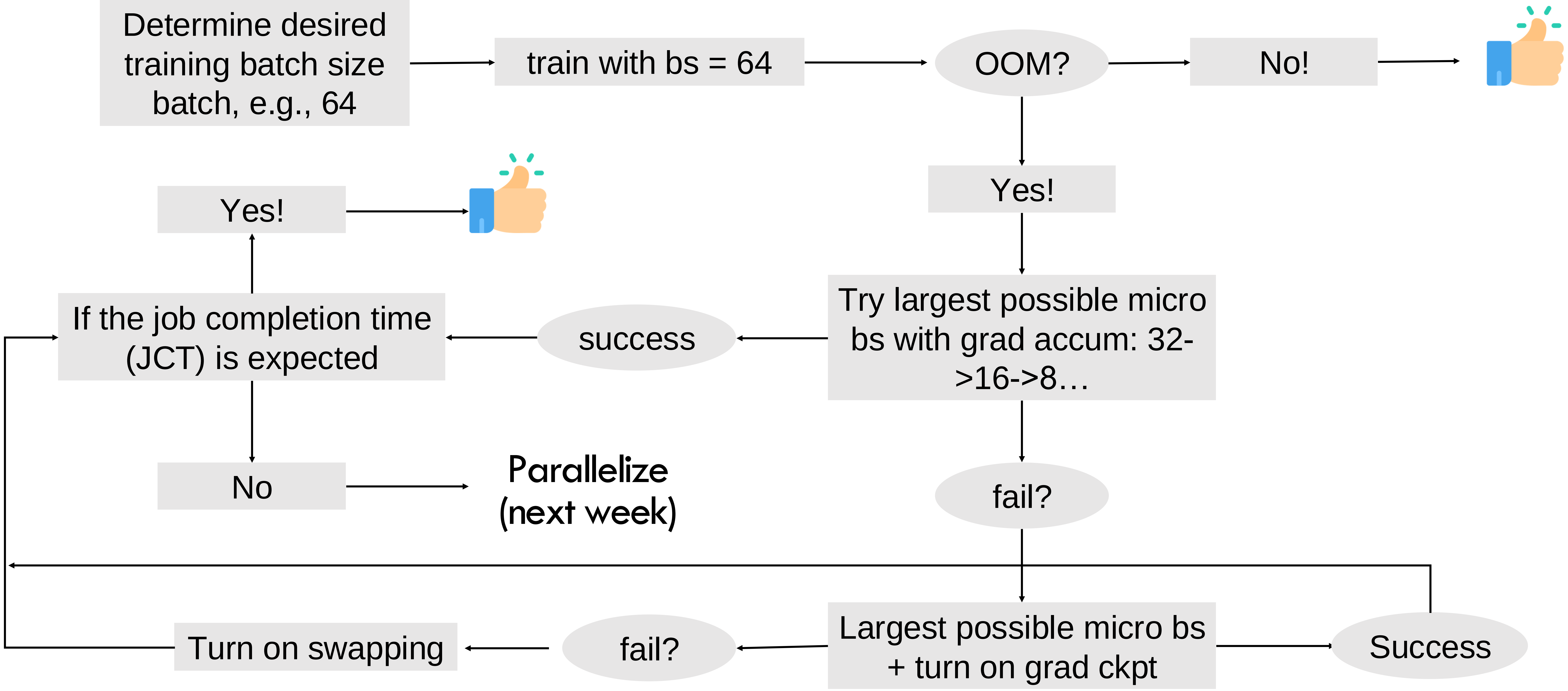
B. 201.34GB

C. 152.59GB

D. 305.18GB

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

How to Choose/Tune Memory Optimization



Quantization

- **Digital representation of data**
- Basics of quantization
- Quantization in ML
  - Post-training quantization
- Mixed precision

Floating-point Representation

$2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

Sign 8 bit Exponent **23 bit Fraction**

$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$ ← Exponent Bias = 127 = $2^{8-1}-1$

(significant / mantissa)

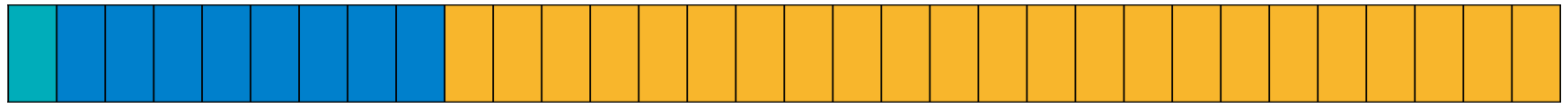
$$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$$

0	0	1	1	1	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$\underbrace{01111101}_{125}$ $\underbrace{00010000000000000000000}_{0.0625}$

Q: How to represent 0?

Floating-point Number: normal vs. subnormal



Sign **8 bit Exponent** **23 bit Fraction**

$$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$

(Normal Numbers, Exponent≠0)

Should have been $(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\mathbf{0}-127}$

But we force to be $(-1)^{sign} \times \textbf{Fraction} \times 2^{\mathbf{1}-127}$

(Subnormal Numbers, Exponent=0)

0 | 01111101 | 00010000000000000000000

$$\underbrace{01111101}_{125} \quad \underbrace{00010000000000000000000}_{0.0625}$$

$$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$$

0 | 00000000 | 00000000000000000000000

$$\underbrace{00000000}_{0} \quad \underbrace{00000000000000000000000}_{0}$$

1 | 00000000 | 00000000000000000000000

$$\underbrace{00000000}_{0} \quad \underbrace{00000000000000000000000}_{0}$$

$$0 = 0 \times 2^{-126}$$

Q: What is them minimum positive value?

What is the minimum positive value?

Sign 8 bit Exponent 23 bit Fraction

$$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$$

(Normal Numbers, Exponent≠0)

0 | 00000001 | 00000000000000000000000

$\underbrace{00000001}_{1}$ $\underbrace{00000000000000000000000}_{0}$

$$2^{-126} = (1 + \underline{0}) \times 2^{\underline{1}-127}$$

$$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$$

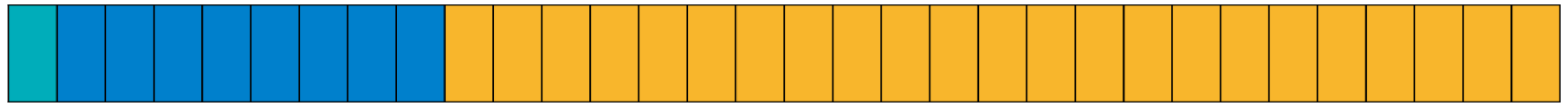
(Subnormal Numbers, Exponent=0)

0 | 00000000 | 00000000000000000000001

$\underbrace{00000000}_{0}$ $\underbrace{00000000000000000000001}_{2^{-23}}$

$$2^{-149} = 2^{-23} \times 2^{-126}$$

Some Special Values



Sign **8 bit Exponent** **23 bit Fraction**

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

+∞ (positive infinity)

1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

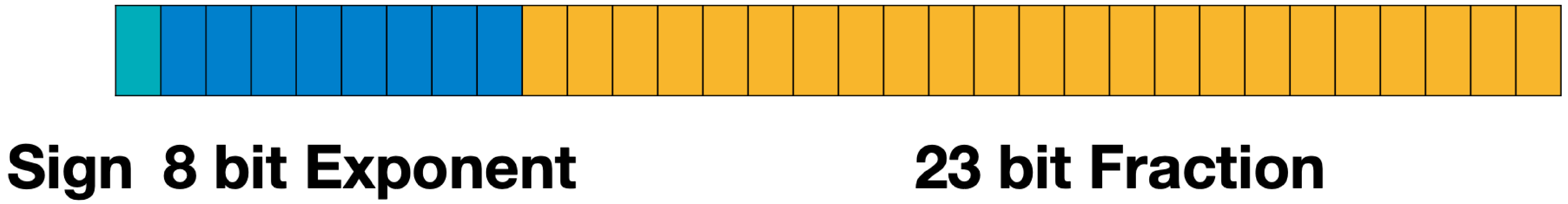
-∞ (negative infinity)

- 1 1 1 1 1 1 1 1 -

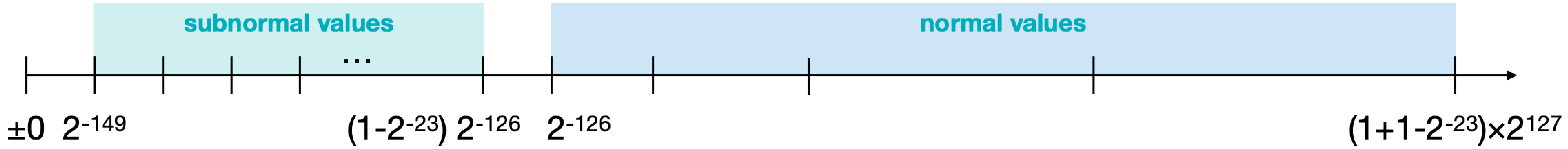
NaN (Not a Number)

much waste. Revisit in fp8.

Summary of fp32

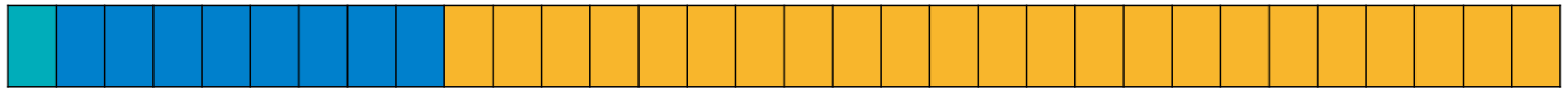
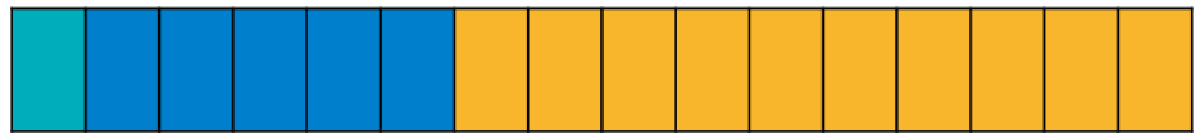
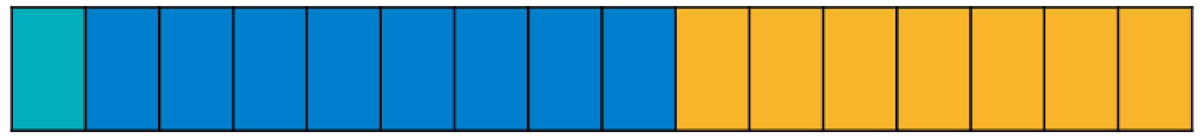


Exponent	Fraction=0	Fraction≠0	Equation
$00_H = 0$	±0	subnormal	$(-1)^{sign} \times \textbf{Fraction} \times 2^{\mathbf{1}-127}$
$01_H \ldots FE_H = 1 \ldots 254$	normal		$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$
$FF_H = 255$	±INF	NaN	

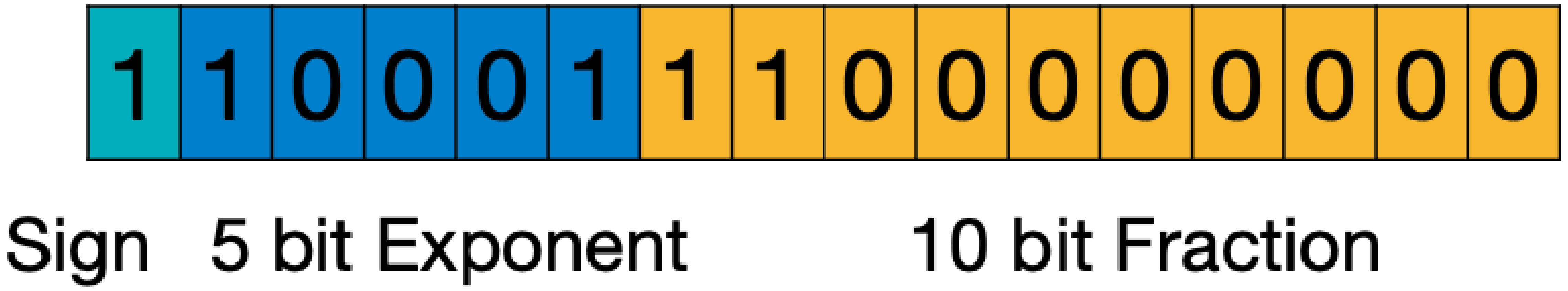


FP32 vs. FP16 vs. BF16

- Exponent width -> Range; Fraction width -> precision

	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE 754 Single Precision 32-bit Float (IEEE FP32) 	8	23	32
IEEE 754 Half Precision 16-bit Float (IEEE FP16) 	5	10	16
Google Brain Float (BF16) 	8	7	16

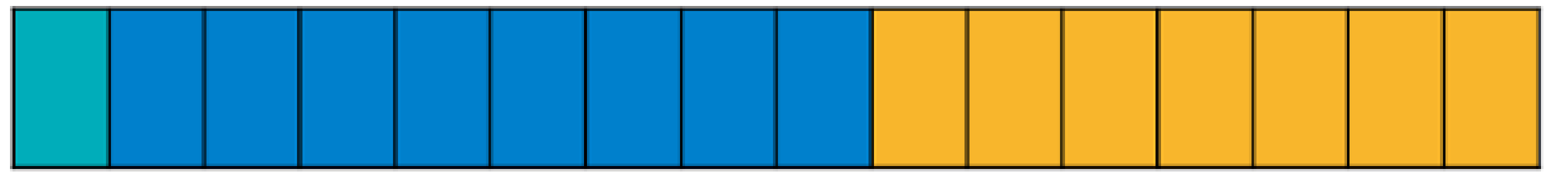
Exercise



- Sign: -
- Exponent
  - Bias:$2^4 - 1 = 15_{10}$
  - $10001_2 - 15_{10} = 17_{10} - 15_{10} = 2_{10}$
- Fraction
  - $1100000000_2 = 0.75_{10}$
- Answer: $-(1 + 0.75) \times 2^2 = -7.10_{10}$

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

Google Brain Float (BF16)



Exercise

What is Decimal 2.5 in BF16?

- $2.5 = 1.25 \times 2^1$
- Sign: +
- Exponent: bias is $2^7 - 1 = 127$
  - $x - 127 = 1; x = 128_{10} = 10000000_2$
- Fraction: 7-bit fraction
  - $0.25 = 0100000_2$

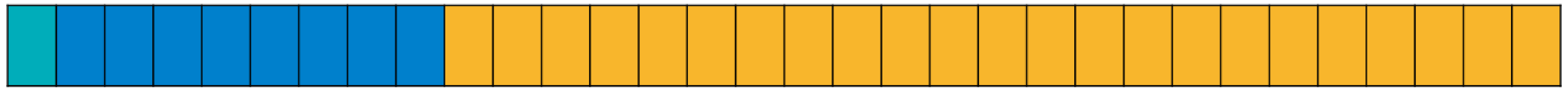
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Sign 8 bit Exponent 7 bit Fraction

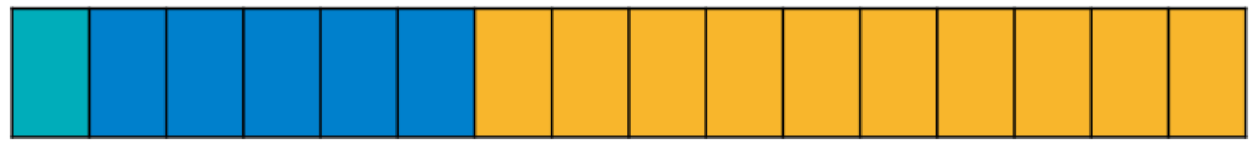
Latest FP8

- Exponent width -> Range; Fraction width -> precision

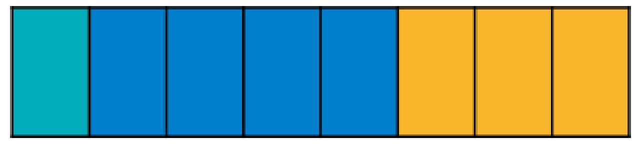
IEEE 754 Single Precision 32-bit Float (IEEE FP32)



IEEE 754 Half Precision 16-bit Float (IEEE FP16)

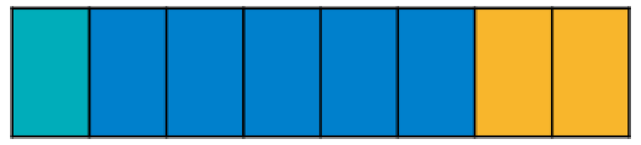


Nvidia FP8 (E4M3)



* FP8 E4M3 does not have INF, and $S.1111.111_2$ is used for NaN.
* Largest FP8 E4M3 normal value is $S.1111.110_2 = 448$.

Nvidia FP8 (E5M2) for gradient in the backward



* FP8 E5M2 have INF ($S.11111.00_2$) and NaN ($S.11111.XX_2$).
* Largest FP8 E5M2 normal value is $S.11110.11_2 = 57344$.

	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE 754 Single Precision 32-bit Float (IEEE FP32)	8	23	32
IEEE 754 Half Precision 16-bit Float (IEEE FP16)	5	10	16
Nvidia FP8 (E4M3)	4	3	8
Nvidia FP8 (E5M2) for gradient in the backward	5	2	8

INT4 and FP8

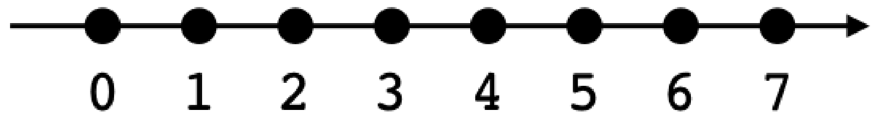
INT4

S			
0	0	0	1
0	1	1	1

0001 $=1$

0111 $=7$

-1,-2,-3,-4,-5,-6,-7,-8
0, 1, 2, 3, 4, 5, 6, 7



-1,-2,-3,-4,-5,-6,-7,-8
0, 1, 2, 3, 4, 5, 6, 7

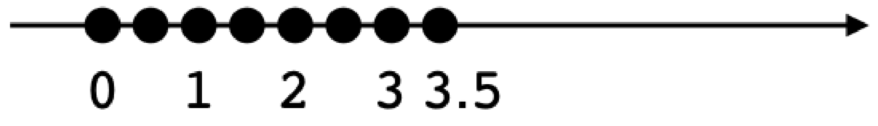
FP4 (E1M2)

S	E	M	M
0	0	0	1
0	1	1	1

0001 $=0.25\times2^{1-0}=0.5$

0111 $=(1+0.75)\times2^{1-0}=3.5$

-0,-0.5,-1,-1.5,-2,-2.5,-3,-3.5
0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5



-0,-1,-2,-3,-4,-5,-6,-7
0, 1, 2, 3, 4, 5, 6, 7 ×0.5

FP4 (E2M1)

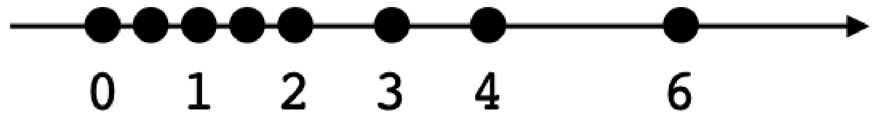
S	E	E	M
0	0	0	1
0	1	1	1

0001 $=0.5\times2^{1-1}=0.5$

0111 $=(1+0.5)\times2^{3-1}=1$

no inf, no NaN

-0,-0.5,-1,-1.5,-2,-3,-4,-6
0, 0.5, 1, 1.5, 2, 3, 4, 6



-0,-1,-2,-3,-4,-6,-8,-12
0, 1, 2, 3, 4, 6, 8, 12 ×0.5

FP4 (E3M0)

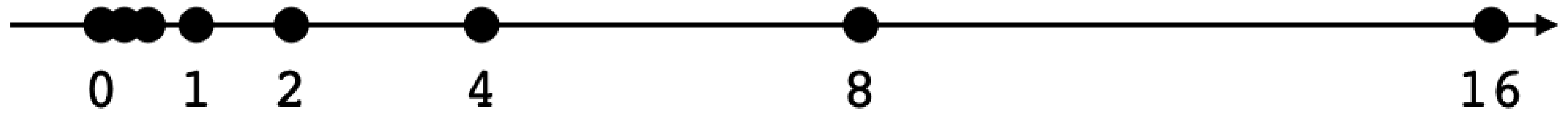
S	E	E	E
0	0	0	1
0	1	1	1

0001 $=(1+0)\times2^{1-3}=0.25$

0111 $=(1+0)\times2^{7-3}=16$

no inf, no NaN

-0,-0.25,-0.5,-1,-2,-4,-8,-16
0, 0.25, 0.5, 1, 2, 4, 8, 16



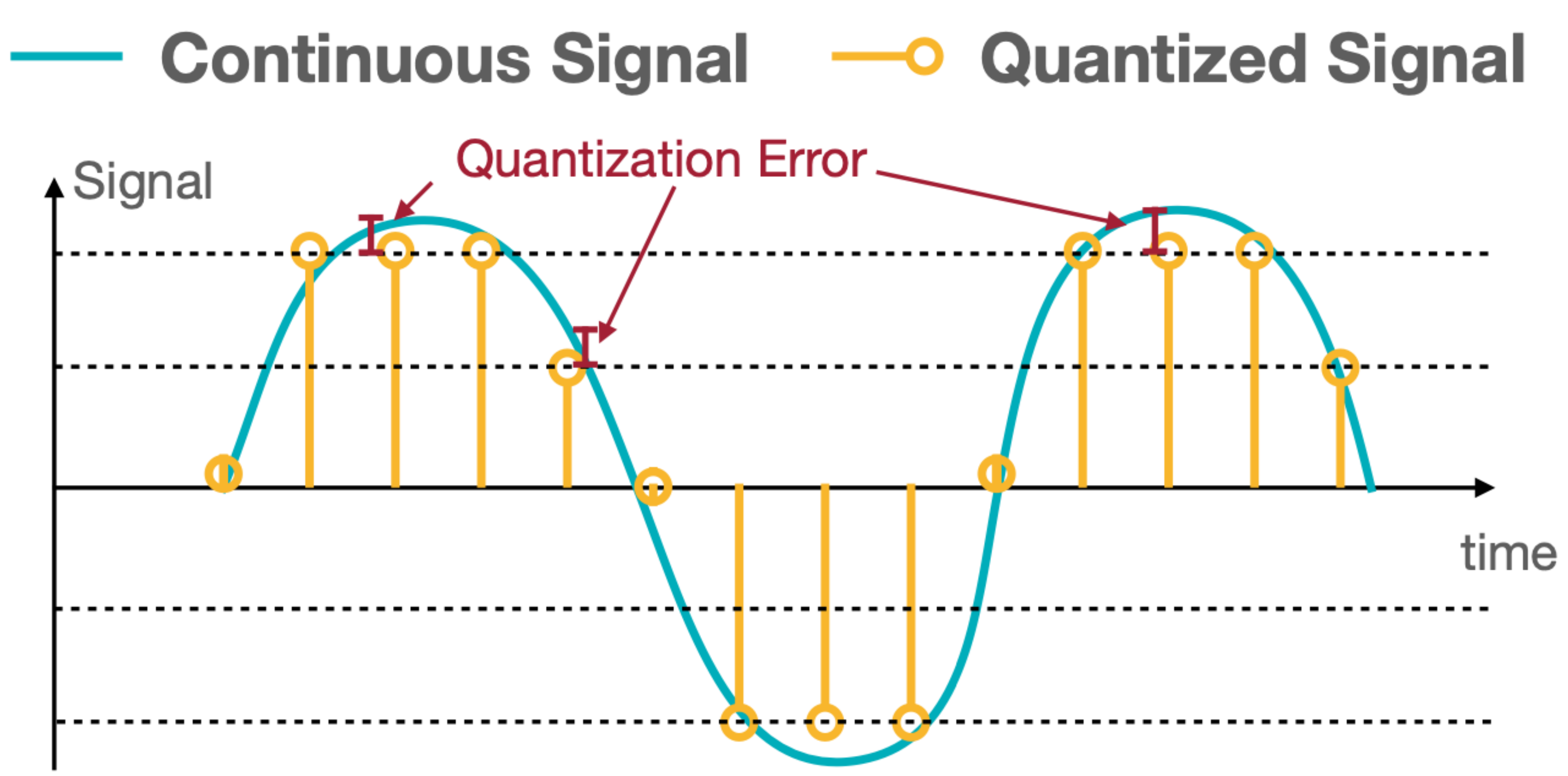
-0,-1,-2,-4,-8,-16,-32,-64
0, 1, 2, 4, 8, 16, 32, 64 ×0.25

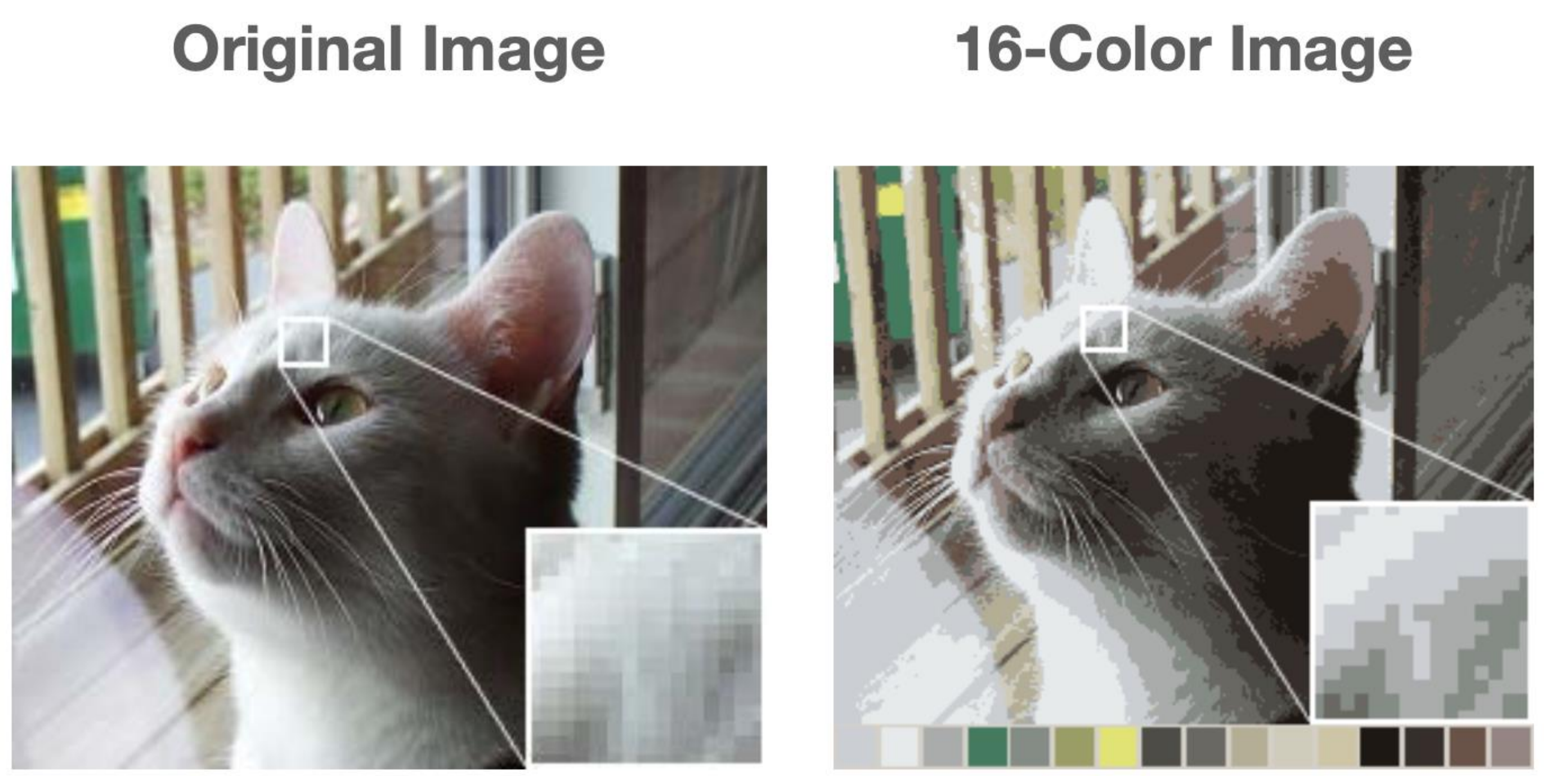
Quantization

- Digital representation of data
- **Basics of quantization**
- Quantization in ML
  - Post-training quantization
  - Quantization aware training
- Mixed precision training

What is Quantization

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



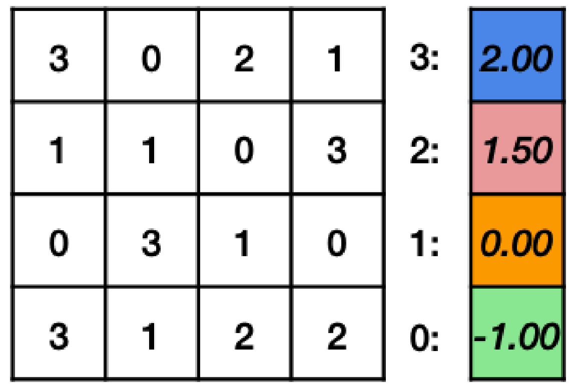


Images are in the public domain.

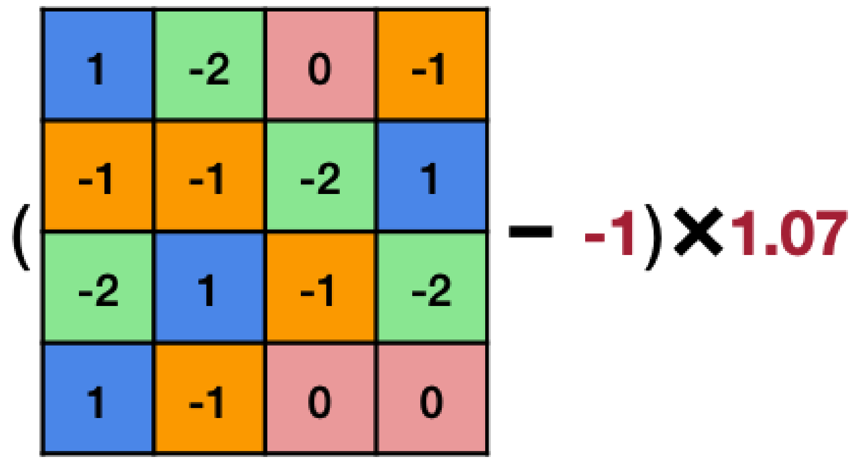
"Palettization"

Quantization Basics

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



K-Means-based Quantization



Linear Quantization

Storage	Floating point weights
Compute	Floating point arithmetic

K-means Quantization

weights
(32-bit 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

2.09, 2.12, 1.92, 1.87

↓

2.0

quantization error: 0.09, 0.12, -0.08, -0.13

K-means Quantization: Clustering

weights
(32-bit 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

cluster

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

indexes

centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

codebook

reconstructed weights
(32-bit float)

2.00	-1.00	1.50	0.00
0.00	0.00	-1.00	2.00
-1.00	2.00	0.00	-1.00
2.00	0.00	1.50	1.50

quantization error

0.09	0.02	-0.02	0.09
0.05	-0.14	-0.08	0.12
0.09	-0.08	0	-0.03
-0.13	0	0.03	-0.01

storage

32bits x 16 = 512 bit = 64bytes

2 bit x 16 = 32bit = 4 bytes

32 bit x 4 = 128bit = 16 bytes

3.2x reduction

K-means Quantization: Clustering

weights
(32-bit 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

cluster

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

indexes

centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

codebook

reconstructed weights
(32-bit float)

2.00	-1.00	1.50	0.00
0.00	0.00	-1.00	2.00
-1.00	2.00	0.00	-1.00
2.00	0.00	1.50	1.50

quantization error

0.09	0.02	-0.02	0.09
0.05	-0.14	-0.08	0.12
0.09	-0.08	0	-0.03
-0.13	0	0.03	-0.01

Assume: N-bit quantization, and #entries = M >> 2^N

storage   32bits x M = 32M bit   N bit x M= NM bit   32 bit x 2^N = 2^(N+5) bits

32M / NM = 32/N x reduction

K-means Quantization: Backward

weights
(32-bit 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

gradient

-0.03	-0.01	0.03	0.02
-0.01	0.01	-0.02	0.12
-0.01	0.02	0.04	0.01
-0.07	-0.02	0.01	-0.02

cluster →

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

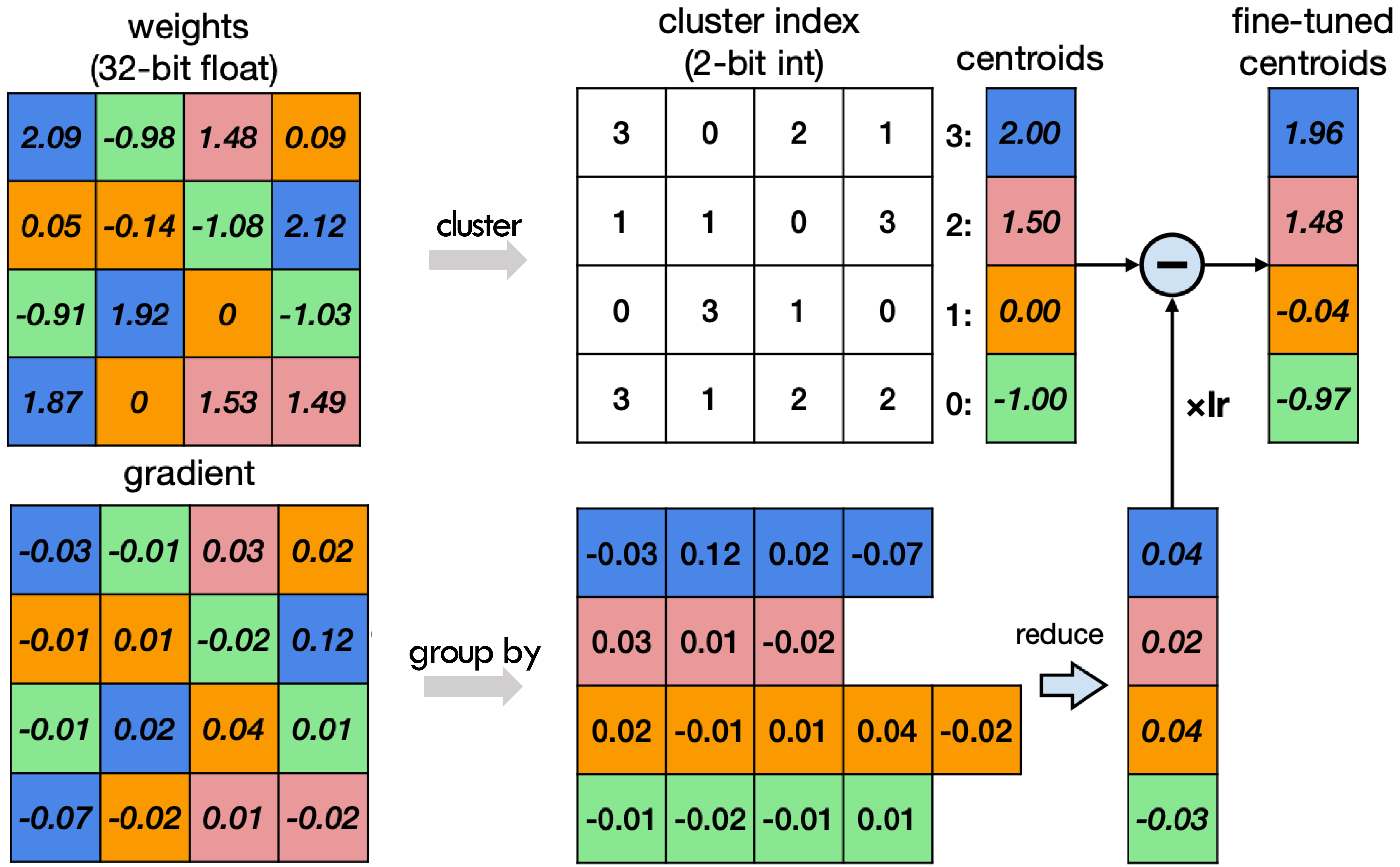
indexes

centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

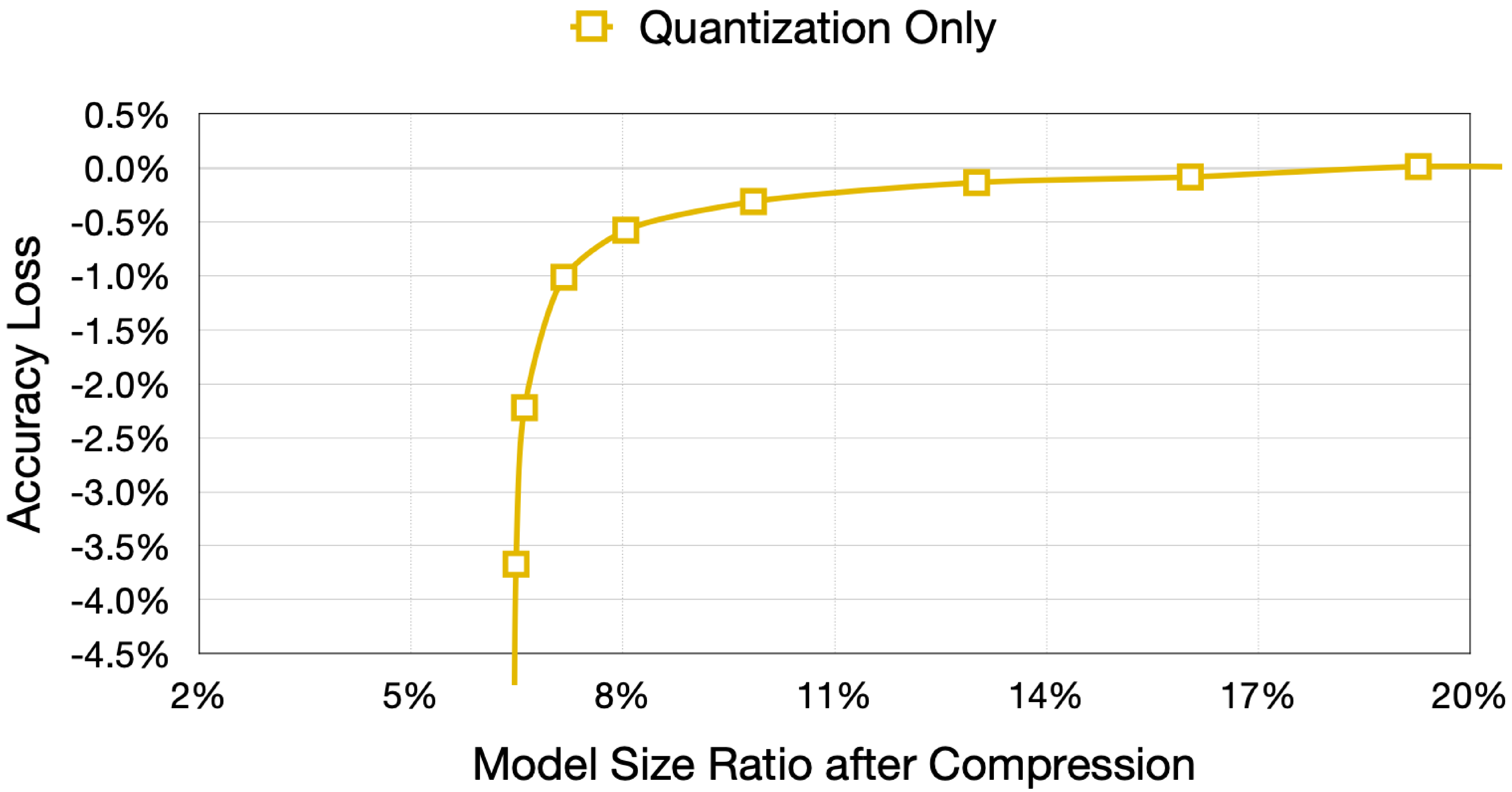
codebook

K-means Quantization: Backward

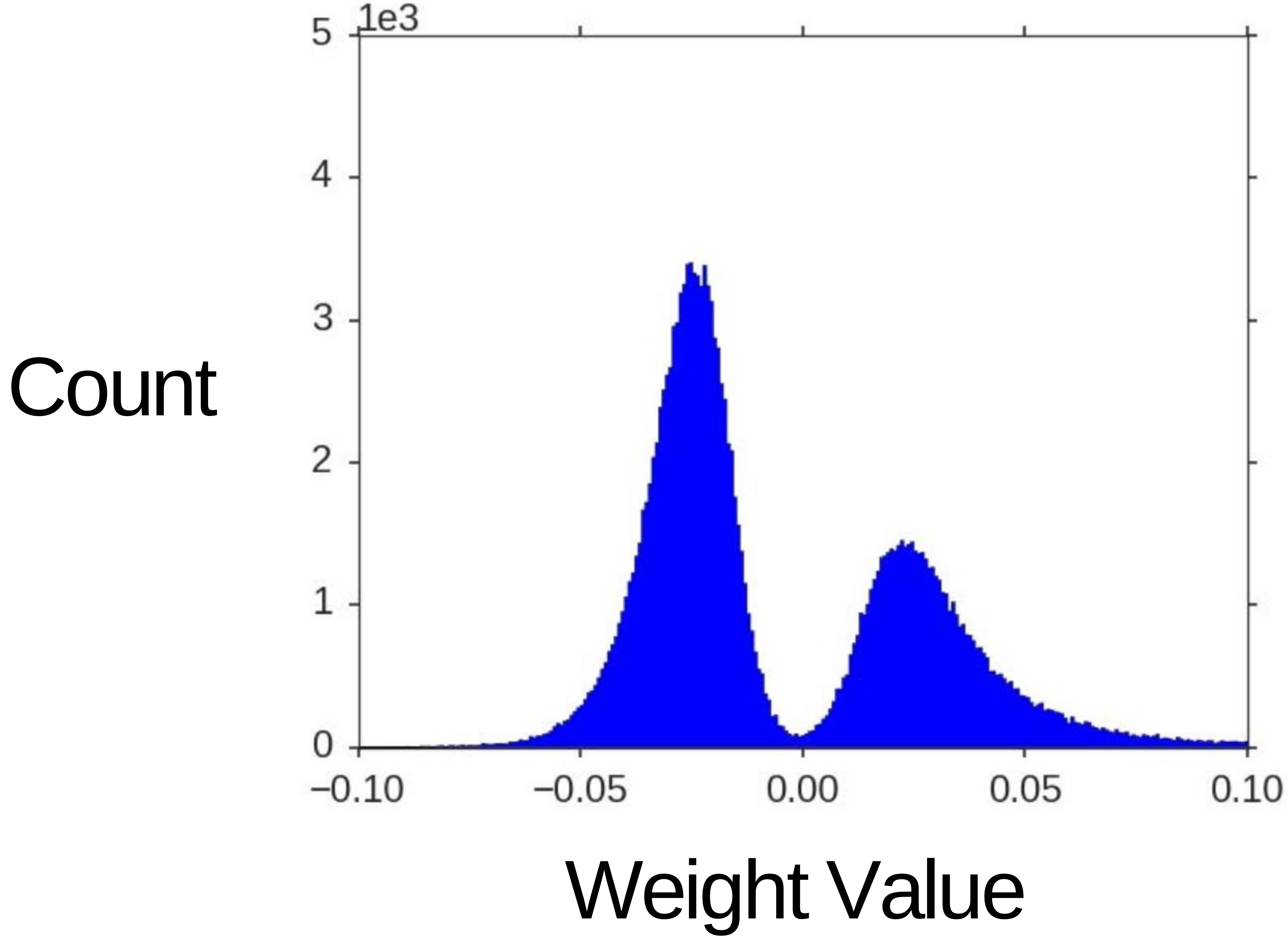


K-means Quantization:

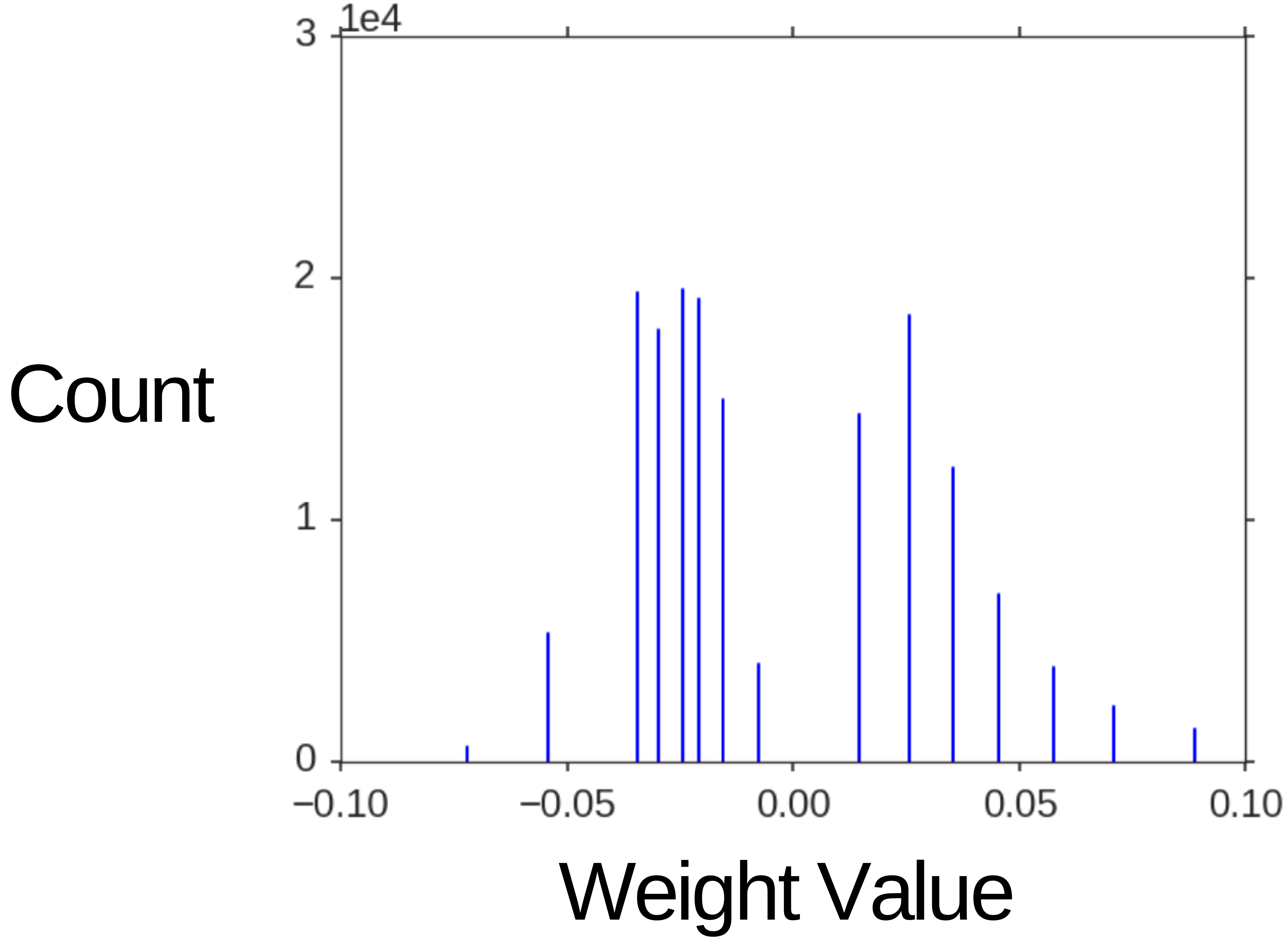
- **Accuracy vs. compression rate for AlexNet on ImageNet dataset**



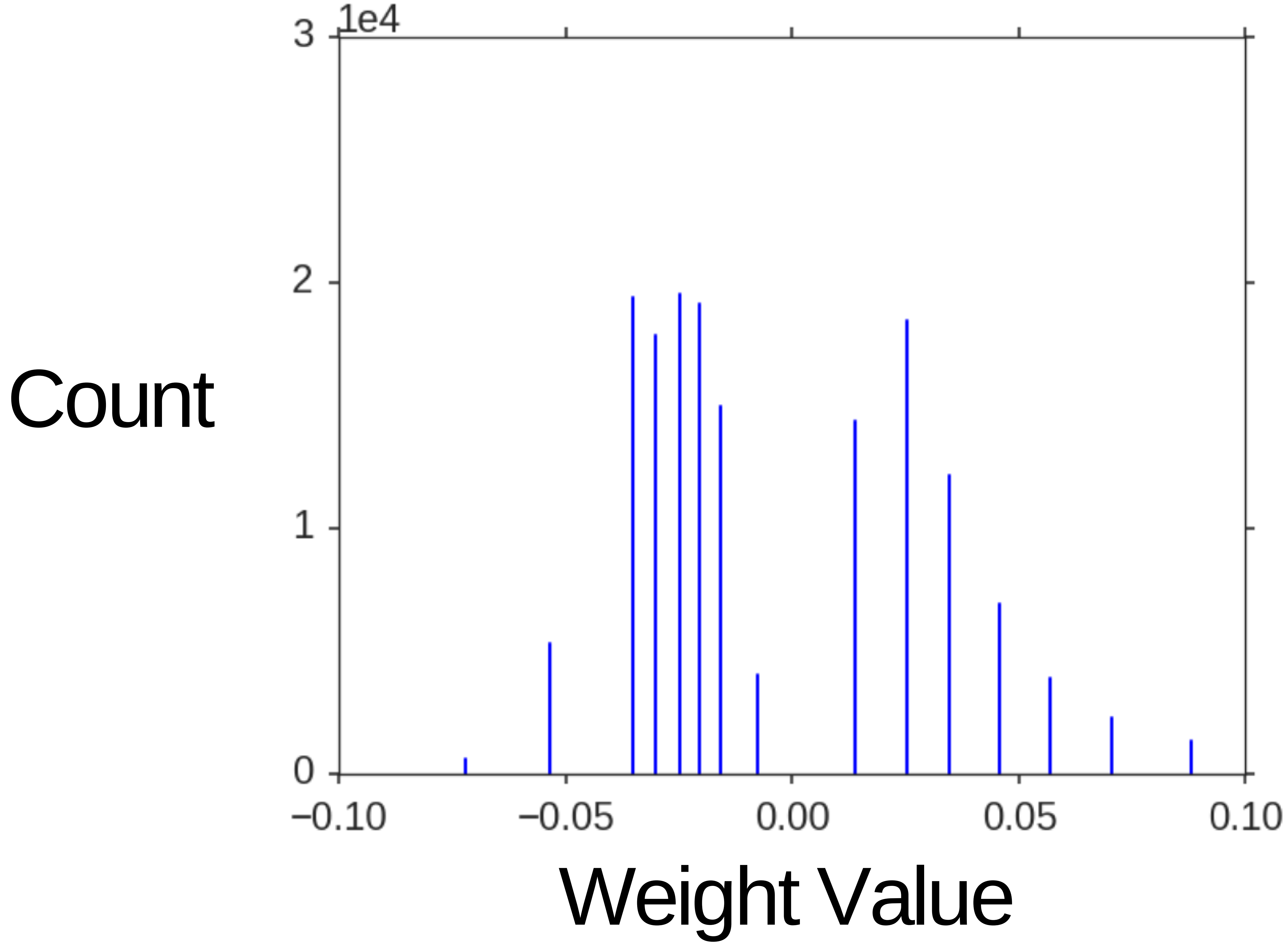
Before Quantization: Continuous Weights



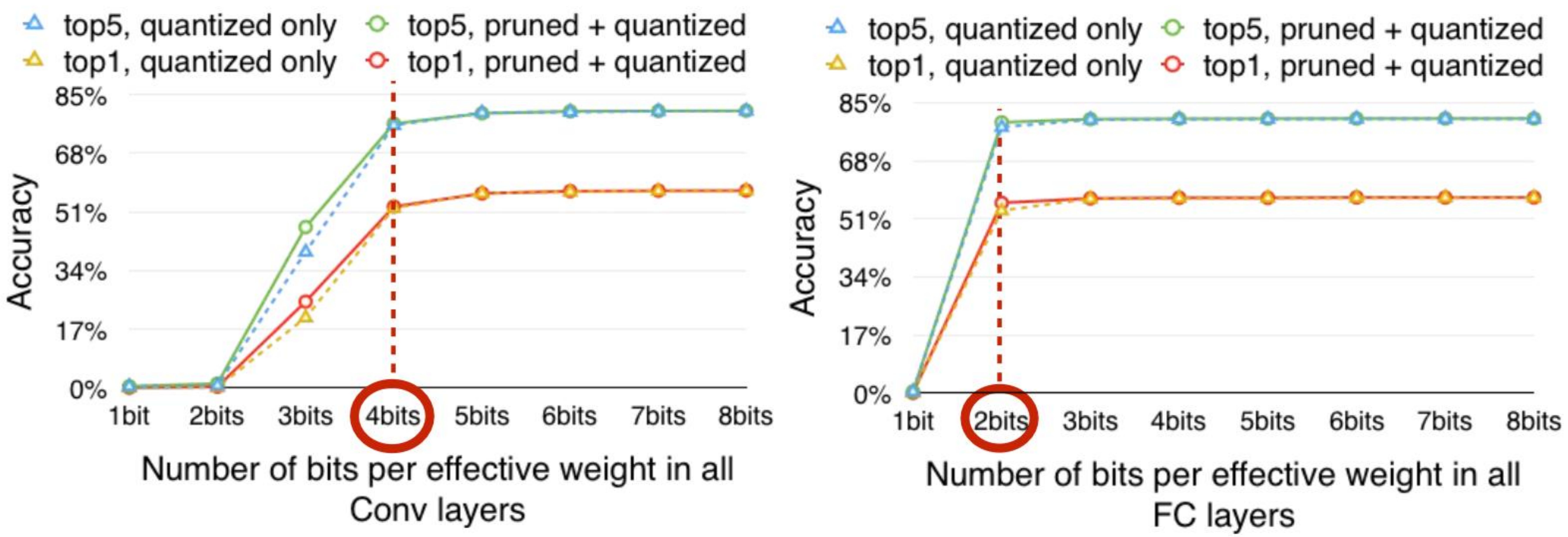
After Quantization: Discrete Weights



After Quantization: Weights Shift after training

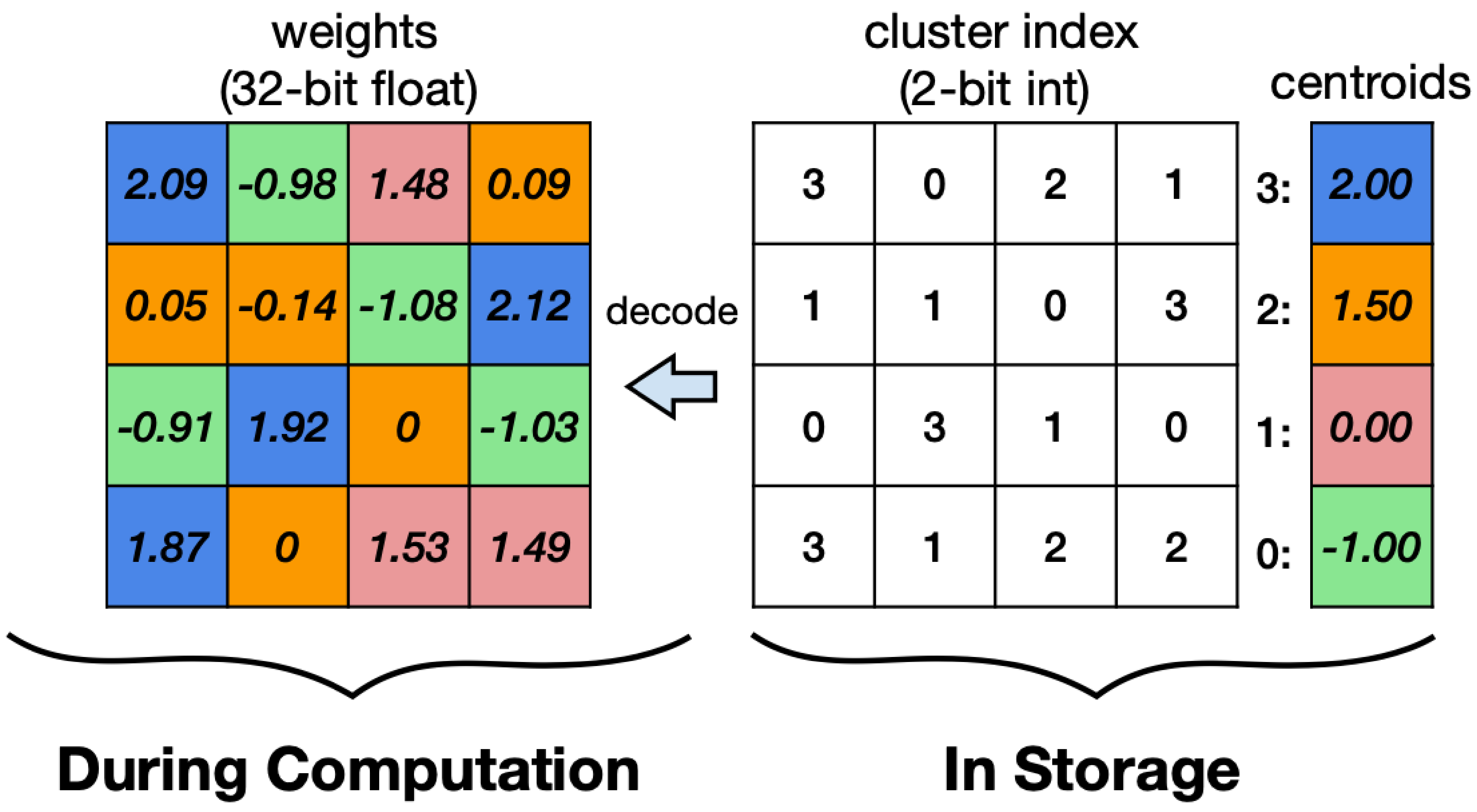


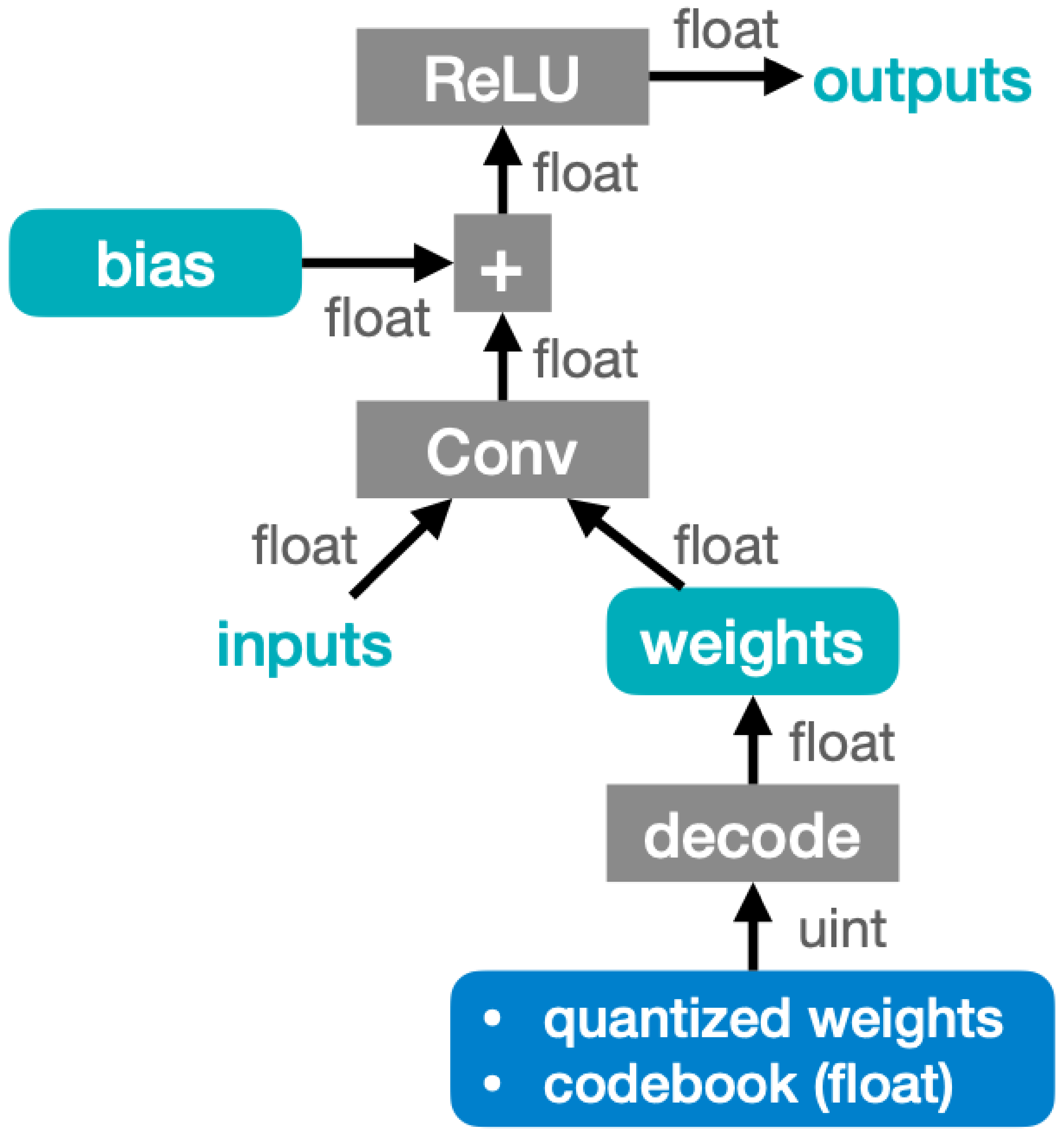
How Many Bits do We Need?



K-Means Quantization: Runtime

- Weights are decompressed using a lookup table (*i.e.*, codebook) at runtime. inference.
- Storage: quantized
- Compute: still float-point arithmetic





Quantization Basics

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

3	0	2	1	3:	2.00
1	1	0	3	2:	1.50
0	3	1	0	1:	0.00
3	1	2	2	0:	-1.00

K-Means-based Quantization

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1\right) \times 1.07$$

Linear Quantization

Storage	Floating point weights	integer weights; floating-point codebook	
Compute	Floating point arithmetic	Floating point arithmetic	

Linear Quantization

weights
(32-bit 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*

Linear Quantization

- A linear mapping of integers to real numbers

weights
(32-bit 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

quantized weights (2-bit signed int) · zero point (2-bit signed int) · scale (32-bit float) · reconstructed weights (32-bit float)

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - (-1)\right) \times 1.07 = \begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}$$

we will learn how to determine these parameters later

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

Binary	Decimal
01	1
00	0
11	-1
10	-2

Linear Quantization

- Critical parameters to determine:
  - Zero point: Z
  - Scale: S

$$
\begin{array}{ccccccc}
\begin{array}{c}\text{weights}\\\text{(32-bit float)}\end{array} & & \begin{array}{c}\text{quantized weights}\\\text{(2-bit signed int)}\end{array} & & \begin{array}{c}\text{zero point}\\\text{(2-bit signed int)}\end{array} & & \begin{array}{c}\text{scale}\\\text{(32-bit float)}\end{array} \\
\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} & \Rightarrow \Bigg(& \begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} & - & -1 & \Bigg) \times & 1.07 \\
r & = (& q & - & Z &) \times & S
\end{array}
$$

Linear Quantization: $r = S(q - Z)$

weights (32-bit float)	quantized weights (2-bit signed int)	zero point (2-bit signed int)	scale (32-bit float)

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \Rightarrow \left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - \mathbf{-1} \right) \times \mathbf{1.07}$$

$$r \quad = (\quad q \quad - \quad Z \quad) \times \quad S$$

Floating-point ⟵ integer

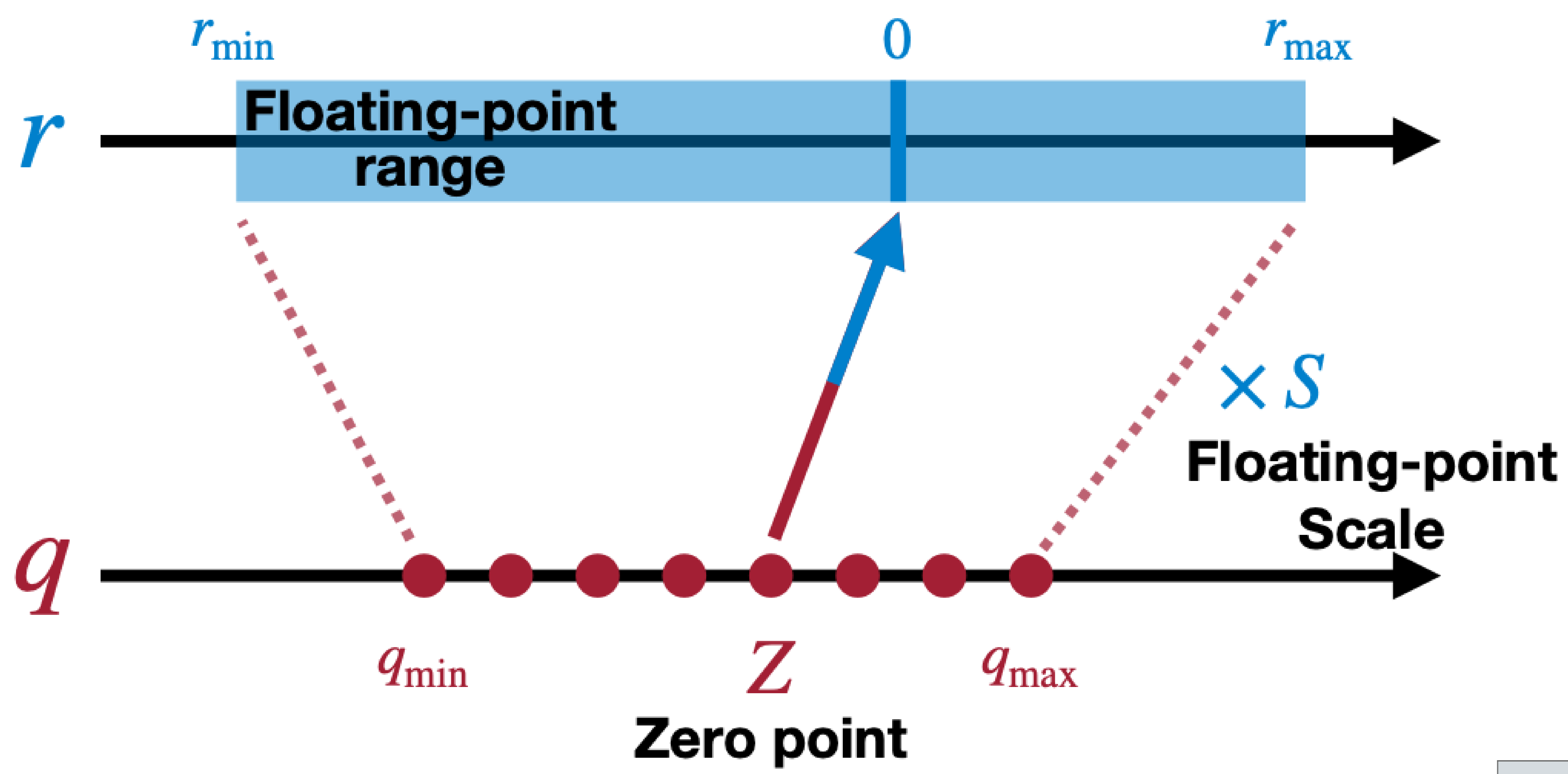
Integer (zero point)

- quantization parameters
- allow real number $r = 0$ be represented by a quantized integer Z

floating point

- quantization parameters

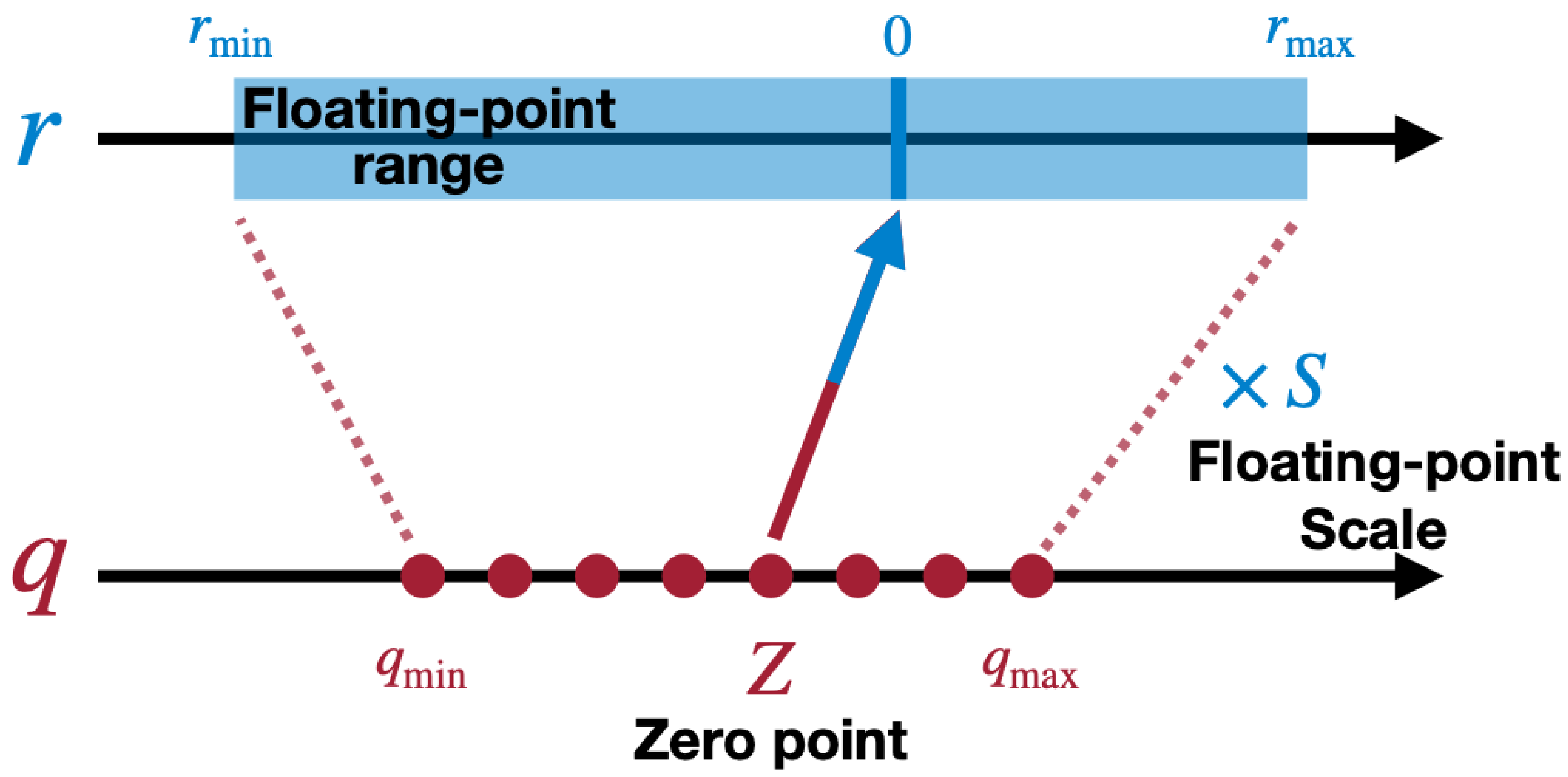
$r = S(q - Z)$: Geometric Interpretation



Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Q: How to determine S and Z?

$r = S(q - Z)$: Determine S and Z

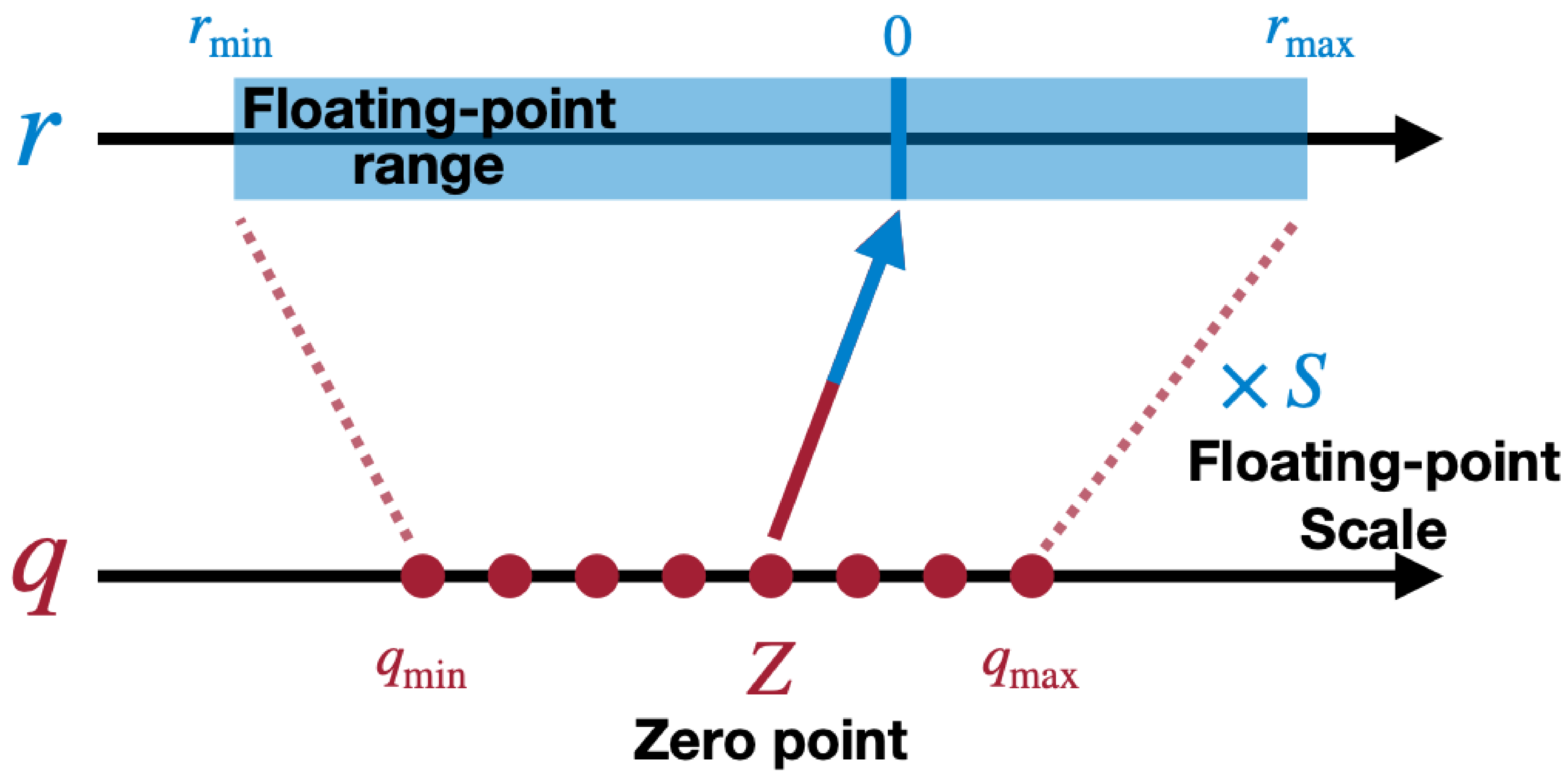


$$r_{\max} = S(q_{\max} - Z)$$
$$r_{\min} = S(q_{\min} - Z)$$

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

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

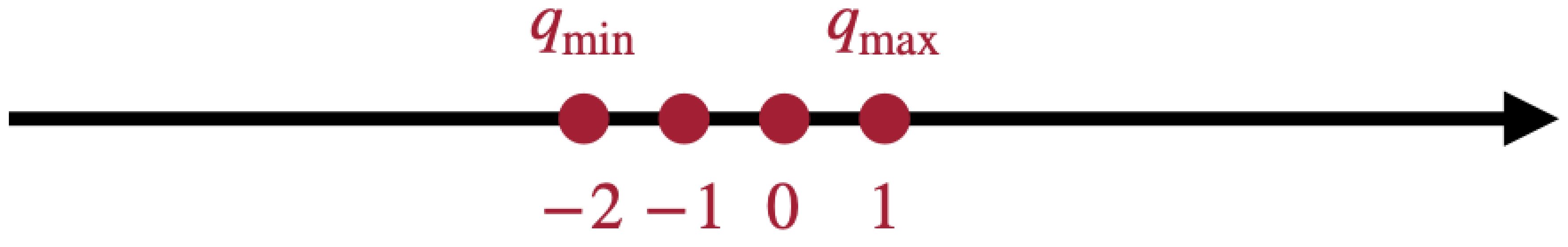
$r = S(q - Z)$: Determine S and Z



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

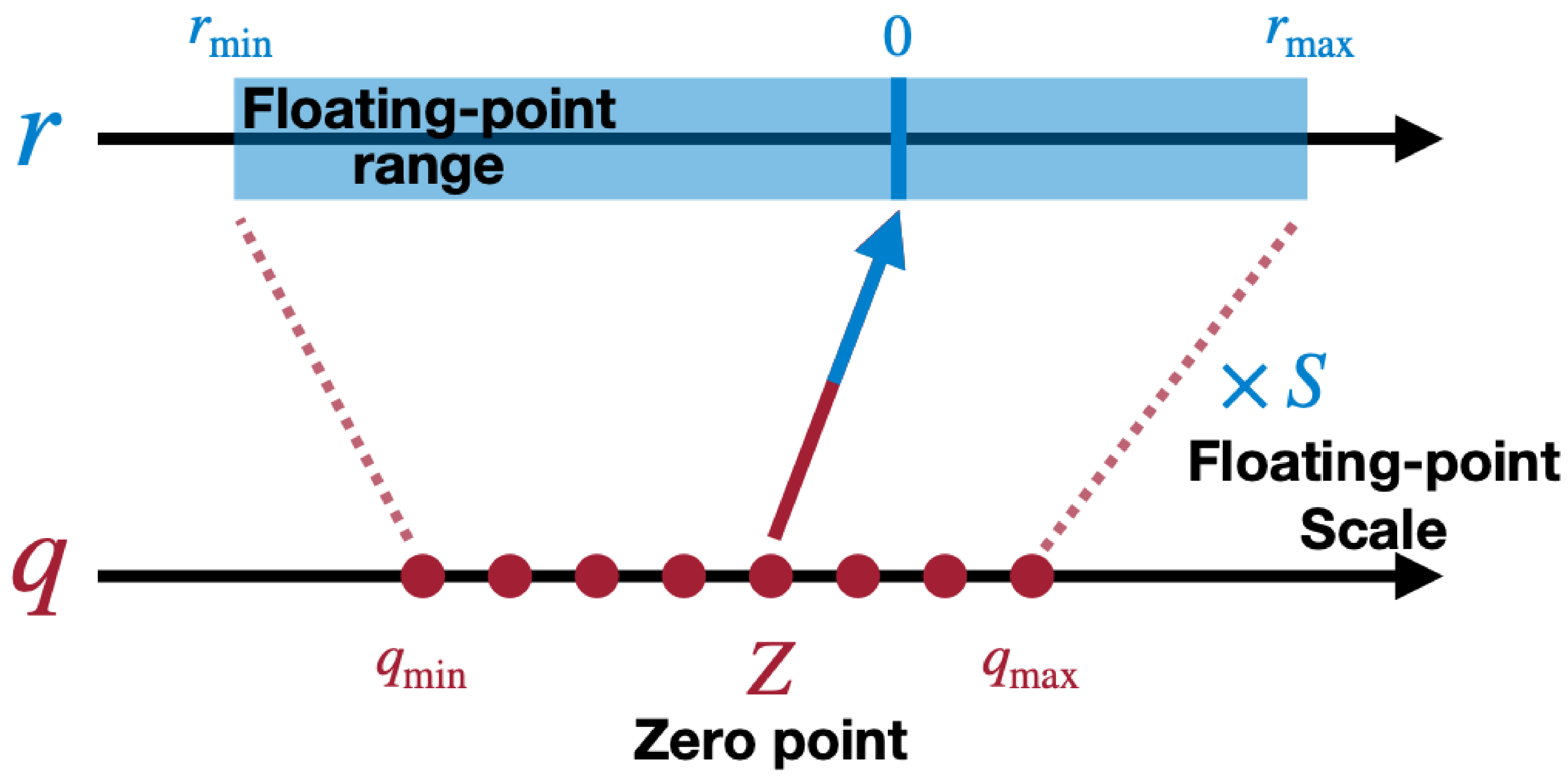
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

Binary	Decimal
01	1
00	0
11	-1
10	-2



$$S = \frac{2.12 - (-1.08)}{1 - (-2)} = 1.07$$

$r = S(q - Z)$: Determine S and Z



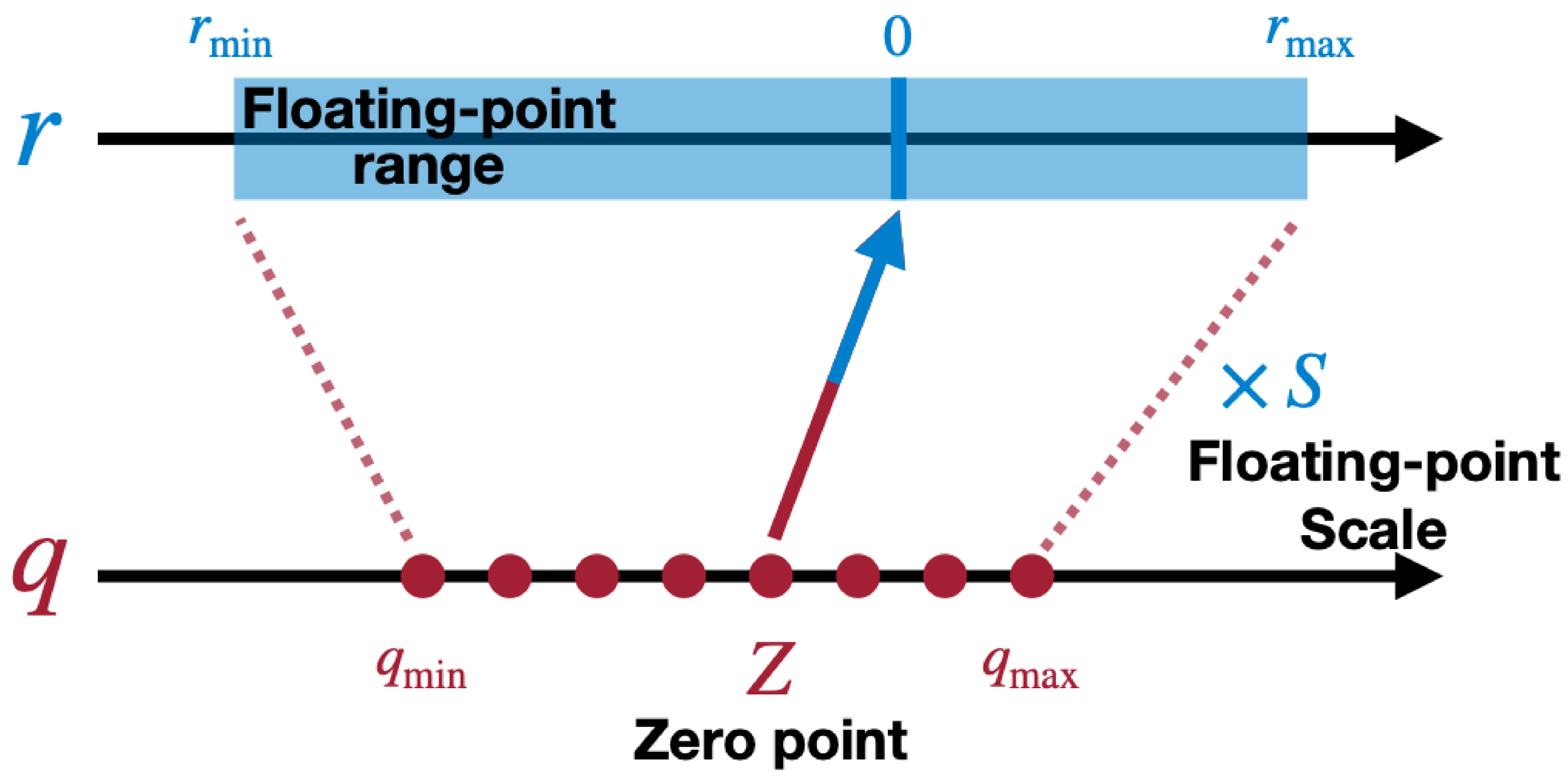
$$r_{\max} = S(q_{\max} - Z)$$

$$Z = q_{min} - \frac{r_{min}}{S}$$

$$Z = \mathrm{round}(q_{min} - \frac{r_{min}}{S})$$

$r = S(q - Z)$: Determine S and Z

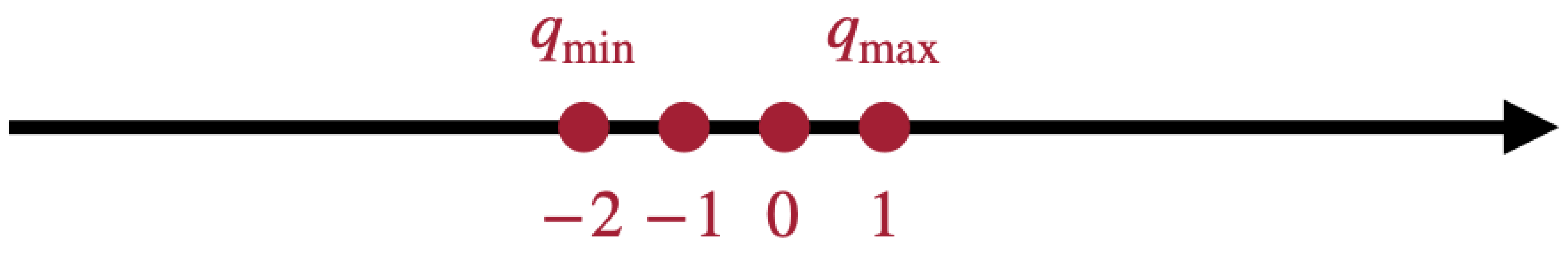
2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49



$$Z = \text{round}\left(q_{min} - \frac{r_{min}}{S}\right)$$

$$= \text{round}\left(-2 - \frac{-1.08}{1.07}\right) = -1$$

Binary	Decimal
01	1
00	0
11	-1
10	-2



Apply Linear Quantization into Matmul

$$Y = WX$$

$$S_Y(q_Y - Z_Y) = S_W(q_W - Z_W)S_X(q_X - Z_X)$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W - Z_W)(q_X - Z_X) + Z_Y$$

$$q_Y = \frac{S_W S_X}{S_Y}(q_W q_X - Z_W q_X - Z_X q_W + Z_W Z_X) + Z_Y$$

Apply Linear Quantization into Matmul

$$q_Y = \boxed{\frac{S_W S_X}{S_Y}}(q_W q_X - Z_W q_X - \boxed{Z_X q_W + Z_W Z_X}) + \boxed{Z_Y}$$

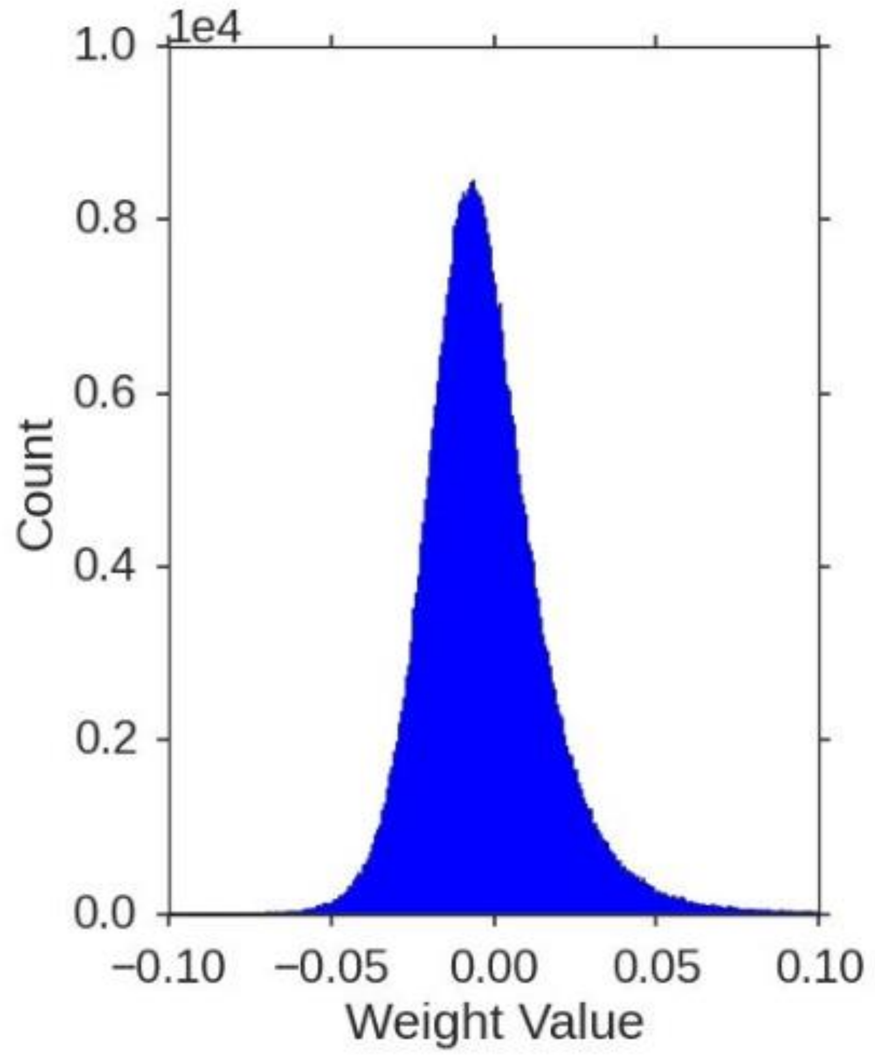
- precomputed;
- N-bit integer multiplication
- 32-bit integer addition/subtraction

- Empirically, $\frac{S_W S_X}{S_Y} \in (0, 1)$
- $\frac{S_W S_X}{S_Y} = 2^{-n} M_0, M_0 \in [0.5, 1)$ using fixed point multiplication and bit shift

Apply Linear Quantization into Matmul

$$q_Y = \boxed{\frac{S_W S_X}{S_Y}}\left(q_W q_X - \boxed{Z_W q_X} - \boxed{Z_X q_W + Z_W Z_X}\right) + \boxed{Z_Y}$$

- precomputed;
- N-bit integer multiplication
- 32-bit integer addition/subtraction



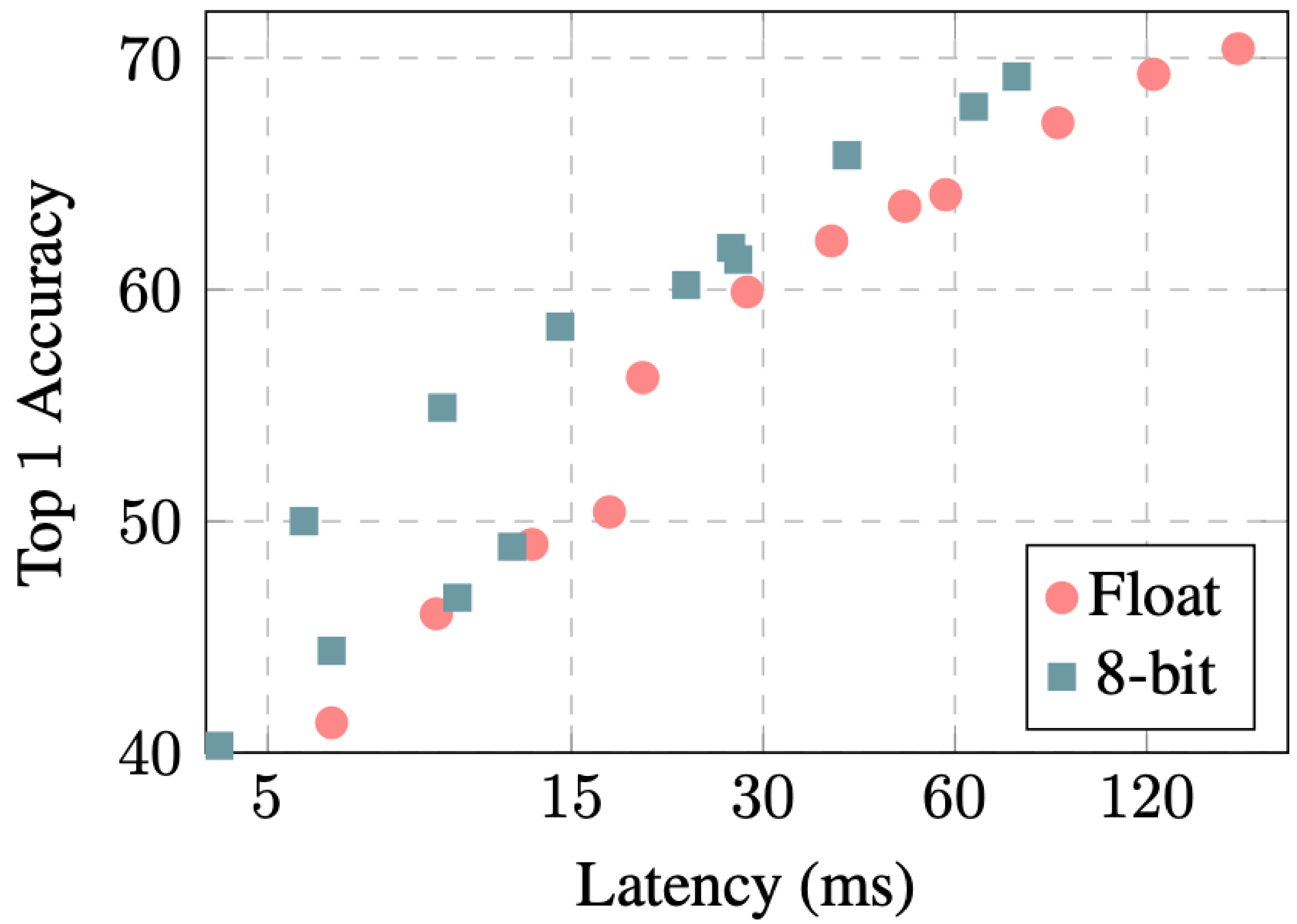
Empirically: $Z_W = 0$

$$q_Y = \frac{S_W S_X}{S_Y}\left(\boxed{q_W q_X} - Z_X q_W\right) + Z_Y$$

- Heavy lifting part
- integer multiplication

INT8 Linear Quantization Performance

Neural Network	ResNet-50	Inception-V3
Floating-point Accuracy	76.4%	78.4%
8-bit Integer-quantized Acurracy	74.9%	75.4%



Latency-vs-accuracy tradeoff of float vs. integer-only MobileNets on ImageNet using Snapdragon 835 big cores.

Quantization Basics

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

3	0	2	1	3:	2.00
1	1	0	3	2:	1.50
0	3	1	0	1:	0.00
3	1	2	2	0:	-1.00

K-Means-based Quantization

$$\left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1\right) \times 1.07$$

Linear Quantization

Storage	Floating point weights	integer weights; floating-point codebook	integer weights;
Compute	Floating point arithmetic	Floating point arithmetic	Integer arithmetic

Next Lecture

- Post-training quantization
- Mixed precision
- Parallelization