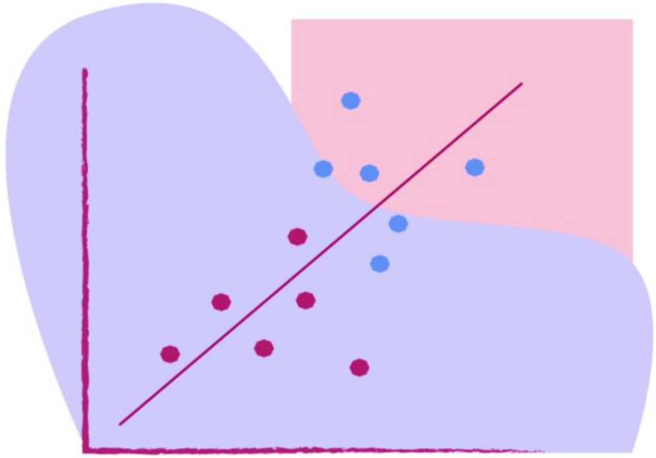
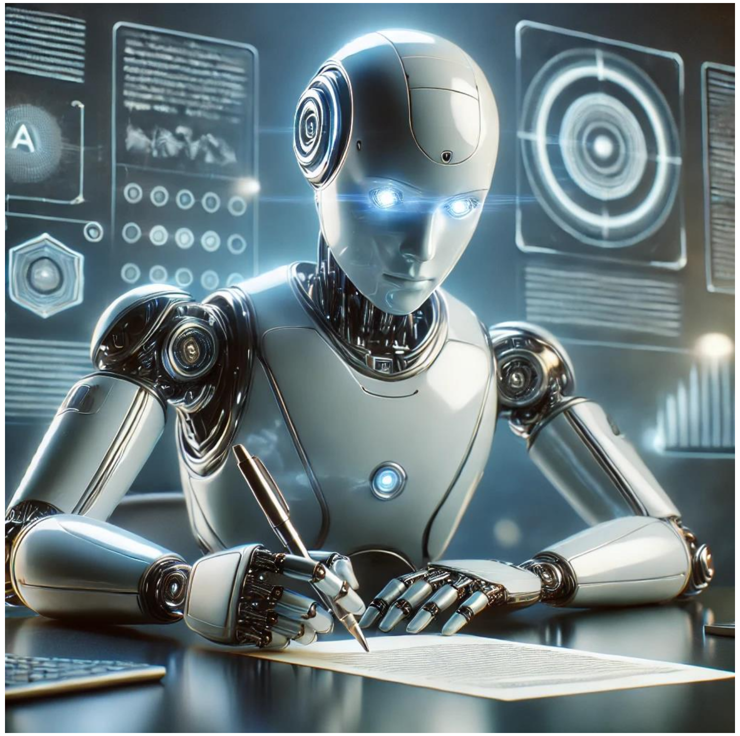
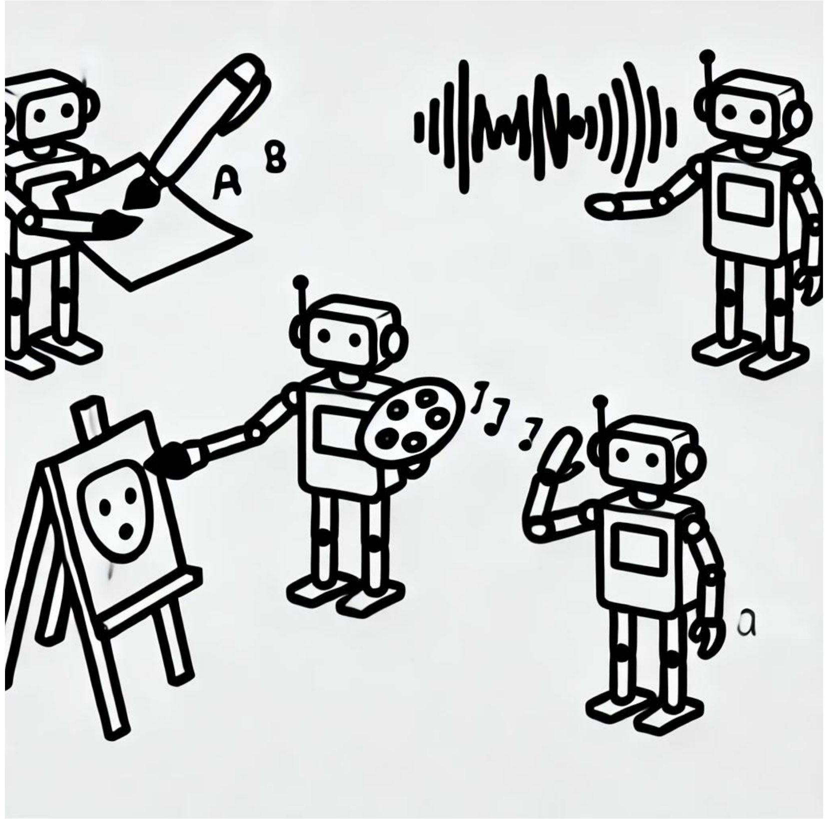
Generative AI models







Generative AI models

Transformer

Transformers do not process sequences of data one step at a time (as RNNs or LSTMs do). Instead, they use self-attention to weigh the relationships between all tokens in a sequence simultaneously.

- Self-Attention Mechanism
- Multi-Head Attention
- Positional Encoding

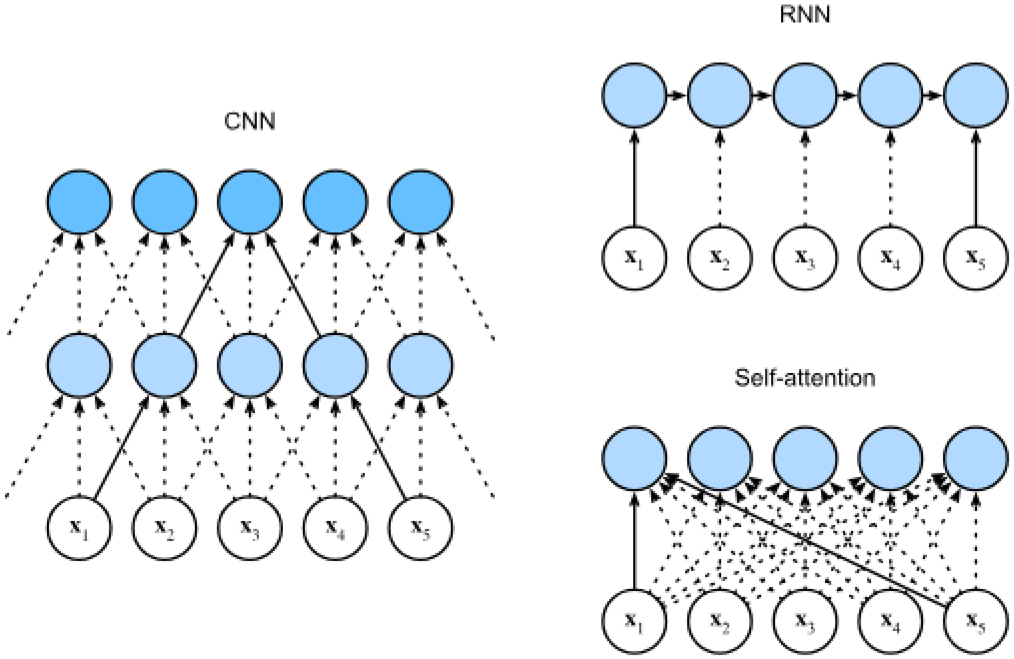


Image source : https://d2l.ai/chapter_attention-mechanisms-and-transformers/self-attention-and-positional-encoding.html

Generative AI models

Why Gen AI models are so large?

Larger models tend to perform better, they are computationally expensive to train and deploy

1. **High Number of Parameters**
2. **Complex Architecture**
3. **Large Vocabulary and Embeddings**

Model Optimization Techniques

Types

1. **Distillation**
2. **Quantization**
3. **Pruning**

Pruning

What is Pruning?

Pruning is a technique used to reduce the size of a neural network by removing parameters—specifically, weights or connections—that don't contribute much to the network's overall performance

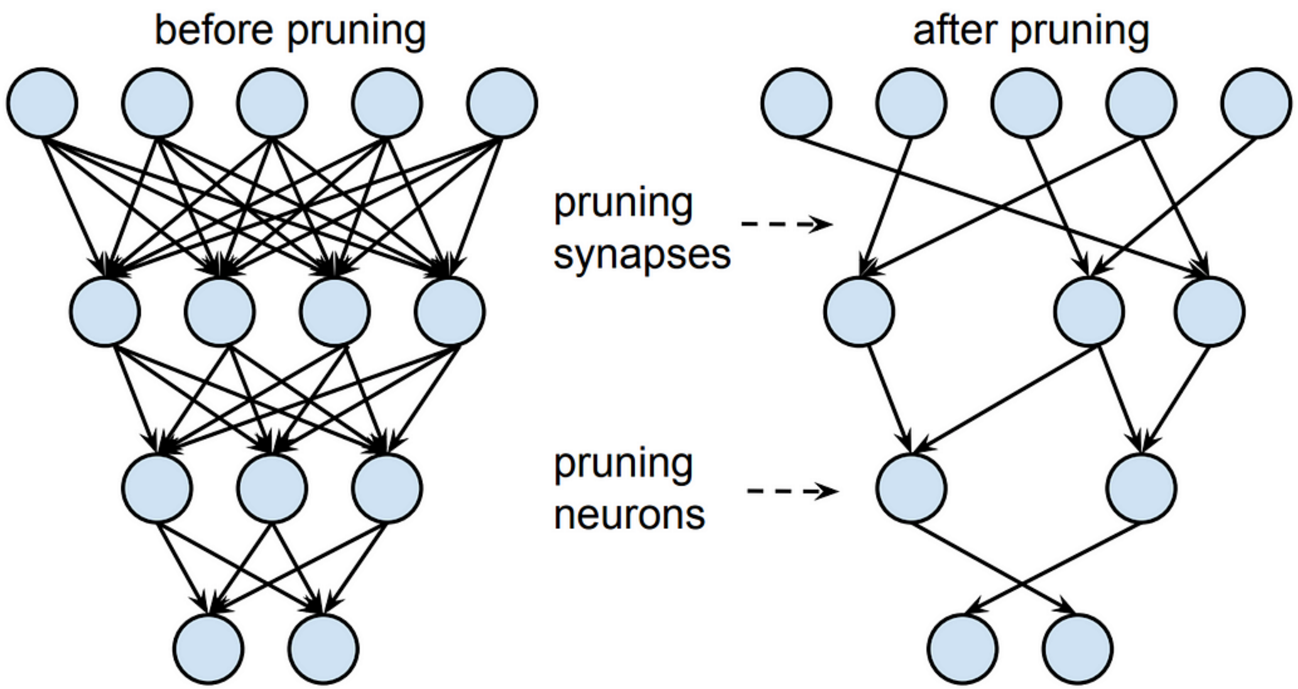
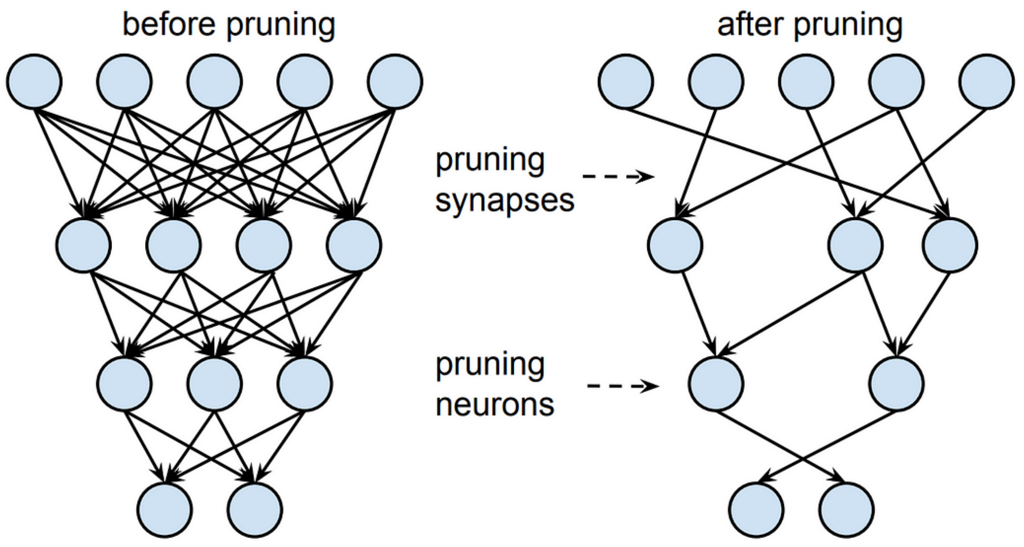


Image source : https://arxiv.org/abs/1506.02626

Pruning

Types of Pruning



1. **Unstructured Pruning**
2. **Structured Pruning**

Image source : https://arxiv.org/abs/1506.02626

Pruning

How Pruning Works

1. **Training the Full Model**
2. **Identify Unimportant Weights**
3. **Magnitude / WANDA Pruning**

   In magnitude pruning, we rank all the weights in the model by their absolute value. We assume that weights with smaller absolute values are less important, so we prune—or remove—those weights
4. **Fine-Tuning**

Image source : https://arxiv.org/abs/1506.02626

Distillation

Knowledge Distillation

Knowledge distillation is a technique that transfers knowledge from a large, complex model, often called the teacher, to a smaller, simpler model, known as the student.

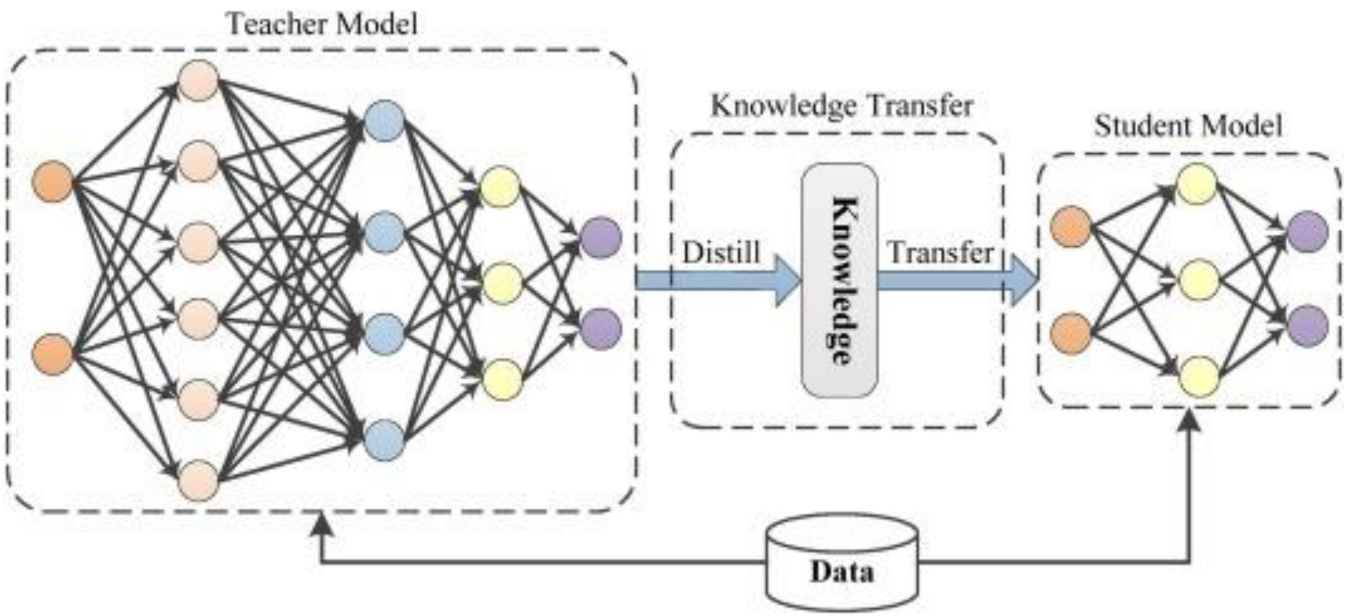
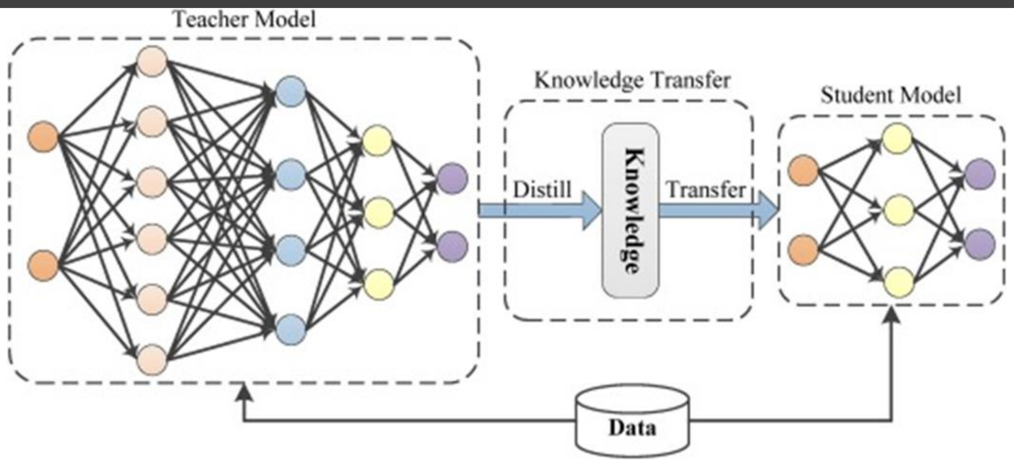


Image source : https://www.sciencedirect.com/science/article/abs/pii/S0020025523001950

Distillation

How Does Knowledge Distillation Work?



1. **Train the Teacher Model**
2. **Soft Labels**
3. **Train the Student Model**

   The smaller student model is trained using two types of information:

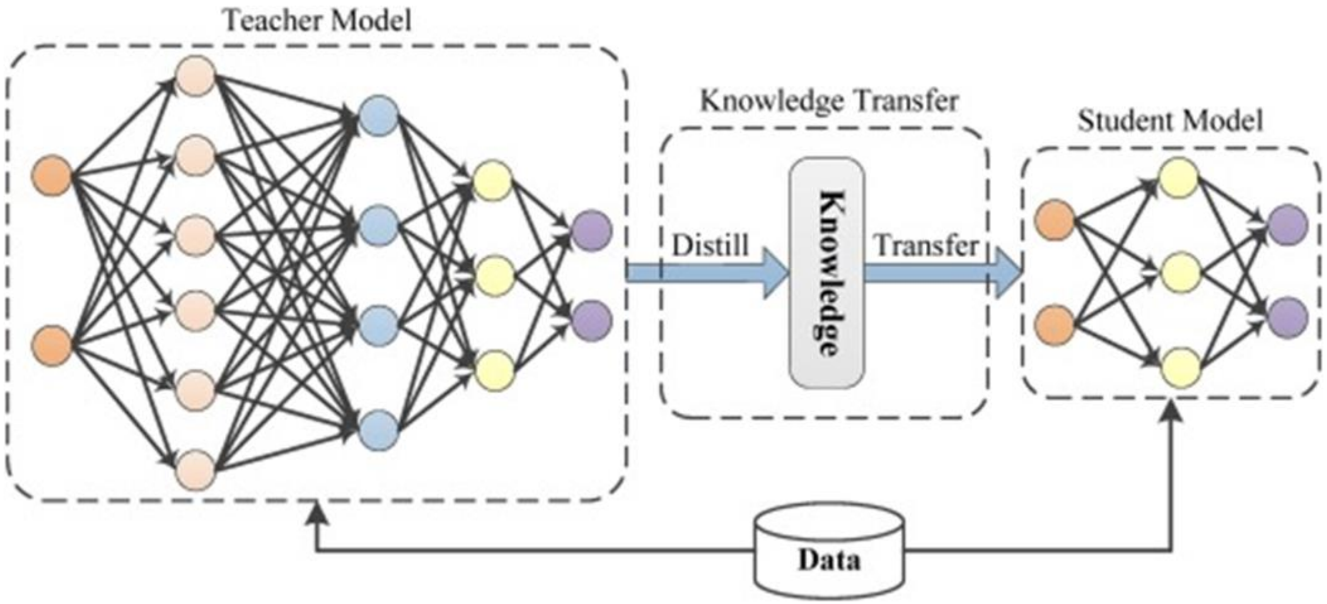
   - The original hard labels from the dataset.
   - The soft labels provided by the teacher model.

4. **Loss Function**

Image source : https://www.sciencedirect.com/science/article/abs/pii/S0020025523001950

Distillation

Types of Knowledge Distillation



1. **Logit Distillation**
2. **Feature-Based Distillation**

Image source : https://www.sciencedirect.com/science/article/abs/pii/S0020025523001950

Quantization

What is Quantization?

Quantization refers to the process of reducing the precision of the weights and activations in a neural network.

Floating Point → **Integer**

1231.4531 → 1231

32 bit

2.34	0	-1.54
5.40	6.22	3.81
-2.10	-4.32	2.88

Quantization →

8 bit

2	0	-2
5	6	4
-2	-4	3

Quantization

How Does Quantization Work?

There are two main types of quantization:

- Post-Training Quantization
- Quantization-Aware Training.

Post-Training Quantization (PTQ)

1. Train the Model:
2. Convert the Weights
3. Deploy the Quantized Model

Quantization

How Does Quantization Work?

There are two main types of quantization:

- Post-Training Quantization
- Quantization-Aware Training.

Quantization-Aware Training (QAT)

1. Simulate Quantization During Training
2. Train the Model with Quantization
3. Deploy the Quantized Model

Quantization

Quantization Types

1. Integer Quantization (INT8):
2. Half-Precision Floating Point (FP16):
3. Mixed Precision Quantization

Quantization

Challenges in Quantization

1. Accuracy Loss
2. Hardware Compatibility
3. Quantization Granularity

Quantization

Benefits of Quantization

1. Smaller Model Sizes
2. Faster Inference
3. Cost and Energy Efficiency

Data Types and Number Representation

Data Types in Neural Networks

1. **Integer Representation**

   Integers are simple, whole numbers without any fractional component.

2. **Floating-Point Representation**

   Floating-point numbers can represent both large and small numbers, including fractions

Integer Data Types

Basics

Bit

A bit is the smallest unit of data in computing, and it can have one of two values: 0 or 1

4-bit integers

0	1	0	1

Integer Data Types

4-Bit Representation

8s	4s	2s	1s
2^3	2^2	2^1	2^0

Integer Data Types

4-Bit Unsigned Integer

An unsigned integer can only represent positive numbers, and all the bits are used for the value itself

1	0	1	0
×8	×4	×2	×1
8	0	2	0

8 + 0 + 2 + 0 = 10

Integer Data Types

Range of 4-Bit Unsigned Integers

0	**0**	**0**	**0**	=	**0**
0	0	0	1	=	**1**
0	0	1	0	=	**2**
…	…	…	…		
1	1	1	1	=	**15**

Integer Data Types

4-Bit Signed Integers

A signed integer can represent both positive and negative numbers.

To do this, we need to reserve one of the bits to indicate whether the number is positive or negative

+/-	4s	2s	1s
+/-	2^2	2^1	2^0

Integer Data Types

4-Bit Signed Integers

Signed magnitude representation

1	1	0	1
-/+	×4	×2	×1
-	4	0	1

- (4 + 0 + 1) = -5

Range is -7 (1111) to +7 (0111)

Integer Data Types

Two's complement

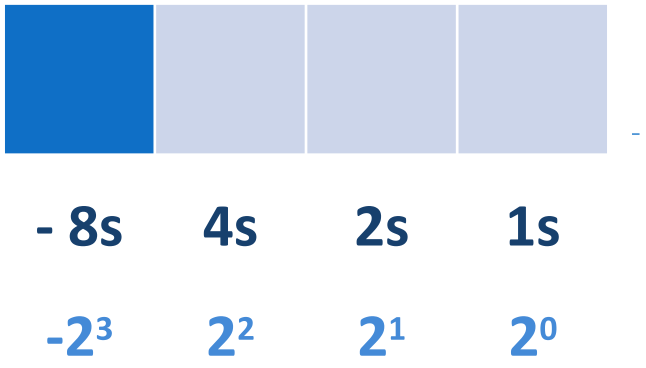
The leftmost bit (the most significant bit) has a negative sign as well the magnitude of highest power of 2

- 8s	4s	2s	1s
-2^3	2^2	2^1	2^0

Integer Data Types

Two's complement

The leftmost bit (the most significant bit) has a negative sign as well the magnitude of highest power of 2



Integer Data Types

Two's complement

-3 in Two's complement representation

1	1	0	1
- 8s	**4s**	**2s**	**1s**
-2^3	2^2	2^1	2^0

-8 + 4 + 0 + 1 = -3

Integer Data Types

Range of 4-Bit Two's complement

0	0	0	0	=	0
0	0	0	1	=	1
0	1	1	1	=	7
…	…	…	…		
1	0	0	0	=	-8

Range is -8 (1000) to +7 (0111)

Integer Data Types

Adding +5 (positive) and -3 (negative)

Adding two numbers

0	1	0	1	= 5
1	1	0	1	= -3

1	0	0	1	0

0	0	1	0

= 2

Integer Data Types

Unsigned 8-Bit Integers (INT8)

The range is from 0 to 255

0	0	0	0	0	0	0	0	= 0
0	0	0	0	0	0	0	1	= 1
1	1	1	1	1	1	1	1	= 255

Integer Data Types

Signed 8-Bit Integers (UINT8)

The range is from -128 to 127

0	0	0	0	0	0	0	0	= 0
0	1	1	1	1	1	1	1	= 127
1	0	0	0	0	0	0	0	= -128

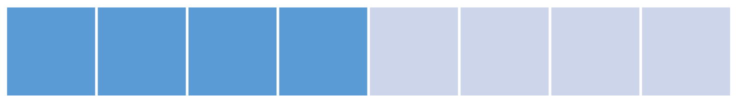
Fixed-Point Numbers

8-Bit Fixed-Point Numbers

We use 8 bits to represent a number. Some of these bits represent the integer part of the number, and some represent the fractional part, depending on how we decide to 'fix' the decimal point.

8-bit fixed-point number could be formatted as 4.4

- 4 bits for the integer part (to the left of the binary point).
- 4 bits for the fractional part (to the right of the binary point).



Integer Data Types

8-Bit Fixed-Point Numbers

0	0	1	1	0	0	1	1
8s	4s	2s	1s	1/2	1/4	1/8	1/16
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

0	0	0	0	1	0	0	0	= 0.5
0	0	0	0	0	1	0	0	= 0.25

Integer Data Types

Signed 8-Bit Fixed-Point

Representing -2.5

-3+0.5

1	1	0	1	= -3

1	0	0	0	= 0.5

1	1	0	1	1	0	0	0	= -2.5

Floating-Point Numbers

Floating-Point Numbers

A floating-point number is a way to represent real numbers in a way that can support a wide range of values by using a formula that includes a base number (called the mantissa or significand), an exponent, and a sign.

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-127)}$$

- The sign bit determines whether the number is positive or negative.
- The mantissa represents the significant digits of the number.
- The exponent controls how large or small the number is, by raising 2 to the power of the exponent.

Floating-Point Numbers

FP32
Single-precision floating point

- 1 bit for the sign
- 8 bits for the exponent
- 23 bits for the mantissa

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-127)}$$

Floating-Point Numbers

Representing a Normal Number in FP32

- We'll use the number 6.5
- The decimal number 6.5 can be written as 1.625*4 in binary.

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-127)}$$

- Sign is 0
- (1 +mantissa) is 1.625, mantissa is 0.625
- 0.625 = **10100000000000000000000**
- (Exponent -127) = 2, Exponent =129
- 129 = 10000001

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

Floating-Point Numbers

Subnormal Numbers

- Subnormal numbers (also called denormalized numbers) are used to represent numbers that are very close to zero but can’t be represented in the normalized form.
- A subnormal number is used when the exponent is all zeros.

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

- For subnormal numbers, the leading 1 in the mantissa is not assumed (like in normal numbers). Instead, the mantissa starts with a 0.

$$(-1)^{\text{sign}} \times (\text{mantissa}) \times 2^{(\text{exponent}-127)}$$

Floating-Point Numbers

Subnormal Numbers

- Assume the number is

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

$$(-1)^{\text{sign}} \times (\text{mantissa}) \times 2^{(\text{exponent}-127)}$$

$$\text{Value} = (-1)^0 \times 0.00000000000000000000001 \times 2^{-(127-1)} = 2^{-126}$$

Floating-Point Numbers

Special Cases

- **Zero:** Zero is represented by setting both the exponent and the mantissa to all zeros

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Infinity**: If the exponent is all ones and the mantissa is all zeros, the value represents infinity.

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

- **NaN**: If the exponent is all ones and the mantissa is non-zero, the value represents NaN

0	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

Other floating point formats

FP16
half-precision floating point

- 1 bit for the sign
- 5 bits for the exponent
- 10 bits for the mantissa

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-15)}$$

Floating-Point Numbers

Representing a Normal Number in FP16

- We'll use the number 6.5
- The decimal number 6.5 can be written as 1.625*4 in binary.

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-15)}$$

- Sign is 0
- (1 +mantissa) is 1.625, mantissa is 0.625
- 0.625 = **1010000000**
- (Exponent -15) = 2, Exponent =17
- 17 = 10001

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

Other floating point formats

BFloat16
Brain Floating Point

- 1 bit for the sign
- 8 bits for the exponent
- 7 bits for the mantissa

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-127)}$$

Floating-Point Numbers

BFloat16 Brain Floating Point

- We'll use the number 6.5
- The decimal number 6.5 can be written as 1.625*4 in binary.

$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent}-127)}$$

- Sign is 0
- (1 +mantissa) is 1.625, mantissa is 0.625
- 0.625 = **1010000**
- (Exponent -127) = 2, Exponent =129
- 129 = 10000001

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

Comparison of FP32, FP16, & BFloat16

Ranges

FP32 (32-bit Floating Point):

-3.4×10^{38} to 3.4×10^{38}

FP16 (16-bit Floating Point):

-6.55×10^{4} to 6.55×10^{4}

BFloat16 (16-bit Brain Floating Point):

-3.39×10^{38} to 3.39×10^{38}

Comparison of FP32, FP16, & BFloat16

Format	Bits	Exponent	Mantissa	Precision	Range	Usage
FP32	32 bits	8 bits	23 bits	High	Large range	Training, precise tasks
FP16	16 bits	5 bits	10 bits	Lower	Smaller range	Inference, efficiency
BFloat16	16 bits	8 bits	7 bits	Moderate	Same as FP32	Large-scale training

Other floating point formats

Other formats

- One of the latest innovations in this area is FP8 (8-bit floating-point), introduced by NVIDIA

- **E4M3**:

  4 bits for the exponent and 3 bits for the mantissa.

- **E5M2**:

  5 bits for the exponent and 2 bits for the mantissa.

Other floating point formats

FP8
NVIDIA

- One of the latest innovations in this area is FP8 (8-bit floating-point), introduced by NVIDIA

- **E4M3**:

  4 bits for the exponent and 3 bits for the mantissa.

- **E5M2**:

  5 bits for the exponent and 2 bits for the mantissa.

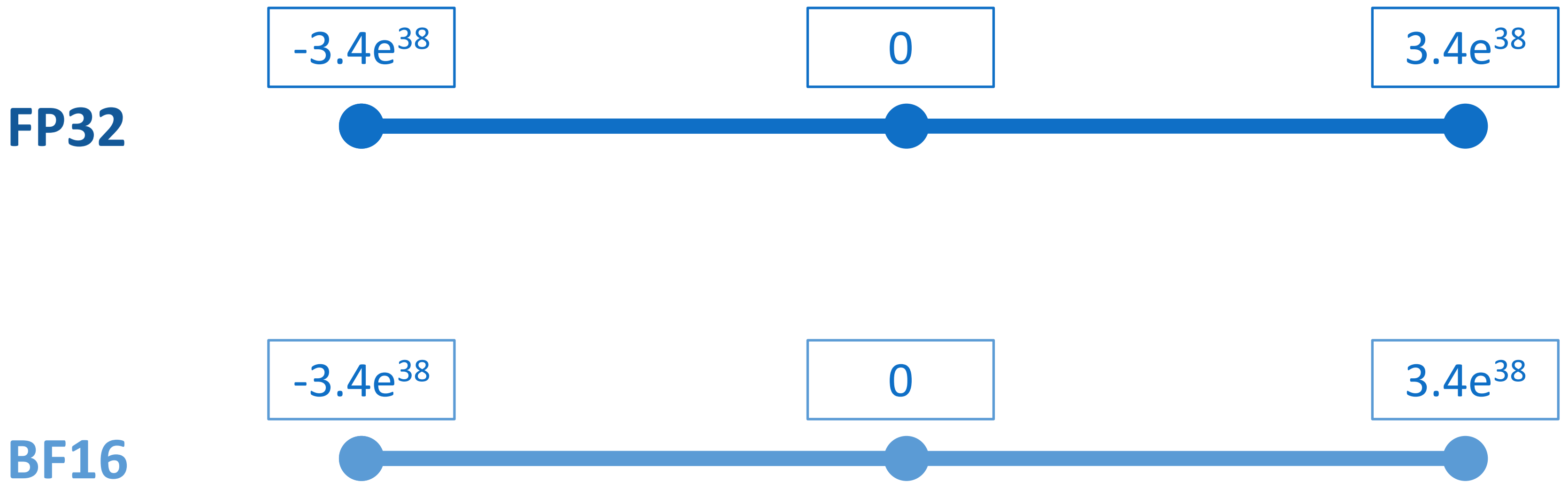
Other floating point formats

INT4
FP4

- **INT4** uses just 4 bits to represent integer values.
- **FP4** is an experimental format using just 4 bits to represent floating-point numbers

Quantization - Downcasting

FP32 to BF16



Same dynamic range

Quantization - Downcasting

FP32 to BF16

FP32

$-3.4e^{38}$ 0 $3.4e^{38}$

(In billions) 70*32 bits = 70*4 Bytes = 280 GB (Gigabytes)

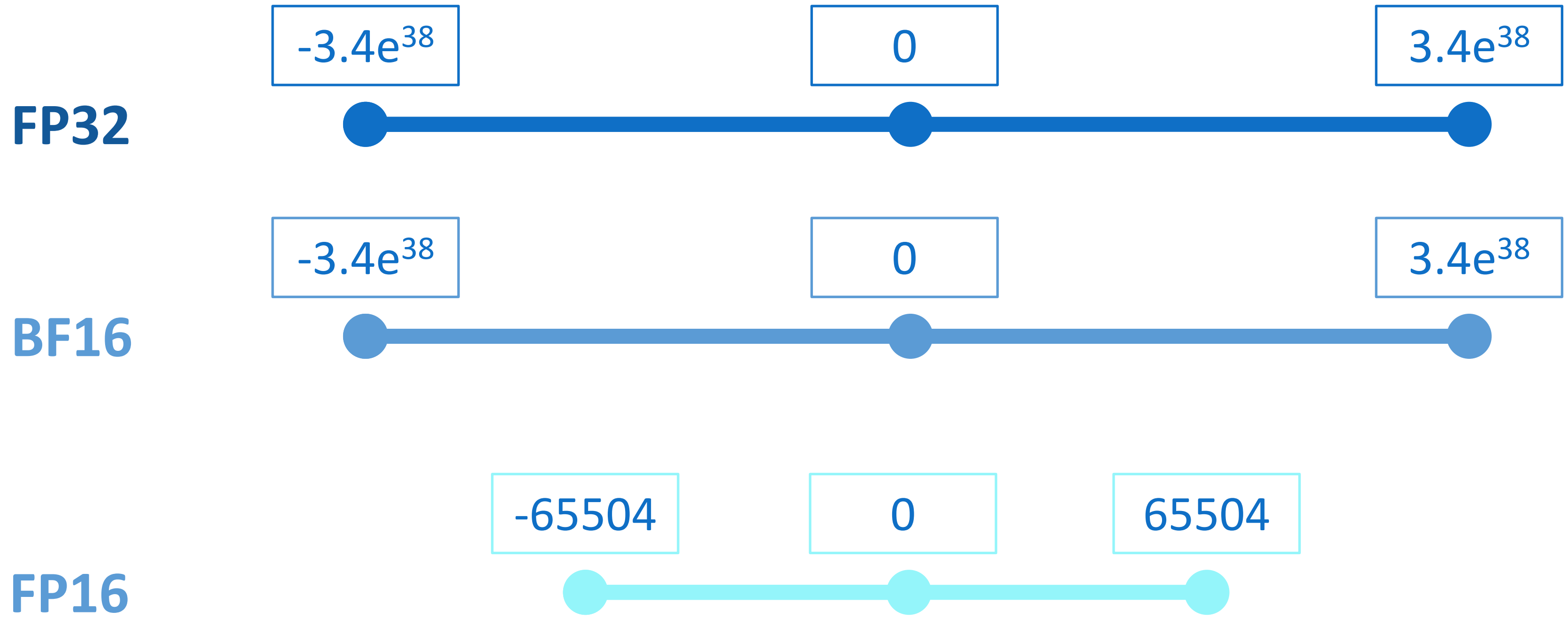
BF16

$-3.4e^{38}$ 0 $3.4e^{38}$

(In billions) 70*16 bits = 70*2 Bytes = 140 GB (Gigabytes)

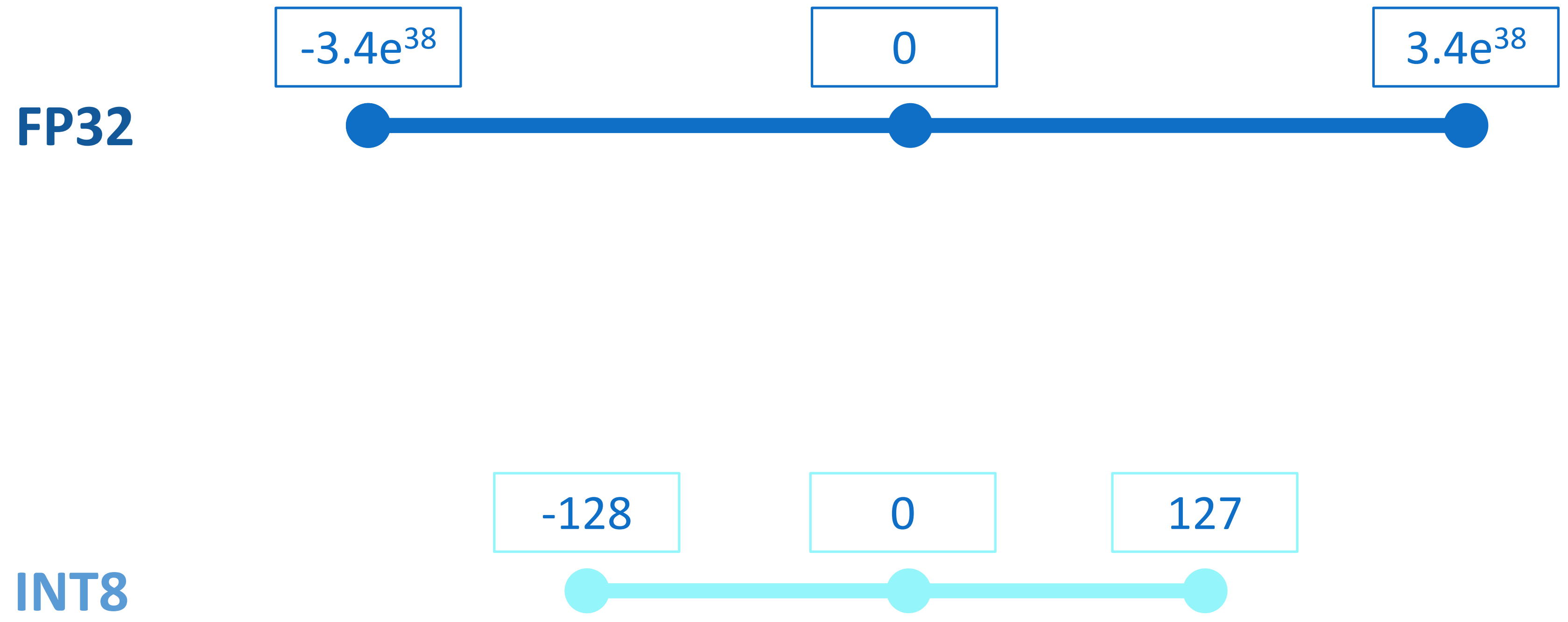
Quantization - Downcasting

Problem with FP16



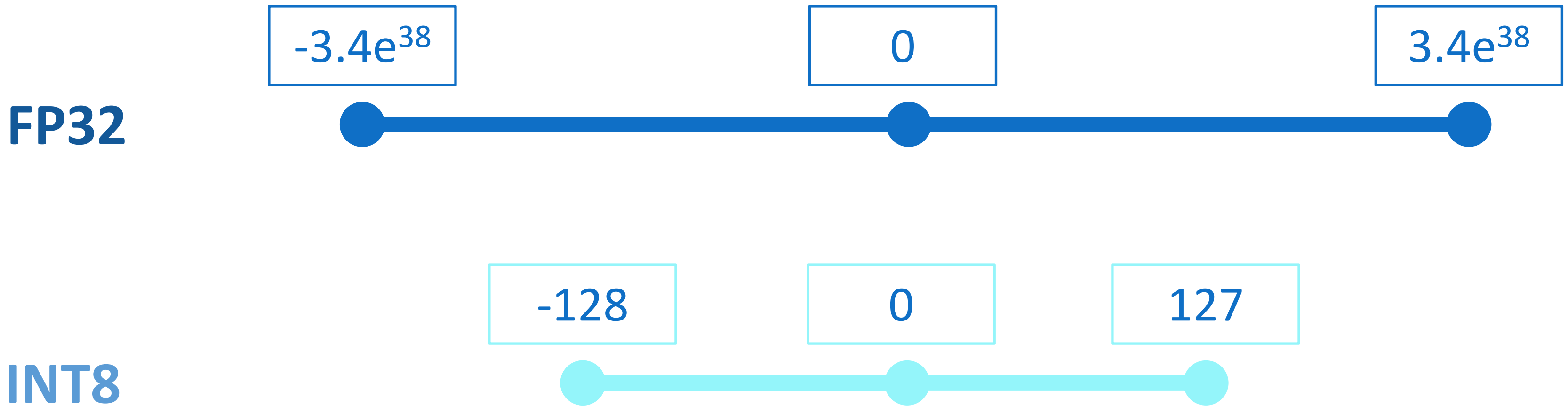
Quantization - FP32 to INT8

FP32 to INT8



Quantization - FP32 to INT8

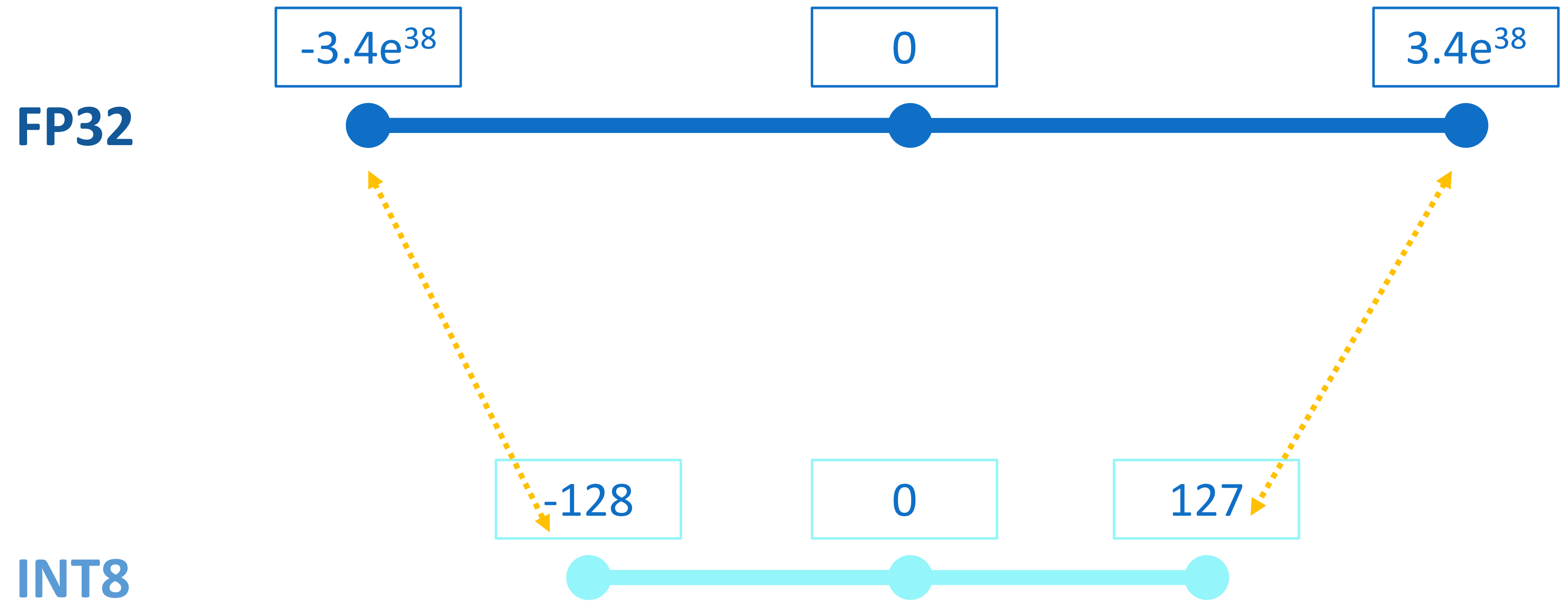
Advantages



- **Reduced** memory footprint
- **Faster** calculations

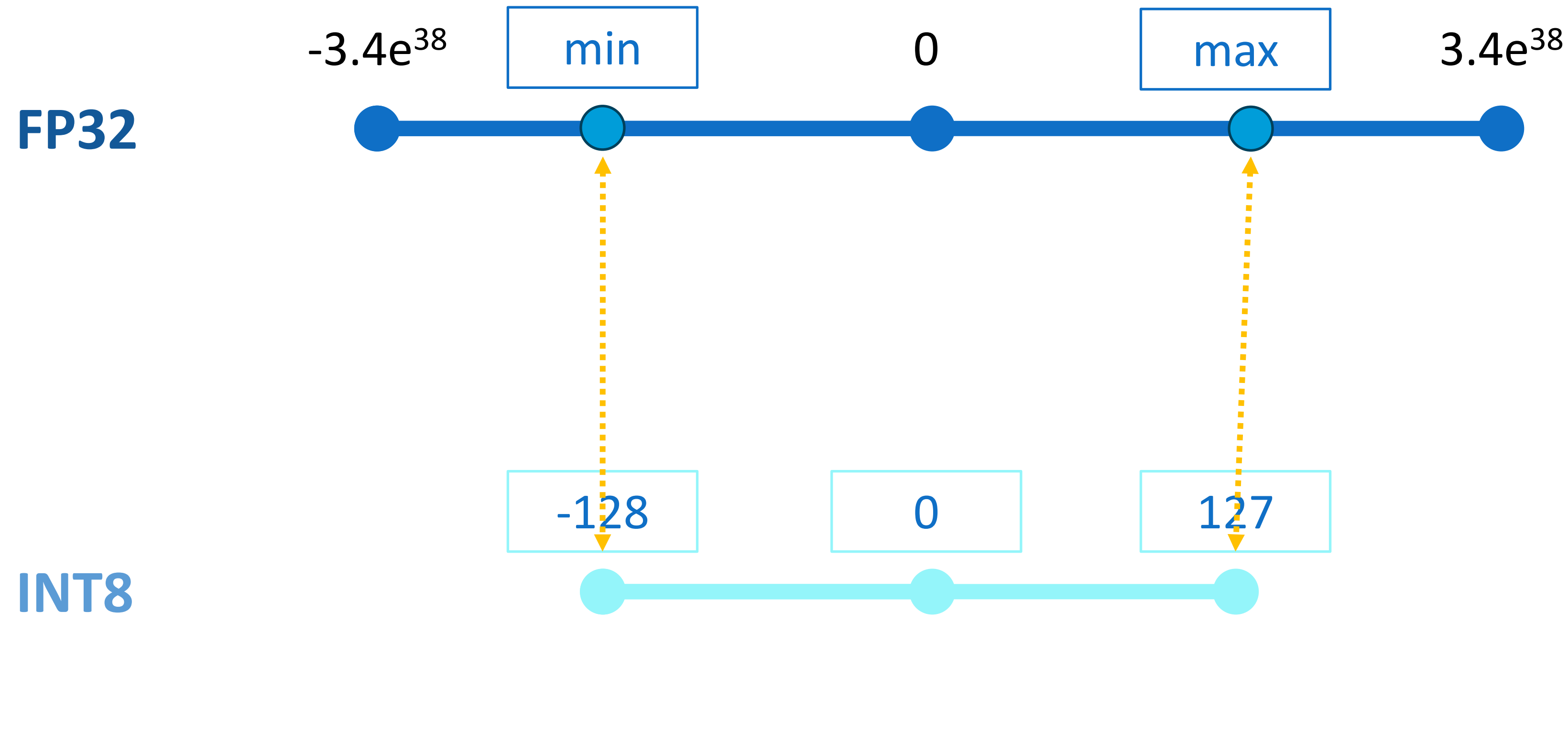
Quantization - FP32 to INT8

Key Concepts



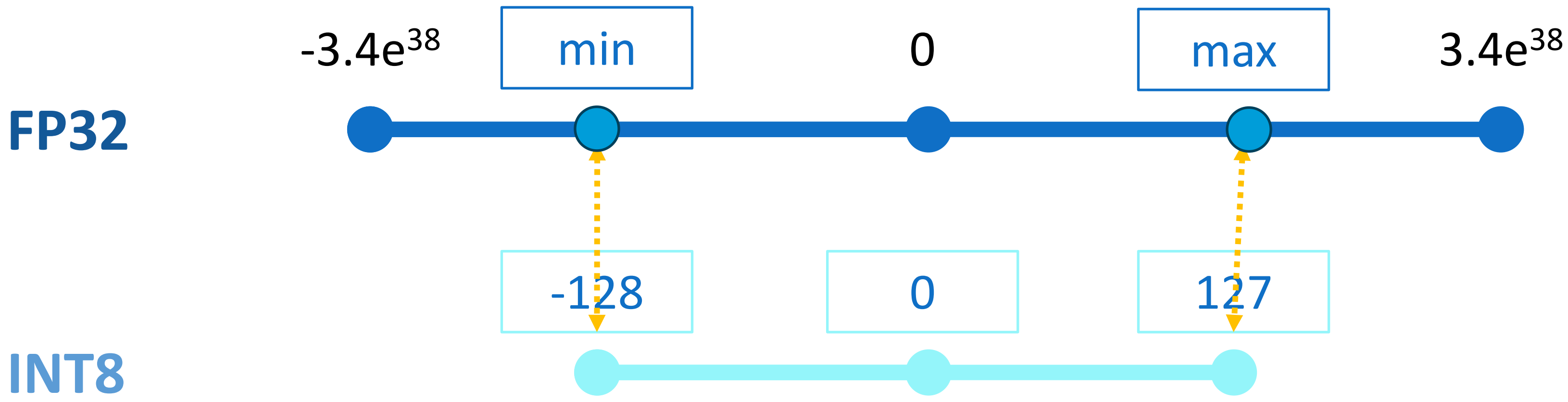
Quantization - FP32 to INT8

Key Concepts



Quantization - FP32 to INT8

Types



1. **Symmetric Quantization**
2. **Asymmetric Quantization**

Quantization - FP32 to INT8

Symmetric Quantization

$$\text{Scaling Factor} = \frac{X}{127}$$

$$\text{INT8 Value} = \frac{\text{FP32 Value}}{\text{Scaling Factor}}$$

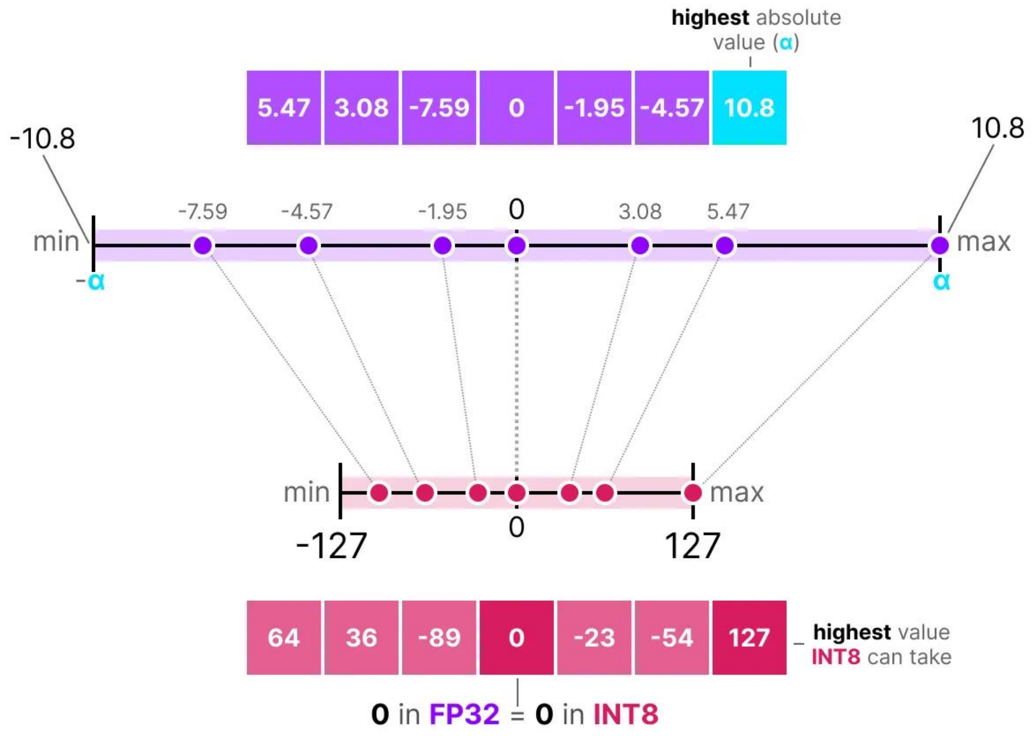
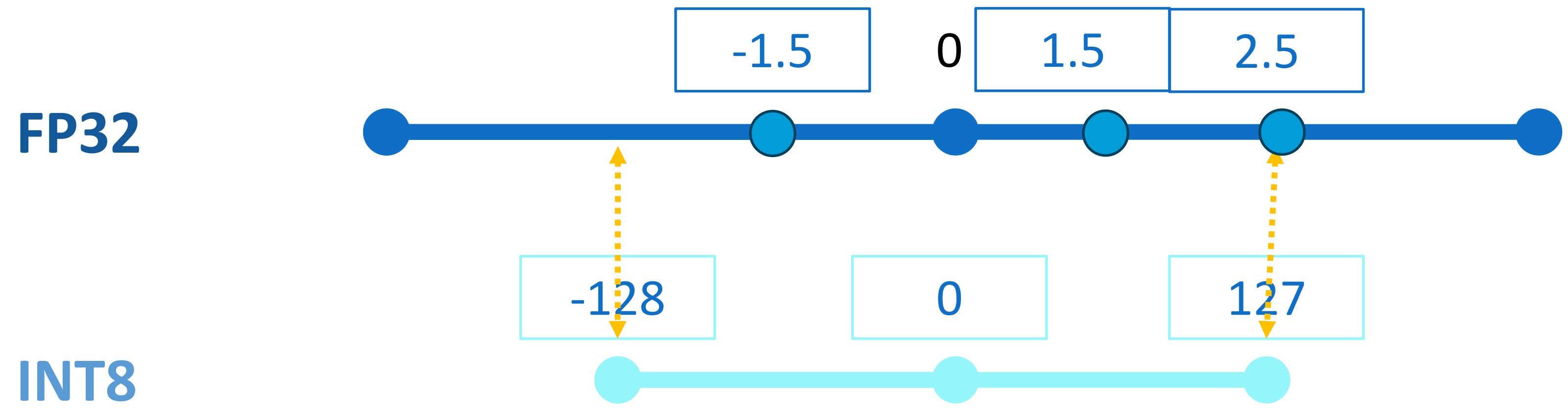


Image source : https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantization - FP32 to INT8

Symmetric Quantization

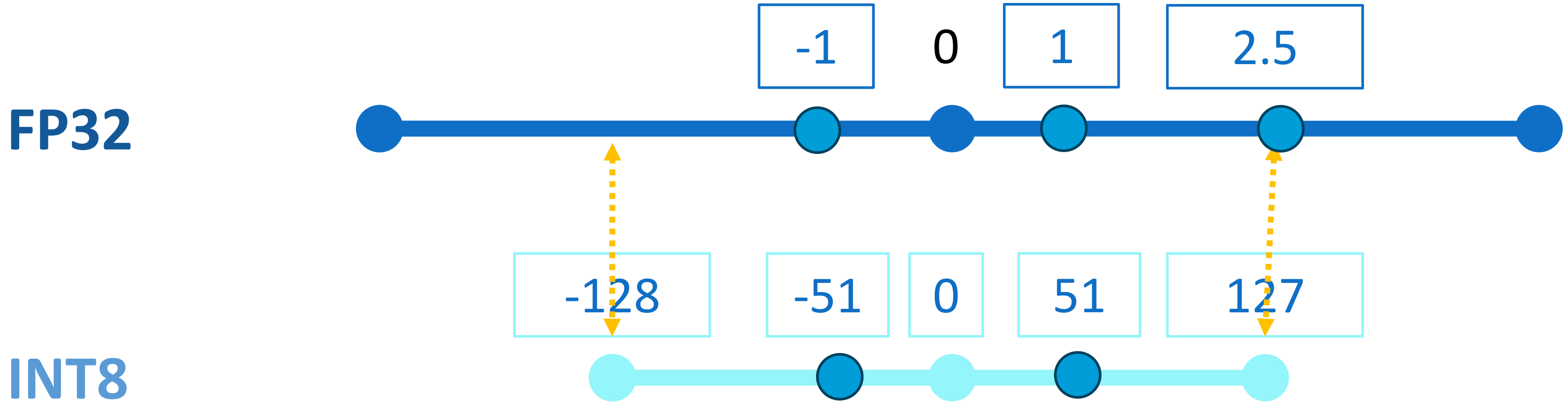


$$\text{Scaling Factor} = \frac{X}{127}$$

$$\text{INT8 Value} = \frac{\text{FP32 Value}}{\text{Scaling Factor}}$$

Quantization - FP32 to INT8

Symmetric Quantization

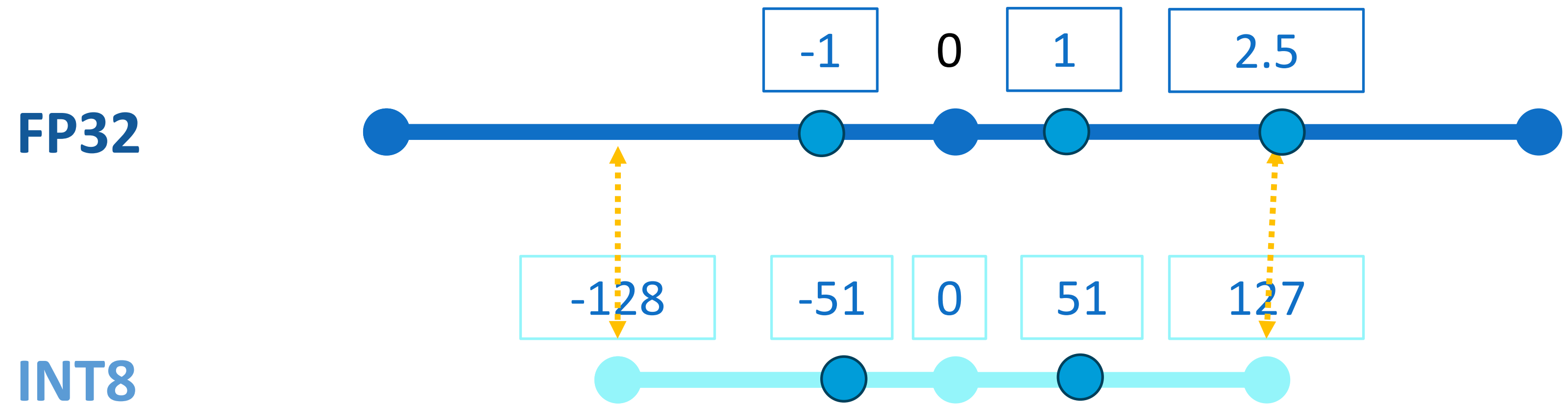


$$\text{Scaling Factor} = \frac{2.5}{127} \approx 0.0197$$

$$\text{INT8 Value} = \frac{1.0}{0.0197} \approx 51$$

Quantization - FP32 to INT8

Symmetric Quantization



$$\text{Scaling Factor} = \frac{2.5}{127} \approx 0.0197$$

$$\text{INT8 Value} = \frac{1.0}{0.0197} \approx 51$$

51× 0.0197 = 1.0047

Quantization - FP32 to INT8

Symmetric Quantization Example

$$\text{FP32 Matrix} = \begin{bmatrix} 1.25 & 2.67 & 3.08 \\ 4.12 & 3.02 & 2.01 \\ 0.89 & 5.45 & 3.76 \end{bmatrix}$$

Quantization - FP32 to INT8

Symmetric Quantization Example

$$\text{FP32 Matrix} = \begin{bmatrix} 1.25 & 2.67 & 3.08 \\ 4.12 & 3.02 & 2.01 \\ 0.89 & 5.45 & 3.76 \end{bmatrix}$$

$$\text{Scaling Factor} = \frac{5.45}{127} \approx 0.0429$$

Quantization - FP32 to INT8

Symmetric Quantization Example

$$\text{FP32 Matrix} = \begin{bmatrix} 1.25 & 2.67 & 3.08 \\ 4.12 & 3.02 & 2.01 \\ 0.89 & 5.45 & 3.76 \end{bmatrix}$$

$$\text{Scaling Factor} = \frac{5.45}{127} \approx 0.0429$$

$$\text{Quantized INT8 Matrix} = \begin{bmatrix} 29 & 62 & 72 \\ 96 & 70 & 47 \\ 21 & 127 & 88 \end{bmatrix}$$

Quantization - FP32 to INT8

Symmetric Quantization Example

$$\text{Quantized INT8 Matrix} = \begin{bmatrix} 29 & 62 & 72 \\ 96 & 70 & 47 \\ 21 & 127 & 88 \end{bmatrix}$$

$$\text{Dequantized FP32 Matrix (Symmetric)} =$$

$$\begin{bmatrix} 29 \times 0.0429 & 62 \times 0.0429 & 72 \times 0.0429 \\ 96 \times 0.0429 & 70 \times 0.0429 & 47 \times 0.0429 \\ 21 \times 0.0429 & 127 \times 0.0429 & 88 \times 0.0429 \end{bmatrix}$$

$$\approx \begin{bmatrix} 1.24 & 2.66 & 3.09 \\ 4.12 & 3.00 & 2.02 \\ 0.90 & 5.45 & 3.77 \end{bmatrix}$$

Quantization - FP32 to INT8

Symmetric Quantization Example

$$\text{Quantization Error} = \text{Original FP32} - \text{Dequantized FP32}$$

$$\text{Quantization Error (Symmetric)} =$$

$$\begin{bmatrix} 1.25 - 1.24 & 2.67 - 2.66 & 3.08 - 3.09 \\ 4.12 - 4.12 & 3.02 - 3.00 & 2.01 - 2.02 \\ 0.89 - 0.90 & 5.45 - 5.45 & 3.76 - 3.77 \end{bmatrix}$$

$$\approx \begin{bmatrix} 0.01 & 0.01 & -0.01 \\ 0 & 0.02 & -0.01 \\ -0.01 & 0 & -0.01 \end{bmatrix}$$

Quantization - FP32 to INT8

Asymmetric Quantization

$$\text{Scaling Factor} = \frac{B - A}{127 - (-128)} = \frac{B - A}{255}$$

$$\text{Zero Point} = \text{Round}\left(q_{\text{min}} - \frac{r_{\text{min}}}{\text{Scale}}\right)$$

$$\text{Quantized Value} = \text{Round}\left(\frac{\text{FP32 Value}}{\text{Scale}} + \text{Zero Point}\right)$$

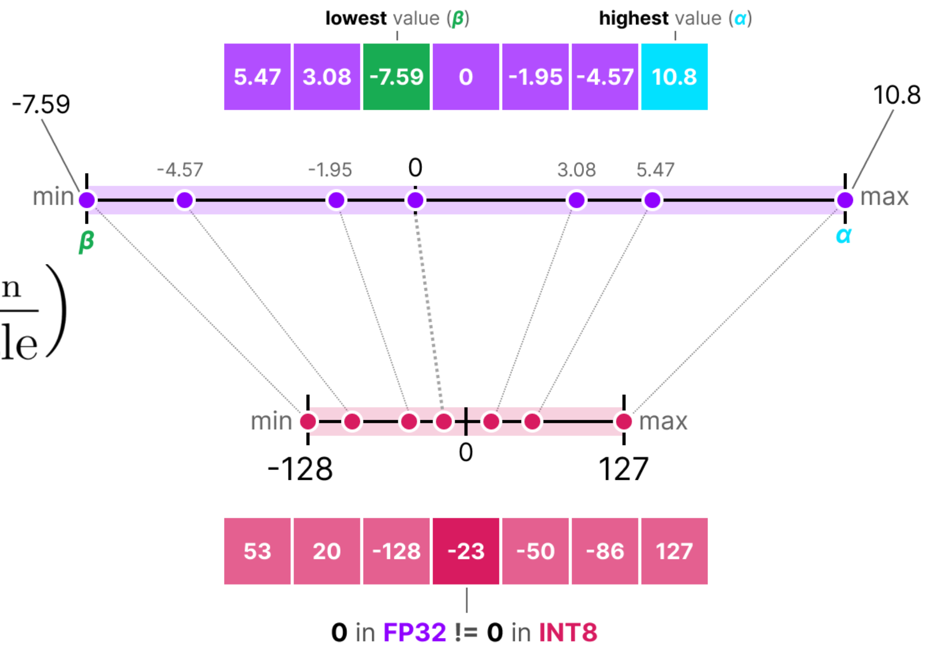


Image source : https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantization - FP32 to INT8

Asymmetric Quantization

$$\text{FP32 Matrix} = \begin{bmatrix} 1.25 & 2.67 & 3.08 \\ 4.12 & 3.02 & 2.01 \\ 0.89 & 5.45 & 3.76 \end{bmatrix}$$

$$\text{Scaling Factor} = \frac{B - A}{127 - (-128)} = \frac{B - A}{255}$$

$$\text{Scale} = \frac{5.45 - 0.89}{127 - (-128)} = \frac{4.56}{255} \approx 0.01788$$

Image source : https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantization - FP32 to INT8

Asymmetric Quantization

$$\text{FP32 Matrix} = \begin{bmatrix} 1.25 & 2.67 & 3.08 \\ 4.12 & 3.02 & 2.01 \\ 0.89 & 5.45 & 3.76 \end{bmatrix}$$

$$\text{Zero Point} = \text{Round}\left(q_{\min} - \frac{r_{\min}}{\text{Scale}}\right)$$

$$\text{Zero Point} = \text{Round}\left(-128 - \frac{0.89}{0.01788}\right)$$

Zero point = **-178**.

Quantization - FP32 to INT8

Asymmetric Quantization

$$\text{FP32 Matrix} = \begin{bmatrix} 1.25 & 2.67 & 3.08 \\ 4.12 & 3.02 & 2.01 \\ 0.89 & 5.45 & 3.76 \end{bmatrix}$$

$$\text{Quantized Value} = \text{Round}\left(\frac{\text{FP32 Value}}{\text{Scale}} + \text{Zero Point}\right)$$

Quantized INT8 Matrix:

$$\begin{bmatrix} -108 & -29 & -6 \\ 52 & -9 & -66 \\ -128 & 127 & 32 \end{bmatrix}$$

Quantization - FP32 to INT8

Asymmetric Quantization

Quantized INT8 Matrix: $\begin{bmatrix} -108 & -29 & -6 \\ 52 & -9 & -66 \\ -128 & 127 & 32 \end{bmatrix}$

Dequantized FP32 value: $= (\text{INT8 Value} - \text{Zero Point}) \times \text{Scale}$

Dequantized FP32 Matrix: $\begin{bmatrix} 1.2516 & 2.6641 & 3.0754 \\ 4.1124 & 3.0217 & 2.0026 \\ 0.8940 & 5.4534 & 3.7548 \end{bmatrix}$

Quantization Error: $\begin{bmatrix} -0.0016 & 0.0059 & 0.0046 \\ 0.0076 & -0.0017 & 0.0074 \\ -0.0040 & -0.0034 & 0.0052 \end{bmatrix}$

Quantization - FP32 to INT8

Asymmetric Quantization

Quantization Error (Symmetric):

$$\approx \begin{bmatrix} 0.01 & 0.01 & -0.01 \\ 0 & 0.02 & -0.01 \\ -0.01 & 0 & -0.01 \end{bmatrix}$$

Quantization Error Asymmetric:

$$\begin{bmatrix} -0.0016 & 0.0059 & 0.0046 \\ 0.0076 & -0.0017 & 0.0074 \\ -0.0040 & -0.0034 & 0.0052 \end{bmatrix}$$

Quantization - FP32 to INT8

Differences

Symmetric Quantization:

- **Advantages**: Simple to implement, fast, and efficient. The scaling factor is uniform, making computations straightforward.
- **Limitations**: The quantization error can be larger, especially for skewed data, as the range is centered around zero.

Asymmetric Quantization:

- **Advantages**: Provides better accuracy and smaller quantization error, especially for data that is not symmetric or skewed toward a particular range. It uses flexible scaling factors and zero points to improve representation.
- **Limitations**: Slightly more complex to compute but provides higher precision.