Pruna AI

Quantization for Language Models

Bertrand Charpentier
Founder, President & Chief Scientist

Overview

- Data types: Int, float
- What is Quantization?
- Weight (Static) vs Activation/KV (Dynamic) Quantization
- Linear Quantization
    - Symetric vs Asymetric Quantization
- Codebook Quantization
    - Scalar vs vector Quantization
- Tensor vs Channel vs Group Quantization
- Training Aware vs Post training quantization
- Outlier Values in Quantization
- Iterative Quantization
- Quantization With/Without Data
- State-of-the-art Quantization Methods

Data Types - Integer

They represent positive and negative natural numbers.

Unsigned Integer

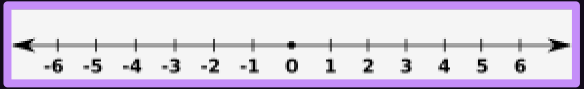
- **n-bit range:** $[0, 2^n - 1]$

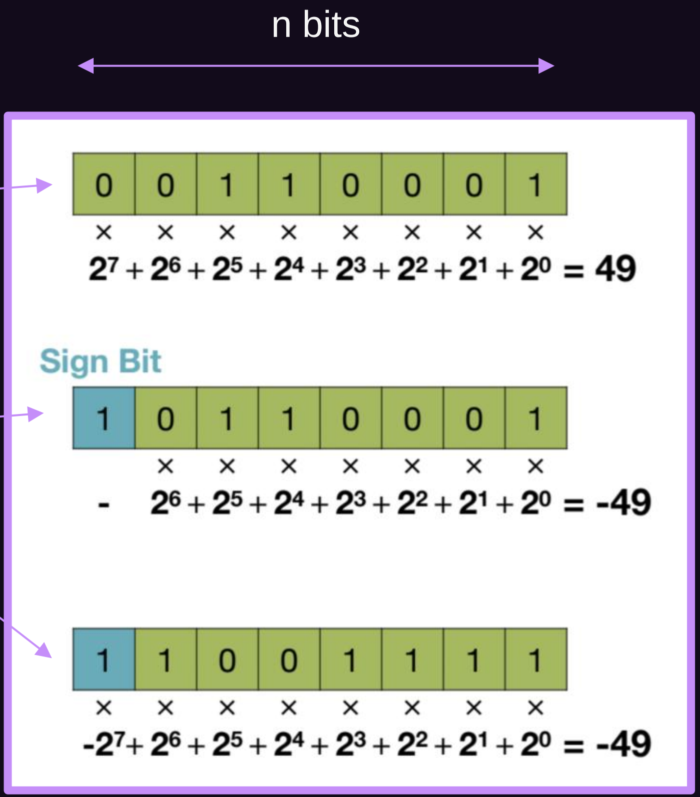
Signed Integer

- **n-bit range:** $[-2^{n-1} + 1, 2^{n-1} - 1]$ (Sign magnitude representation)
- **n-bit range:** $[-2^{n-1}, 2^{n-1} - 1]$ (Two's complement representation)

Remarks

- Integer are regularly space on the number line.





Reference:
- https://hanlab.mit.edu/courses/2024-fall-65940

Data Types – Fixed/Floating Point

They represent positive and negative real numbers.

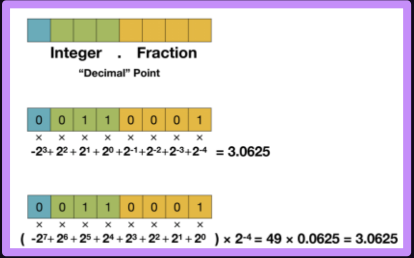
Fixed point

- **n-bit with k-bit fraction range:** $\left[-(2^{n-1})2^{-k}, (2^{n-1}-1)2^{-k}\right]$

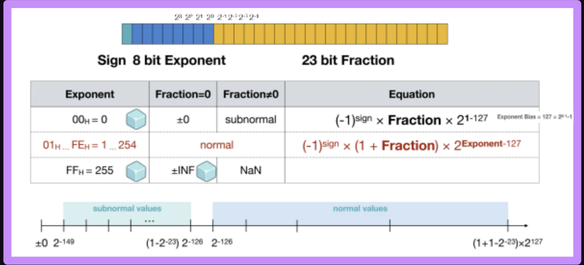
Floating point

- Normal numbers represent large real numbers
- Subnormal numbers represent small real numbers
- +/-Inf, NaN can be represented as well

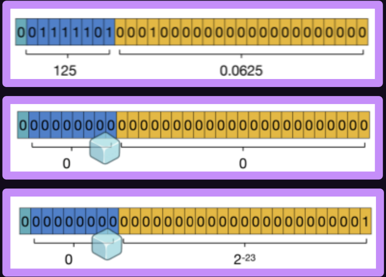
n bits



n bits

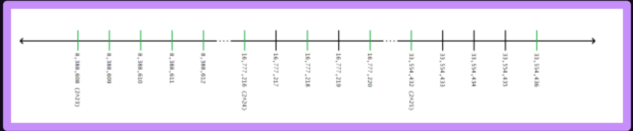


Exponent	Fraction=0	Fraction≠0	Equation
$00_H = 0$	±0	subnormal	$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$
$01_H \ldots FE_H = 1 \ldots 254$	normal		$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\text{Exponent}-127}$
$FF_H = 255$	±INF	NaN	



Remarks

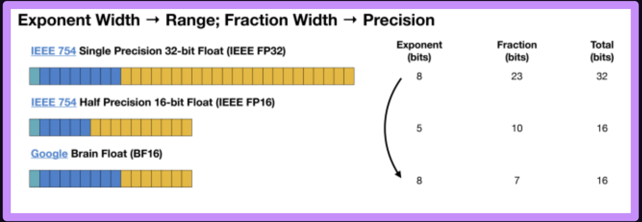
- Fixed points are regularly spaced
- Floating points are not regularly spaced.



Reference:

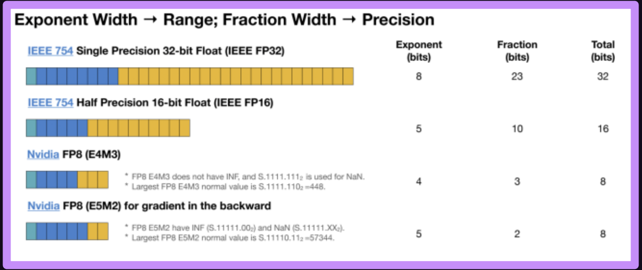
- https://hanlab.mit.edu/courses/2024-fall-65940

Data Types – Examples



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

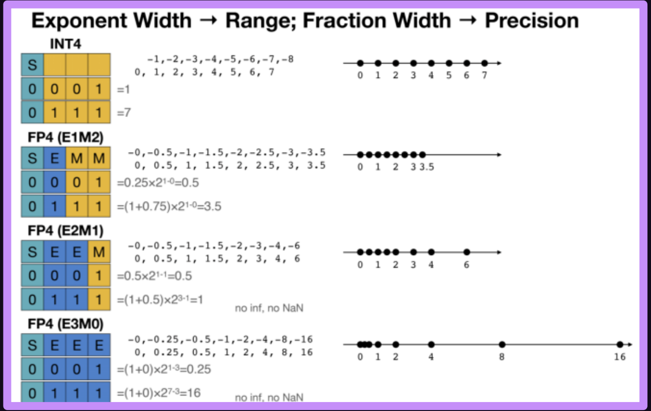


Exponent Width → Range; Fraction Width → Precision

	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
Nvidia FP8 (E4M3)	4	3	8
Nvidia FP8 (E5M2) for gradient in the backward	5	2	8

* 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.

* 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 Width → Range; Fraction Width → Precision

INT4

S			
0	0	0	1
0	1	1	1

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

=1

=7

FP4 (E1M2)

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

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

$=0.25 \times 2^{1-0} = 0.5$

$=(1+0.75) \times 2^{1-0} = 3.5$

FP4 (E2M1)

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

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

$=0.5 \times 2^{1-1} = 0.5$

$=(1+0.5) \times 2^{3-1} = 1$

no inf, no NaN

FP4 (E3M0)

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

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

$=(1+0) \times 2^{1-3} = 0.25$

$=(1+0) \times 2^{7-3} = 16$

no inf, no NaN

Remarks

- Exponent width specifies the max/min range.
- Fraction width specifies the space precision between two subsequent numbers.
- Training usually benefit from higher range.

Reference:

- https://hanlab.mit.edu/courses/2024-fall-65940

What is Quantization?

Quantization the process of mapping continuous infinite/large set of values to a smaller set of discrete finite values.

Input - A variable W (e.g. weights, activations) taking values in a continuous/large set of values.

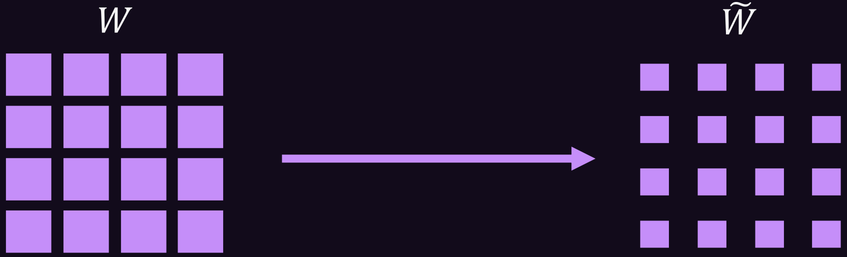
Output - A variable $\widetilde{W}$ taking values in a small set of discrete values where the quantization error $\|W - \widetilde{W}\|$ is small.

Algorithm

- Many options!

Benefits

- It can reduce memory movement.
- It can reduce compute load.



Reference:

- https://hanlab.mit.edu/courses/2024-fall-65940

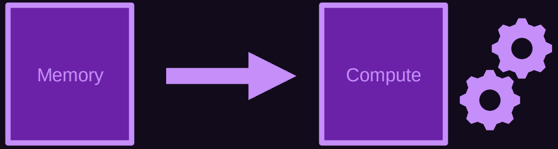
Weight and Activation Quantization

Without Quantization

Step 0: Store the high precision weight/activations on the memory.

Step 1: Loads in high precision the weights/activation values from memory to compute processor units.

Step 2: Performs the operations between the weights and activations in high precision.

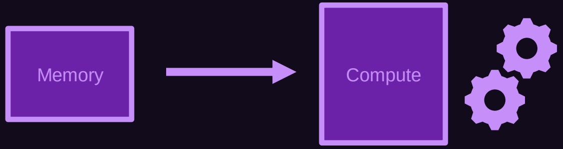


With Weight Quantization

Step 0: Store the low precision weight and high precision activations on the memory. **[Save Memory Space]**

Step 1: Loads in low precision the weights and in high precision the activation values from memory to compute processor units. **[Save Memory Time]**

Step 2: Performs the operations between the weights and activations in high precision (after unpacking & dequantizing the weights).

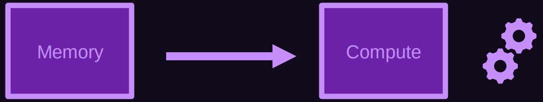


With Activation Quantization

Step 0: Store the low precision weight/activations on the memory. **[Save Memory Space]**

Step 1: Loads in low precision the weights/activation values from memory to compute processor units. **[Save Memory Time]**

Step 2: Performs the operations between the weights and activations in low precision. **[Save Compute Time]**



Remarks

- With weight/activation quantization, Step 1 requires the hardware to support the low bit precisions load. Otherwise, **custom kernels would be required to unpack the low bit precision values encoded into high bit precision** that the hardware supports.
- With activation quantization, Step2 requires the hardware to support the low bit precision computations. Otherwise, **custom kernels would be required to compute in low bit precision** directly.
- It is also possible to quantize the KV cache!

Reference:

- https://hanlab.mit.edu/courses/2024-fall-65940

Linear Quantization

Linear Quantization: it is an affine map from integers to real numbers defined by two parameters

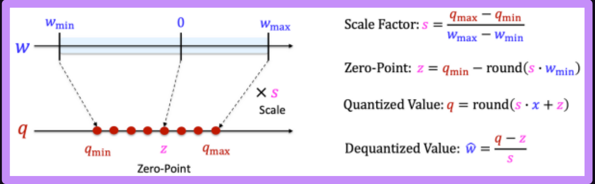
- The floating-point scale $S = \frac{q_{max} - q_{min}}{w_{max} - w_{min}}$
- The integer zero-point $Z = q_{min} - \lfloor S w_{min} \rceil$

Quantization → $q = Quant(w) = \lfloor Sw + Z \rceil$

Dequantizaton → $\tilde{w} = Dequant(q) = \frac{q - Z}{S}$

Remarks

- The formula of S and Z are the consequence of enforcing $w_{min} = Dequant(q_{min})$ and $w_{max} = Dequant(q_{max})$
- The quantized value $q = Z$ is mapped to the dequantized value $\tilde{w} = 0$.
- With weight/activation quantization, we can perform the matrix multiplication $Y = WX$ efficiently.
- For activation quantization, parameters are learned during/after training on data batches.

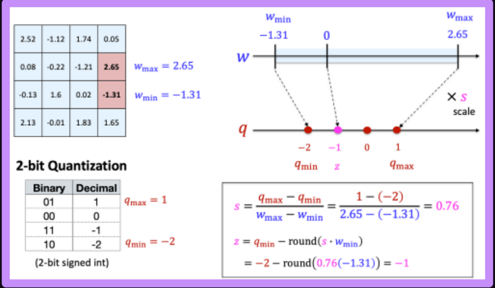


Scale Factor: $s = \frac{q_{\max} - q_{\min}}{w_{\max} - w_{\min}}$

Zero-Point: $z = q_{\min} - \text{round}(s \cdot w_{\min})$

Quantized Value: $q = \text{round}(s \cdot x + z)$

Dequantized Value: $\hat{w} = \frac{q - z}{s}$



2.52	-1.12	1.74	0.05
0.08	-0.22	-1.21	2.65
-0.13	1.6	0.02	-1.31
2.13	-0.01	1.83	1.65

$w_{\max} = 2.65$

$w_{\min} = -1.31$

2-bit Quantization

Binary	Decimal	
01	1	$q_{\max} = 1$
00	0	
11	-1	
10	-2	$q_{\min} = -2$

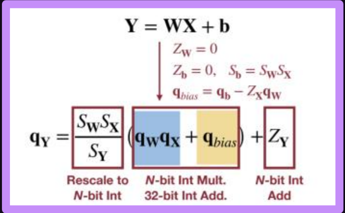
(2-bit signed int)

$$s = \frac{q_{\max} - q_{\min}}{w_{\max} - w_{\min}} = \frac{1 - (-2)}{2.65 - (-1.31)} = 0.76$$

$$z = q_{\min} - \text{round}(s \cdot w_{\min})$$
$$= -2 - \text{round}(0.76(-1.31)) = -1$$

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$S_{\mathbf{Y}}(\mathbf{q_Y} - Z_{\mathbf{Y}}) = S_{\mathbf{W}}(\mathbf{q_W} - Z_{\mathbf{W}}) \cdot S_{\mathbf{X}}(\mathbf{q_X} - Z_{\mathbf{X}}) + S_{\mathbf{b}}(\mathbf{q_b} - Z_{\mathbf{b}})$$



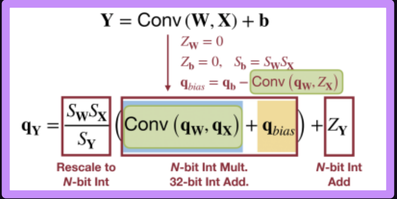
$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$Z_{\mathbf{W}} = 0$

$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$

$\mathbf{q}_{bias} = \mathbf{q_b} - Z_{\mathbf{X}} \mathbf{q_W}$

$$\mathbf{q_Y} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}} (\mathbf{q_W}\mathbf{q_X} + \mathbf{q}_{bias}) + Z_{\mathbf{Y}}$$



$$\mathbf{Y} = \text{Conv}(\mathbf{W}, \mathbf{X}) + \mathbf{b}$$

$Z_{\mathbf{W}} = 0$

$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$

$\mathbf{q}_{bias} = \mathbf{q_b} - \text{Conv}(\mathbf{q_W}, Z_{\mathbf{X}})$

$$\mathbf{q_Y} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}} (\text{Conv}(\mathbf{q_W}, \mathbf{q_X}) + \mathbf{q}_{bias}) + Z_{\mathbf{Y}}$$

Reference:
- https://medium.com/@lmpo/understanding-model-quantization-for-llms-1573490d44ad
- https://hanlab.mit.edu/courses/2024-fall-65940

Asymmetric and Symmetric Quantization

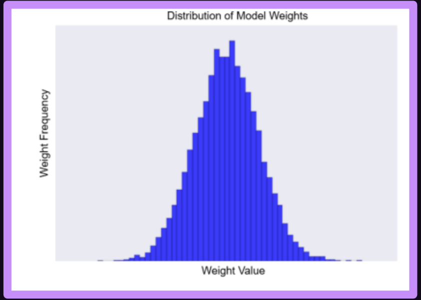
Observation: Weight distributions are usually Gaussian centered around 0.

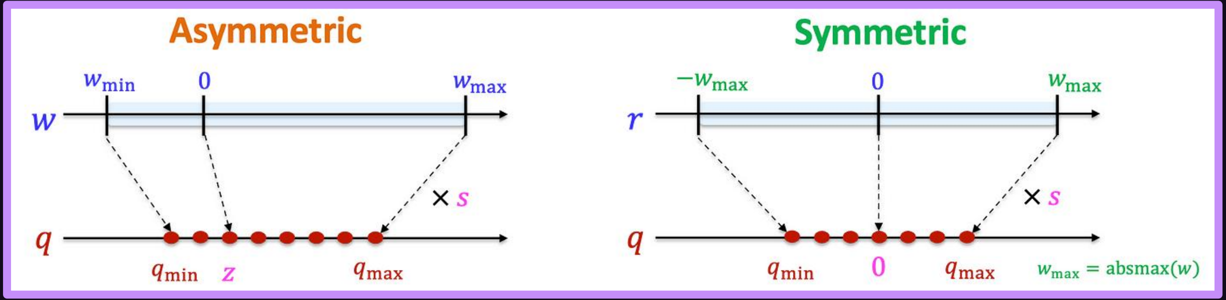
Asymmetric quantization: Assuming $abs(w_{min}) \neq w_{max}$ and $Z \neq 0$ gives asymmetric quantization with $S = \frac{q_{max}-q_{min}}{w_{max}-w_{min}}$ and the $Z = q_{min} - \lfloor Sw_{min} \rceil$

Symmetric quantization: Assuming $abs(w_{min}) = w_{max}$ and set $Z = 0$ gives symmetric quantization with $S = \frac{q_{max}}{w_{max}}$ and $Z = 0$.

Remark

- Symmetric quantization allow to not save the zero-point parameter.





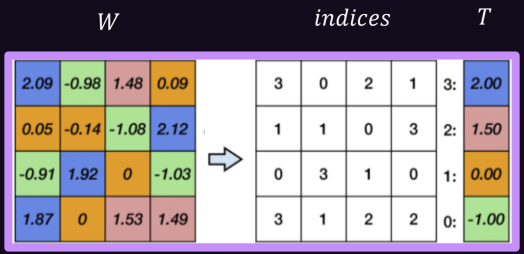
Reference:
- https://medium.com/@lmpo/understanding-model-quantization-for-llms-1573490d44ad
- https://hanlab.mit.edu/courses/2024-fall-65940

Codebook/Lookup Table Quantization

Codebook/Lookup Table Quantization: it is a codebook/lookup table map T from discrete indices to a small set of real numbers.

Quantization → $q = Quant(w) = argmin_q\ (\|T(q) - w\|)$

Dequantizaton → $\tilde{w} = Dequant(q) = T(q)$



W

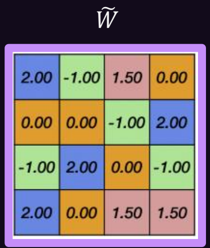
2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$indices$

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

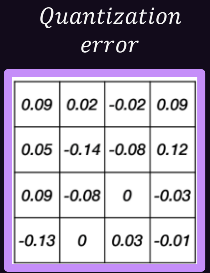
T

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



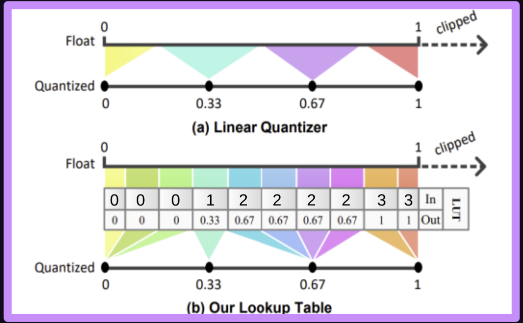
$\tilde{W}$

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



Remarks

- Codebook/lookup tables can be learned with k-means (Deep Compression), simply with gradient descent (AQLM), or with precomputed (optimal) meshes (Higgs).
- After quantization, we need memory for both the integer indices and the floating-point codes.

Reference:
- https://openaccess.thecvf.com/content/CVPR2022/papers/Wang_Learnable_Lookup_Table_for_Neural_Network_Quantization_CVPR_2022_paper.pdf
- https://hanlab.mit.edu/courses/2024-fall-65940

Scalar and Vector Quantization

Without Quantization: Each weight is a floating-point scalar.

- The required memory for a 32-bit matrix $M \in \mathbb{R}^{m \times m}$ is $32 \times m^2$.

With Scalar Quantization: Each single weight is attached an integer index pointing to a floating-point scalar.

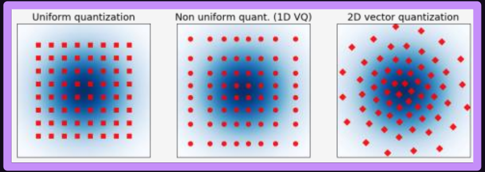
- The required memory for a n-bit index matrix $I \in \mathbb{R}^{m \times m}$ and the scalar codebook/lookup table is $m^2 \times n + 2^n \times 32$.

With Vector Quantization: Each group of d weights is attached an integer index pointing to a floating-point vector.

- The required memory for a n-bit index matrix $I \in \mathbb{R}^{m \times m}$ and the vector codebook/lookup table is $\frac{m^2}{d} \times n + 2^n \times d \times 32$.

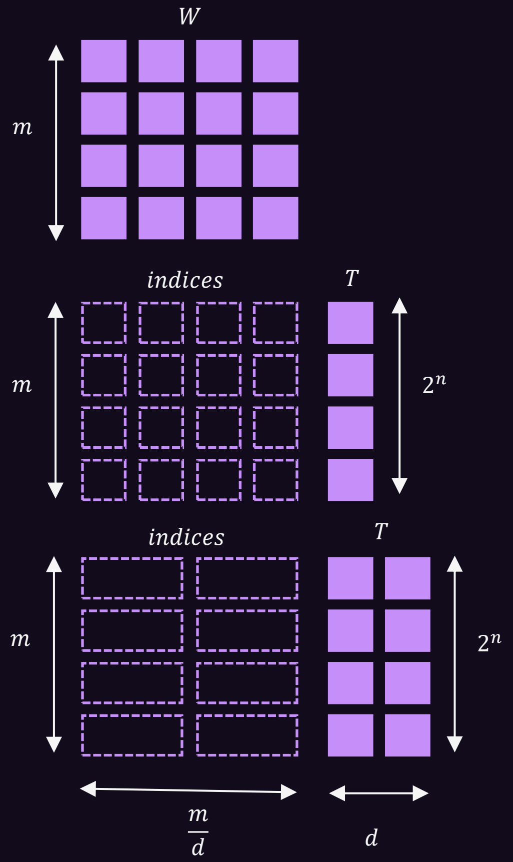
Remarks

- The average number of bit per weights is higher than the chosen bit width n.
- Vector quantization can capture more complex distribution patterns.



Without Quantization	With Scalar Quantization	With Vector Quantization
32	$n + \frac{2^n \times 32}{m^2}$	$n + \frac{2^n \times d \times 32}{m^2}$

Average number of bit per weights

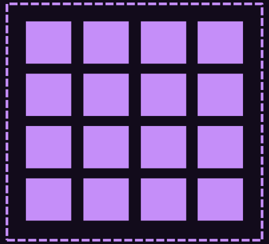


Reference:

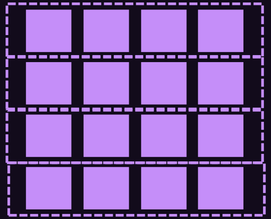
- https://arxiv.org/html/2402.15319v1
-

Tensor, Channel, Group Quantization

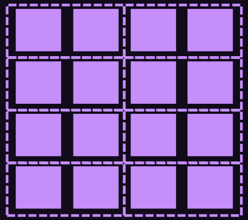
Tensor quantization: The same quantization/dequantization is used for each tensor.



Channel quantization: The same quantization/dequantization is used for each channel.



Group quantization: The same quantization/dequantization is used for each group.



Remarks

- For linear quantization, it means that we have one scale/zero point per tensor/channel/group.
- For codebook/lookup table quantization, it means that we have a different codebook per per tensor/channel/group.
- The quantization parameters are **more accurate** since they are computed on smaller structures.
- The quantization parameters require **more memory** since there is one set of parameter per structure.

Reference:
- https://medium.com/@lmpo/understanding-model-quantization-for-llms-1573490d44ad
- https://hanlab.mit.edu/courses/2024-fall-65940

Post-Training vs Quantization-Aware Training

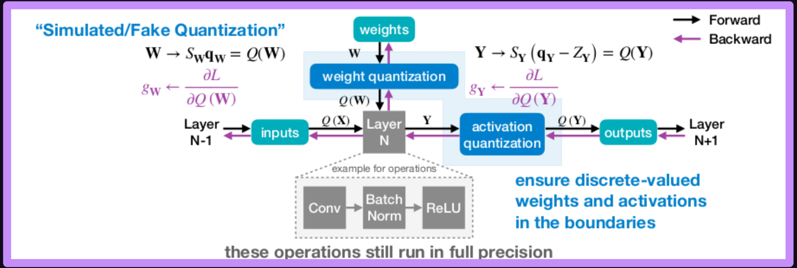
Post-training quantization: The quantization/dequantization process is learned after training.

Quantization-aware training: The quantization/dequantization is simulated during training to facilitate the quantization/dequantization process after training.

- Forward pass with straight through estimator. $WX - (WX + W_{quant}X).detach()$
- Gradient computation & update on full precision weight matrices.

Remarks

- PTQ is convenient to work on pre-trained models.
- QAT can achieve higher quality/efficiency trade-offs.



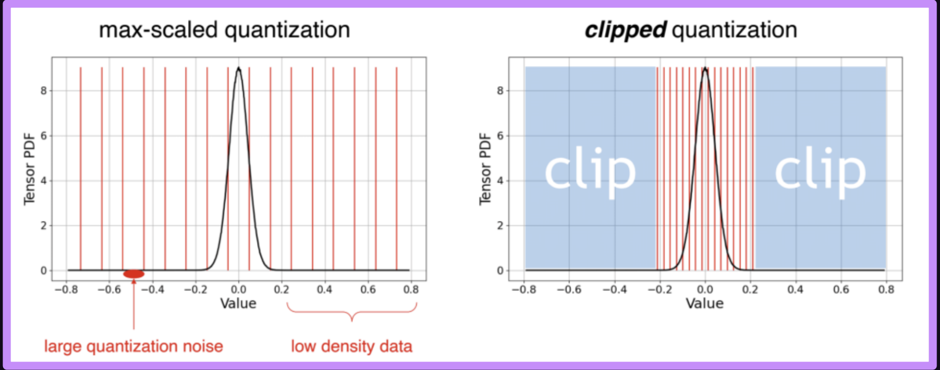
"Simulated/Fake Quantization"
weights
Forward
Backward
$\mathbf{W} \to S_\mathbf{W}\mathbf{q}_\mathbf{W} = Q(\mathbf{W})$
$g_\mathbf{W} \leftarrow \frac{\partial L}{\partial Q(\mathbf{W})}$
W
weight quantization
Q(W)
$\mathbf{Y} \to S_\mathbf{Y}(\mathbf{q}_\mathbf{Y} - Z_\mathbf{Y}) = Q(\mathbf{Y})$
$g_\mathbf{Y} \leftarrow \frac{\partial L}{\partial Q(\mathbf{Y})}$
Layer N-1
inputs
Q(X)
Layer N
Y
activation quantization
Q(Y)
outputs
Layer N+1
example for operations
Conv
Batch Norm
ReLU
ensure discrete-valued weights and activations in the boundaries
these operations still run in full precision

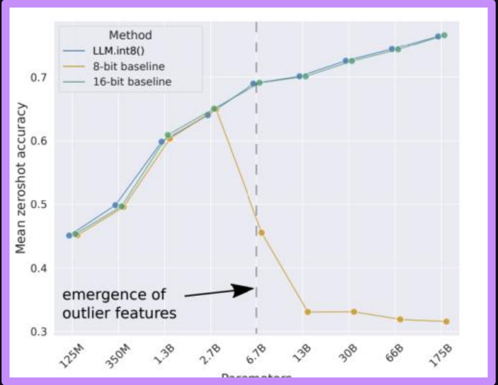
Reference:
- https://medium.com/@lmpo/understanding-model-quantization-for-llms-1573490d44ad
- https://hanlab.mit.edu/courses/2024-fall-65940

Outlier Values in Quantization

Why outliers matter in quantization?

- Outlier values drive the value of the quantization parameters.
- It creates huge quantization error.



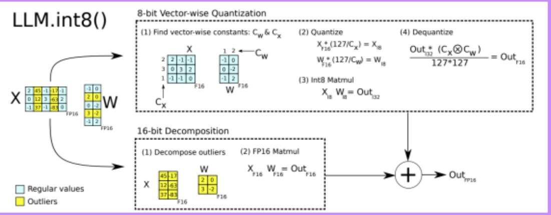


Reference:
- https://arxiv.org/abs/2206.06501
- https://hanlab.mit.edu/courses/2024-fall-65940

Outlier Values in Quantization

Option 1: Exclude outlier values

- **Do not quantize outliers:**
  - Detect outliers based on a threshold.
  - Keep channels with outliers in high precision (i.e. in 16bits), quantize others to low precision (i.e. 8bits).
  - **Example:** LLM.int8().
- **Clip outliers:**
  - Assume a distribution for the values to quantize (e.g. Gaussian, Laplace).
  - Clip all the values outside of some percentiles so that quantization parameters are not sensitive to outliers.
  - Quantize the clipped values.
  - **Example:** ACIQ, OCTAV


LLM.int8()
8-bit Vector-wise Quantization
(1) Find vector-wise constants: C_w & C_x
(2) Quantize
$X_{F16} * (127/C_x) = X_{I8}$
$W_{F16} * (127/C_w) = W_{I8}$
(3) Int8 Matmul
$X_{I8} W_{I8} = Out_{I32}$
(4) Dequantize
$\frac{Out_{I32} * (C_x \otimes C_w)}{127*127} = Out_{F16}$
16-bit Decomposition
(1) Decompose outliers
(2) FP16 Matmul
$X_{F16} W_{F16} = Out_{F16}$
Out_{F16}
Regular values
Outliers

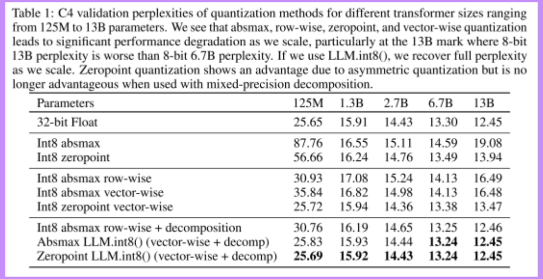


Table 1: C4 validation perplexities of quantization methods for different transformer sizes ranging from 125M to 13B parameters. We see that absmax, row-wise, zeropoint, and vector-wise quantization leads to significant performance degradation as we scale, particularly at the 13B mark where 8-bit 13B perplexity is worse than 8-bit 6.7B perplexity. If we use LLM.int8(), we recover full perplexity as we scale. Zeropoint quantization shows an advantage due to asymmetric quantization but is no longer advantageous when used with mixed-precision decomposition.

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**

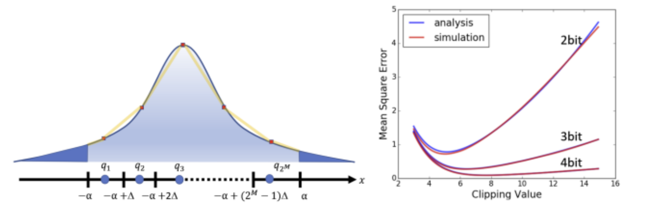


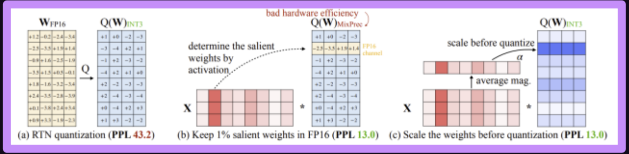
Figure 2: **left:** An activation distribution quantized uniformly in the range $[-\alpha, \alpha]$ with 2^M equal quantization intervals (bins) **right:** Expected mean-square-error as a function of clipping value for different quantization levels (Laplace ($\mu = 0$ and $b = 1$)). Analytical results, stated by Eq. 5, are in a good agreement with simulations, which where obtained by clipping and quantizing 10,000 values, generated from a Laplace distribution.

Reference:
- https://arxiv.org/abs/2206.06501
- https://arxiv.org/pdf/1810.05723
- https://arxiv.org/pdf/2206.06501

Outlier Values in Quantization

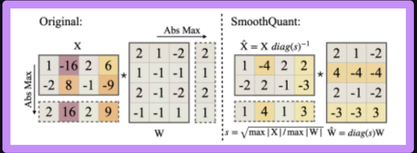
Option 2: Rescale values based on outliers

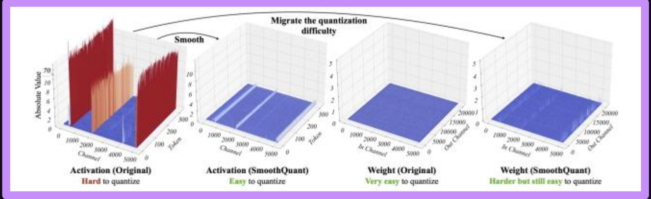
- Detect sensitive weights based on outlier activations dimensions.
  - Compute the average channel magnitude s for the activation channels.
  - Scale up by s the (sensitive) weights before quantization so that they use the full quantization range
  - Divide by s after quantization thus reducing quantization error.
  - **Example:** AWQ
- Transfer the difficulty of the per-token quantization to the weight quantization.
  - Compute the ratio s between the max of activation and weight channels.
  - Scale down/up by s activation/weights channels.
  - **Example:** SmoothQuant



W_{FP16}
$Q(W)_{INT3}$
Q
bad hardware efficiency
determine the salient weights by activation
$Q(W)_{MixPrec}$
FP16 channel
X
scale before quantize
average mag.
α
(a) RTN quantization (PPL 43.2)
(b) Keep 1% salient weights in FP16 (PPL 13.0)
(c) Scale the weights before quantization (PPL 13.0)

$$\mathrm{Err}(Q(w)x) = \Delta \cdot \mathrm{RoundErr}(\frac{w}{\Delta}) \cdot x$$
$$\mathrm{Err}(Q(w \cdot s)(\frac{x}{s})) = \Delta' \cdot \mathrm{RoundErr}(\frac{ws}{\Delta'}) \cdot x \cdot \frac{1}{s}$$



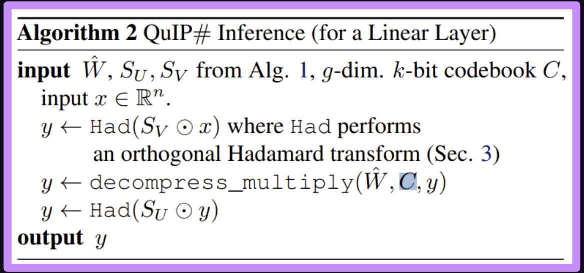


Reference:
- https://arxiv.org/abs/2206.06501

Outlier Values in Quantization

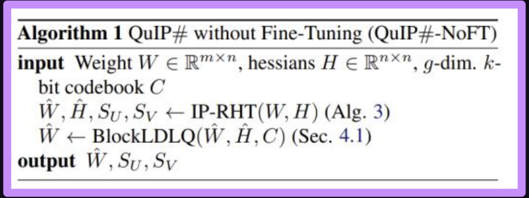
Option 3: Distribution preprocessing

- The weight distribution can be transformed in a distribution easier to quantize.
  - Preprocess the weights values with incoherence preprocessing (e.g. Kronecker or Hadamard transforms).
  - Perform inference where inputs are pre- and post-processed with Hadamard transforms.
  - **Example:** QuiP, QuiP#, QuaRot



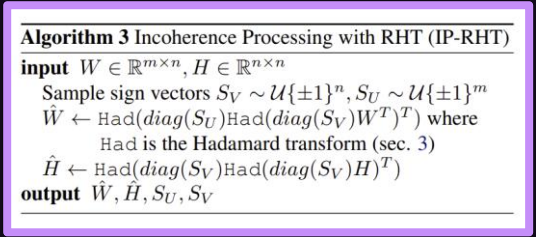
Algorithm 2 QuIP# Inference (for a Linear Layer)

```
input $\hat{W}, S_U, S_V$ from Alg. 1, $g$-dim. $k$-bit codebook $C$,
    input $x \in \mathbb{R}^n$.
    $y \leftarrow \texttt{Had}(S_V \odot x)$ where $\texttt{Had}$ performs
        an orthogonal Hadamard transform (Sec. 3)
    $y \leftarrow \texttt{decompress\_multiply}(\hat{W}, C, y)$
    $y \leftarrow \texttt{Had}(S_U \odot y)$
output $y$
```



Algorithm 1 QuIP# without Fine-Tuning (QuIP#-NoFT)

```
input Weight $W \in \mathbb{R}^{m \times n}$, hessians $H \in \mathbb{R}^{n \times n}$, $g$-dim. $k$-
    bit codebook $C$
    $\hat{W}, \hat{H}, S_U, S_V \leftarrow$ IP-RHT$(W, H)$ (Alg. 3)
    $\hat{W} \leftarrow$ BlockLDLQ$(\hat{W}, \hat{H}, C)$ (Sec. 4.1)
output $\hat{W}, S_U, S_V$
```



Algorithm 3 Incoherence Processing with RHT (IP-RHT)

```
input $W \in \mathbb{R}^{m \times n}, H \in \mathbb{R}^{n \times n}$
    Sample sign vectors $S_V \sim \mathcal{U}\{\pm 1\}^n, S_U \sim \mathcal{U}\{\pm 1\}^m$
    $\hat{W} \leftarrow \texttt{Had}(diag(S_U)\texttt{Had}(diag(S_V)W^T)^T)$ where
        $\texttt{Had}$ is the Hadamard transform (sec. 3)
    $\hat{H} \leftarrow \texttt{Had}(diag(S_V)\texttt{Had}(diag(S_V)H)^T)$
output $\hat{W}, \hat{H}, S_U, S_V$
```

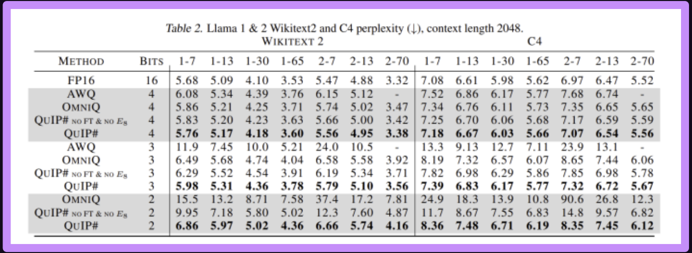


Table 2. Llama 1 & 2 Wikitext2 and C4 perplexity (↓), context length 2048.

		WIKITEXT 2							C4						
METHOD	BITS	1-7	1-13	1-30	1-65	2-7	2-13	2-70	1-7	1-13	1-30	1-65	2-7	2-13	2-70
FP16	16	5.68	5.09	4.10	3.53	5.47	4.88	3.32	7.08	6.61	5.98	5.62	6.97	6.47	5.52
AWQ	4	6.08	5.34	4.39	3.76	6.15	5.12	-	7.52	6.86	6.17	5.77	7.68	6.74	-
OMNIQ	4	5.86	5.21	4.25	3.71	5.74	5.02	3.47	7.34	6.76	6.11	5.73	7.35	6.65	5.65
QUIP# NO FT & NO E_8	4	5.83	5.20	4.23	3.63	5.66	5.00	3.42	7.25	6.70	6.06	5.68	7.17	6.59	5.59
QUIP#	4	**5.76**	**5.17**	**4.18**	**3.60**	**5.56**	**4.95**	**3.38**	**7.18**	**6.67**	**6.03**	**5.66**	**7.07**	**6.54**	**5.56**
AWQ	3	11.9	7.45	10.0	5.21	24.0	10.5	-	13.3	9.13	12.7	7.11	23.9	13.1	-
OMNIQ	3	6.49	5.68	4.74	4.04	6.58	5.58	3.92	8.19	7.32	6.57	6.07	8.65	7.44	6.06
QUIP# NO FT & NO E_8	3	6.29	5.52	4.54	3.91	6.19	5.34	3.71	7.82	6.98	6.29	5.86	7.85	6.98	5.78
QUIP#	3	**5.98**	**5.31**	**4.36**	**3.78**	**5.79**	**5.10**	**3.56**	**7.39**	**6.83**	**6.17**	**5.77**	**7.32**	**6.72**	**5.67**
OMNIQ	2	15.5	13.2	8.71	7.58	37.4	17.2	7.81	24.9	18.3	13.9	10.8	90.6	26.8	12.3
QUIP# NO FT & NO E_8	2	9.95	7.18	5.80	5.02	12.3	7.60	4.87	11.7	8.67	7.55	6.83	14.8	9.57	6.82
QUIP#	2	**6.86**	**5.97**	**5.02**	**4.36**	**6.66**	**5.74**	**4.16**	**8.36**	**7.48**	**6.71**	**6.19**	**8.35**	**7.45**	**6.12**

Reference:

- https://arxiv.org/abs/2206.06501

Iterative Quantization

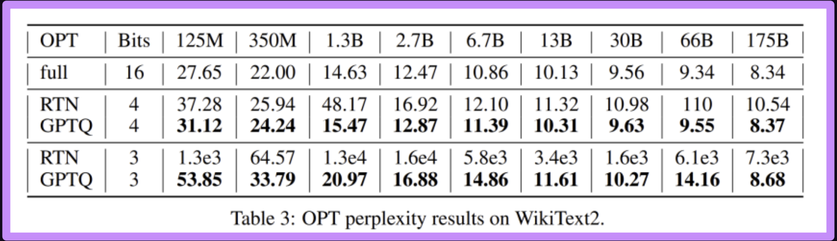
Iterative quantization aim at correcting accumulated quantization error during the quantization process:

- **Step 1:** Quantize a first set of values from $\boldsymbol{W}$.
- **Step 2:** Compute the update $\boldsymbol{\delta}$ for the non-quantized values to minimize the quantization error introduced by step 1.
- **Step 3:** Apply the update $\boldsymbol{\delta}$ to the non-quantized values in $\boldsymbol{W}$.
- **Step 4:** Loop until all values in $\boldsymbol{W}$ are quantized.

Remarks

- The order of values to quantize can be based on the quantization difficulty or random.
- Examples of quantization methods using iterative quantization are GPTQ, GPTVQ, QuiP, QuiP#.

$$w_q = \text{argmin}_{w_q} \frac{(\text{quant}(w_q) - w_q)^2}{[\mathbf{H}_F^{-1}]_{qq}}, \qquad \boldsymbol{\delta}_F = -\frac{w_q - \text{quant}(w_q)}{[\mathbf{H}_F^{-1}]_{qq}} \cdot (\mathbf{H}_F^{-1})_{:,q}.$$



OPT	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
full	16	27.65	22.00	14.63	12.47	10.86	10.13	9.56	9.34	8.34
RTN	4	37.28	25.94	48.17	16.92	12.10	11.32	10.98	110	10.54
GPTQ	4	**31.12**	**24.24**	**15.47**	**12.87**	**11.39**	**10.31**	**9.63**	**9.55**	**8.37**
RTN	3	1.3e3	64.57	1.3e4	1.6e4	5.8e3	3.4e3	1.6e3	6.1e3	7.3e3
GPTQ	3	**53.85**	**33.79**	**20.97**	**16.88**	**14.86**	**11.61**	**10.27**	**14.16**	**8.68**

Table 3: OPT perplexity results on WikiText2.

Reference:
- https://arxiv.org/pdf/2210.17323

Quantization With/Without Data

For **activation quantization,** parameters are learned during/after training on data batches.

For **weight quantization,** the need of data during quantization on the objective optimized by the quantization.

Option 1: Optimize the quantization error $\|W - \tilde{W}\|$

- It does not require data.
- Optimization based on closed-form updates exist. **Example:** HQQ.

Option 2: Optimize the activation error $\|WX - \tilde{W}X\|$

- Except if using fake data, it requires data.
- Optimization based on closed-form updates exist. **Example:** GPTQ.

Option 3: Optimize the final output error $\|f_W(X) - f_{\tilde{W}}(X)\|$

- Except if using fake data, it requires data.
- Optimization is usually based on fine-tuning. **Example:** QuiP#.

Reference:

- https://mobiusml.github.io/hqq_blog/

State-of-the-art Quantization

	Quantized Values		Linear Quantization		Codebook/Lookup table Quantization		Quantization Granularity			Quantization Time		Quantization Data	
	Weight	Activation	Symmetric	Asymmetric	Scalar	Vector	Per Tensor	Per Channel	Per Group	PTQ	QAT	With Data	Without Data
LLM.int8()	✅	✅	✅	❌	❌	❌	❌	✅	❌	✅	❌	❌	✅
Quanto	✅	✅	✅	❌	❌	❌	✅	✅	❌	✅	❌	❌	✅
AWQ	✅	❌	✅	❌	❌	❌	✅	✅	✅	✅	❌	✅	❌
SmoothQuant	✅	✅	✅	❌	❌	❌	✅	✅	❌	✅	❌	✅	❌
HQQ	✅	❌	✅	✅	❌	❌	✅	✅	✅	✅	❌	❌	✅
QuaRot	✅	✅	✅	✅	❌	❌	✅	✅	✅	✅	❌	✅	❌
QoQ	✅	✅	✅	✅	❌	❌	✅	✅	✅	✅	❌	✅	❌
GPTQ	✅	❌	✅	✅	❌	❌	✅	✅	✅	✅	❌	✅	❌
AQLM	✅	❌	❌	❌	✅	✅	✅	✅	✅	✅	❌	✅	❌
GPTVQ	✅	❌	❌	❌	✅	✅	✅	✅	✅	✅	❌	✅	❌
QuiP	✅	❌	✅	✅	❌	❌	✅	✅	✅	✅	❌	✅	❌
QuiP#	✅	❌	❌	❌	✅	✅	✅	✅	✅	✅	❌	✅	❌
QTIP	✅	❌	❌	❌	✅	✅	✅	✅	✅	✅	❌	✅	❌
Higgs	✅	❌	❌	❌	✅	✅	✅	✅	✅	✅	❌	❌	✅
QuEST	✅	✅	✅	❌	❌	❌	✅	✅	✅	❌	✅	✅	❌

Reference:
- https://medium.com/@lmpo/understanding-model-quantization-for-llms-1573490d44ad
- https://hanlab.mit.edu/courses/2024-fall-65940