CS11-711 Advanced NLP

Quantization

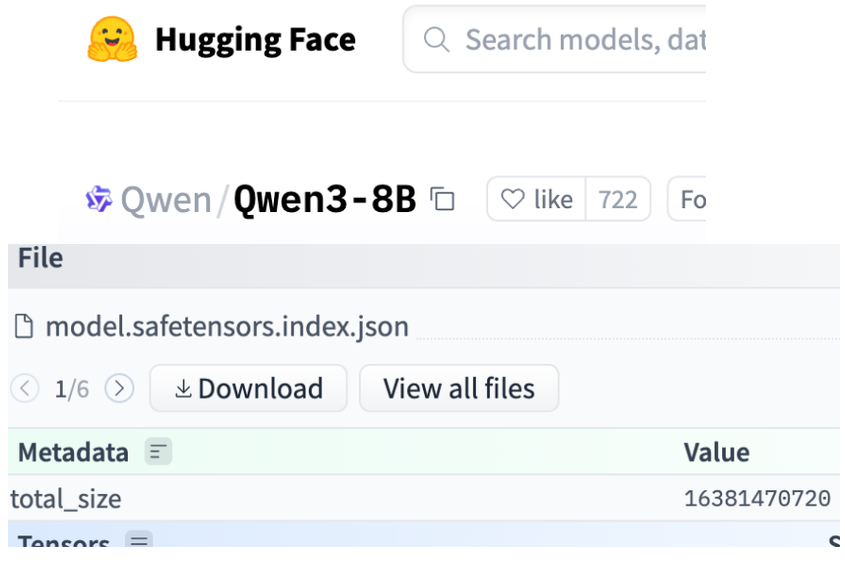
Sean Welleck

Motivation

- Key problem: Models are large.
- This makes it difficult to:
    - Run **inference** on a variety of devices
        - Laptop
        - Single GPU
    - **Fine-tune** on a variety of devices
    - **Pre-train** on many tokens

Example

- Qwen3-8B
  - 8 billion parameters x 16 bits per parameter
  - => 8 billion x 2 bytes = 16 gigabytes

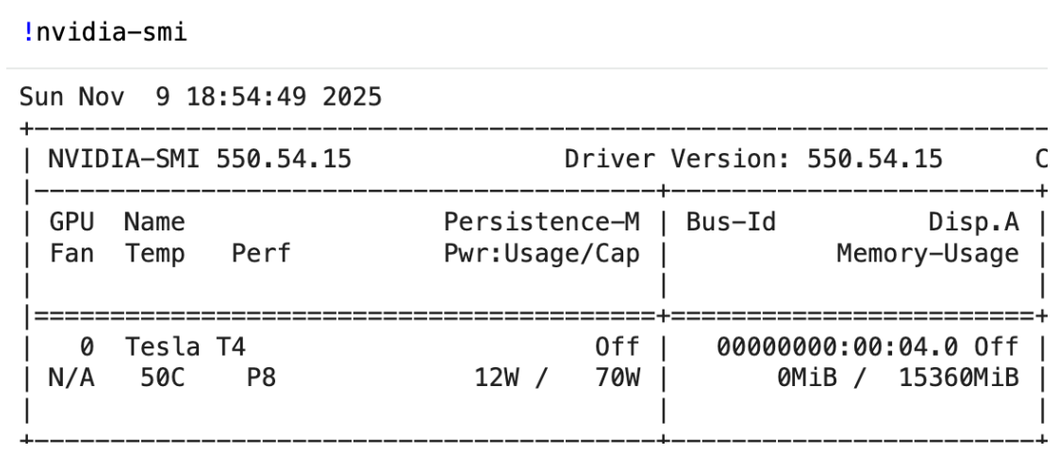


- Suppose we need around 1.5x the model weights to run inference (e.g., activations)
  - => 24 GB total memory
- MacBook Air: 16 GB shared memory

❌ Inference requires swapping to disk

- T4 GPU: 16GB memory

❌ Inference leads to out-of-memory



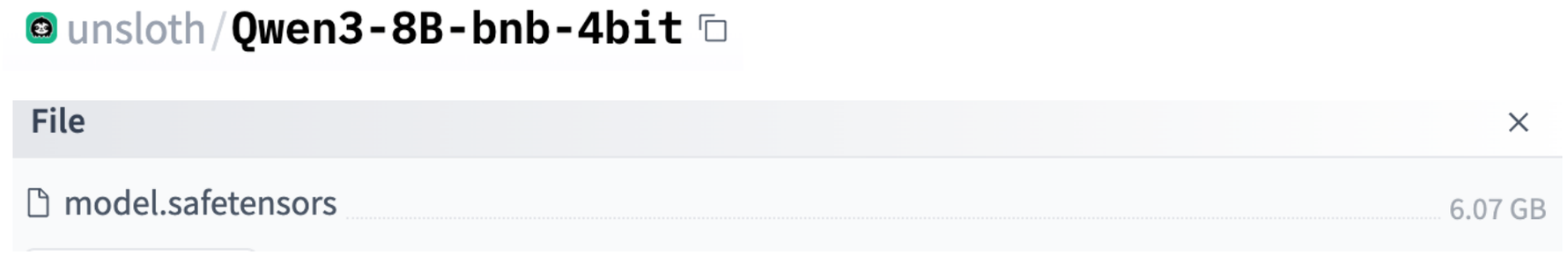
```
!nvidia-smi

Sun Nov  9 18:54:49 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      C
|-------------------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage |
|                                     |                        |
|=====================================+========================+
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
| N/A   50C    P8              12W /   70W |       0MiB /  15360MiB |
|                                     |                        |
+-------------------------------------+------------------------+
```

Quantization

- Key idea: represent each model weight (a number) using a smaller number of bits.

- Example:

- 4bit: ~6GB:



Today's lecture

- Representing numbers on a computer
- Quantizing for inference
- Quantizing & training

Representing numbers

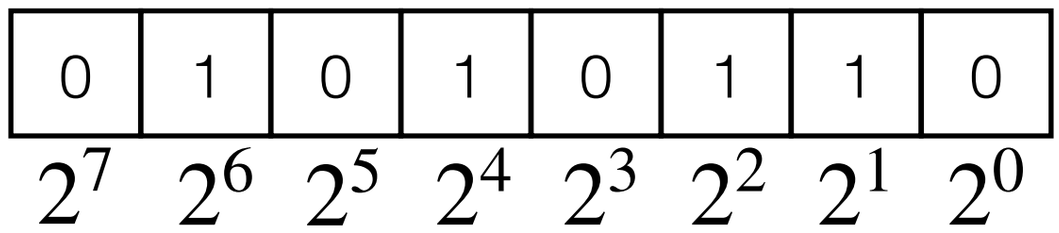
- Data is stored as a sequence of bits
  - Bit: either 0 or 1
  - Byte: 8 bits
    - 00000000
    - 01011001
    - …

Representing numbers

- Number storage depends on the number type
  - Unsigned integer: 0, 1, 2, 3, …
  - Integer: 0, -1, 1, -2, 2, -3, 3, …
  - Real values: 12.34, -0.001, …

Unsigned integer

- Each bit represents a power of 2

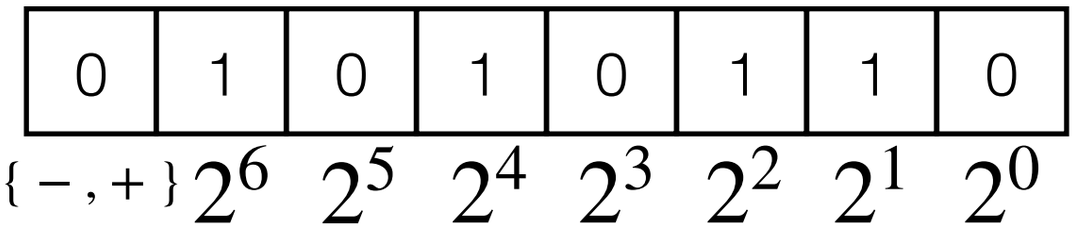


- Number: sum of powers corresponding to the positions with a 1
  - $1 \times 2^6 + 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1$
    $= 64 + 16 + 4 + 2$
    $= 86$

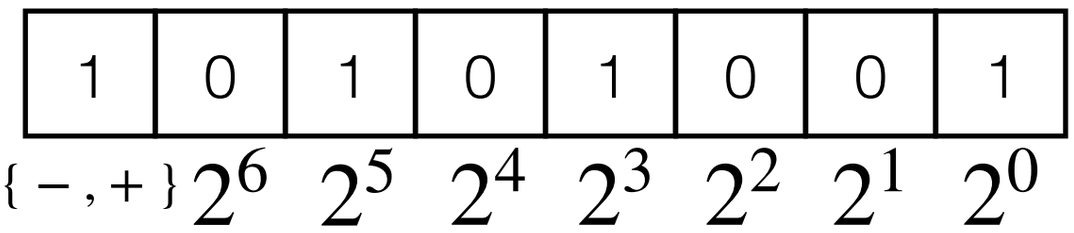
- U8: 8-bit unsigned integer: 0 to $2^8 - 1$ = 255
- U16: 16-bit unsigned integer: 0 to $2^{16} - 1$ = 65,535
- U32: 32-bit unsigned integer: 0 to 4,294,967,295

Signed integer

- Leftmost bit represents a sign



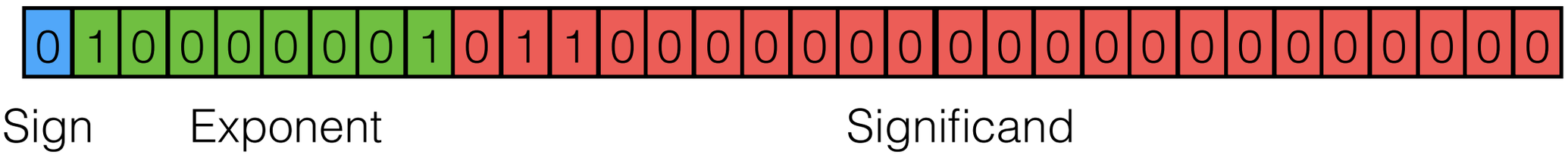
- Algorithm (“two’s complement”): flip all bits, add 1



- INT8: 8-bit [-128, 127], INT16, INT32

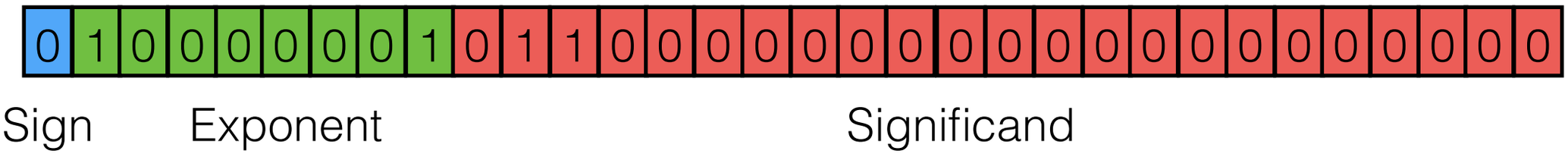
Floating point

- Represents a subset of real-valued numbers
- Sign, exponent, and significand bits



- value $= (-1)^{\text{sign}} \times (1.\text{significand}) \times 2^{\text{exponent}-\text{bias}}$
  - Bias allows for storing the exponent as an unsigned int
  - Binary scientific notation: 1.101×2^3 = (1+0.5+0.125)*8

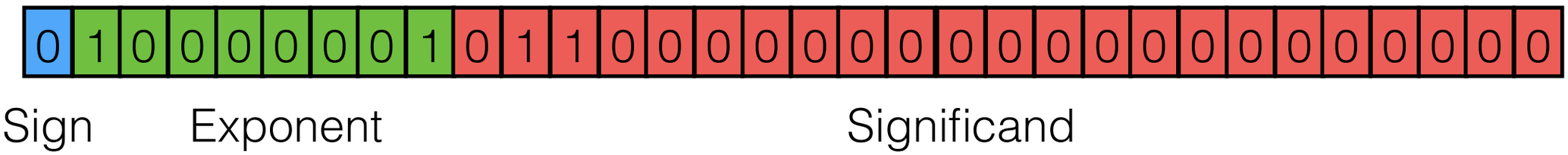
Floating point: example



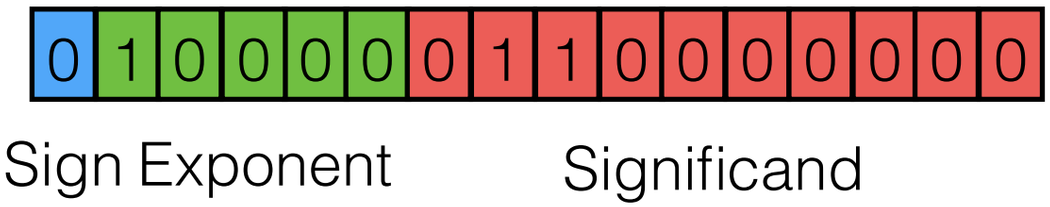
- Sign: positive
- Exponent: 10000001 = 129
  - Exponent - bias: 129 - 127 = 2
- Significand: 1.011000…0
  - $1 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 1 + 0.25 + 0.125 = 1.375$
- Value: $1.375 \times 2^2 = 5.5$

Floating point types

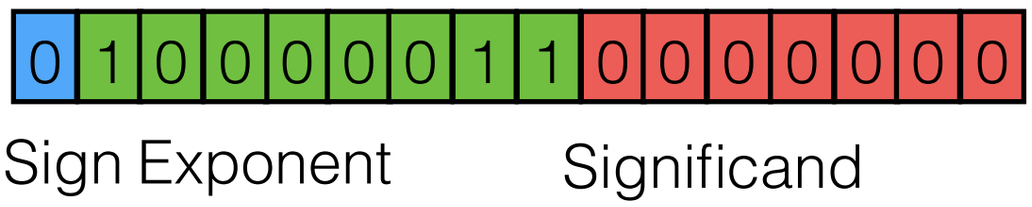
- Float32 (“full precision”)



- Float16 (“half precision”)



- BFloat16



Discuss:
Pros and cons of BFloat16?

Example

Qwen / **Qwen3-8B**

Tensors	Shape	Precision
model (3)		
model.embed_tokens.weight	[151 936, 4 096]	BF16
model.layers (36)		
model.layers.0 (4)		
model.layers.0.input_layernorm.weight	[4 096]	BF16
model.layers.0.mlp (3)		
model.layers.0.mlp.down_proj.weight	[4 096, 12 288]	BF16
model.layers.0.mlp.gate_proj.weight	[12 288, 4 096]	BF16
model.layers.0.mlp.up_proj.weight	[12 288, 4 096]	BF16
model.layers.0.post_attention_layernorm.weight	[4 096]	BF16
model.layers.0.self_attn (6)		
model.layers.0.self_attn.k_norm.weight	[128]	BF16
model.layers.0.self_attn.k_proj.weight	[1 024, 4 096]	BF16
model.layers.0.self_attn.o_proj.weight	[4 096, 4 096]	BF16
model.layers.0.self_attn.q_norm.weight	[128]	BF16
model.layers.0.self_attn.q_proj.weight	[4 096, 4 096]	BF16
model.layers.0.self_attn.v_proj.weight	[1 024, 4 096]	BF16

https://huggingface.co/Qwen/Qwen3-8B

Today's lecture

- Representing numbers on a computer
- **Quantizing for inference**
- Quantizing for training

Quantization

- The process of mapping input values from a large set to output values in a smaller set.
  - Example: map a higher-precision format (float32) to a lower-precision format (int8)
  - Example: round a real number x to the nearest integer value

$$Q(x) = \text{round}(x)$$

- Quantization error: difference between an input value and its quantized value

$$Q(1.4) = 1,\ Q(0.8) = 1$$

Quantization for LLMs

- For large transformer LMs, the feed-forward and attention projection layers and their matrix multiplications are 95% of parameters and 65-85% of computation [Ilharco et al 2020]

- Goal:
    - Quantize these parameters to less bits
        - Example: [b]float16 or float32 to INT8
    - Use low-bit-precision matrix multiplication
- Doing so will reduce memory requirements, letting us run inference with large models on various devices.

8-bit quantization: absmax

- Notation:
  - $X_{f16} \in \mathbb{R}^{T \times h}$: FP16 input matrix
  - $X_{i8} \in \mathbb{Z}^{T \times h}$: INT8 quantized matrix
- Scales inputs into the 8-bit range [-127, 127] using:

$$X_{i8} = \text{round}\left(\frac{127 \cdot X_{f16}}{\max_{ij}(|X_{f16_{ij}}|)}\right)$$

8-bit quantization: absmax

```python
# absmax quantization example

import numpy as np
x = np.array([-3.0, 1.0, 2.0, 4.0])

if np.max(np.abs(x)) != 0:
    scale = 127.0 / np.max(np.abs(x))
else:
    scale = 1.0

x_quantized = np.round(x * scale).astype(np.int8)
x_dequantized = x_quantized.astype(np.float32) / scale

print("Original array:    ", x)
print("Quantized array:   ", x_quantized)
print("Dequantized array: ", x_dequantized)

mse = np.mean((x - x_dequantized) ** 2)
print("Mean Squared Error:", mse)
```

✓ 0.0s

```
Original array:     [-3.  1.  2.  4.]
Quantized array:    [-95  32  64 127]
Dequantized array:  [-2.99212598  1.00787402  2.01574803  4.        ]
Mean Squared Error: 9.300018600037166e-05
```

```python
x = np.array([-3.112, 1.567, 2.789, 4.345])
x_quantized, x_dequantized = absmax_quantize(x)
analyze(x, x_quantized, x_dequantized)
```

✓ 0.0s

```
Original array:     [-3.112  1.567  2.789  4.345]
Quantized array:    [-91  46  82 127]
Dequantized array:  [-3.11334646  1.57377953  2.80543307  4.345
Mean Squared Error: 0.000079
```

8-bit quantization: absmax

- Matrix multiplication:

$$XW = \frac{1}{s_x s_w} Z_{i32}$$
$$\approx \frac{1}{s_x s_w} Q(X)Q(W)$$

- Where $Q(X)Q(W)$ means quantizing to 8-bit and performing 8-bit matrix multiplication, and the outputs are accumulated into an INT32 buffer

- ***Vector-wise quantization***: quantize each row of X and each column of W separately

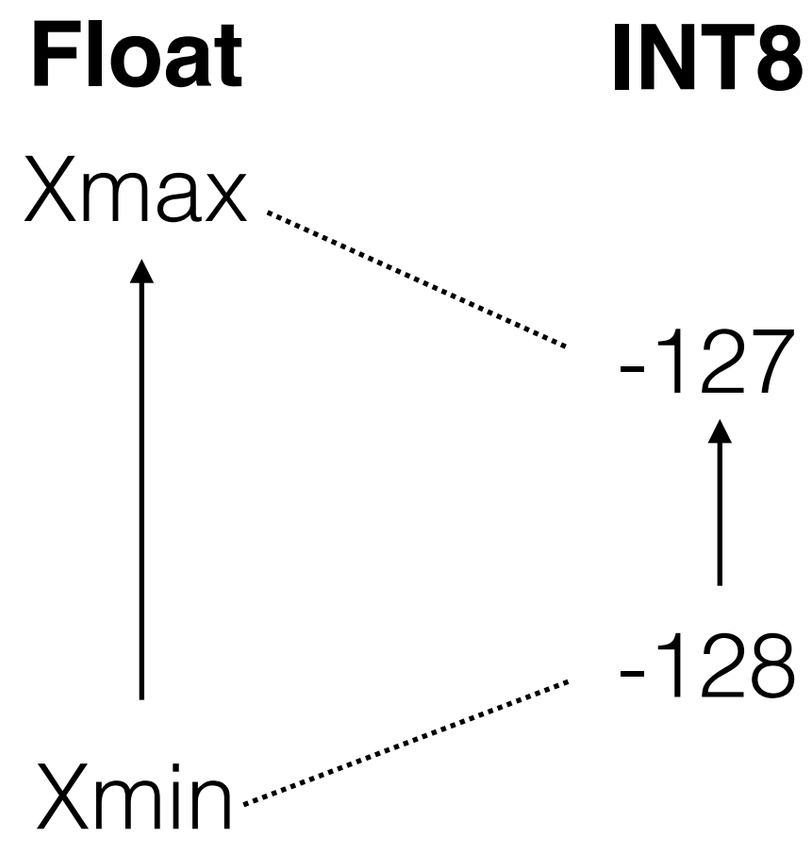
8-bit quantization: absmax

```
W (Original):
 [[-0.08636 -0.3987   0.5825 ]
 [ 0.5903   1.288   -0.5093 ]
 [ 1.753   -1.472   -1.09   ]
 [ 0.1805  -0.3015   0.7886 ]]
X (Original):
 [[-0.1633   1.0205   0.3948  -0.7993 ]
 [ 0.4998   2.674   -0.9775  -0.646  ]
 [-0.574    0.7104  -0.6064   0.7417 ]
 [-0.4246  -0.501    1.045   -0.523  ]
 [-0.5767   0.3062   0.1727  -0.9565 ]
 [ 1.231   -1.801    0.491   -0.2106 ]
 [ 0.923   -0.449   -0.3884  -0.5205 ]
 [ 0.0656   1.466   -0.02473  0.7637 ]]
W_quantized:
 [[  -6  -29   42]
 [  43   93  -37]
 [ 127 -107  -79]
 [  13  -22   57]]
X_quantized:
 [[ -8  48  19 -38]
 [ 24 127 -46 -31]
 [-27  34 -29  35]
 [-20 -24  50 -25]
 [-27  15   8 -45]
 [ 58 -86  23 -10]
 [ 44 -21 -18 -25]
 [  3  70  -1  36]]
```

```
Y (Quantized MatMul Result before scaling):
 [[  4031   3499  -5779]
 [  -928  16719  -1824]
 [ -1604   6278   1894]
 [  5113  -6452  -5327]
 [  1238   2312  -4886]
 [ -1255 -11921   3231]
 [ -3778   -753   2622]
 [  3333   5738   -333]]
Dequantized MatMul Result:
 [[ 1.1718023   1.0171511  -1.6799419 ]
 [-0.26976743  4.8601747  -0.53023255]
 [-0.46627906  1.825       0.5505814 ]
 [ 1.4863372  -1.8755814  -1.5485466 ]
 [ 0.35988373  0.67209303 -1.4203489 ]
 [-0.36482558 -3.465407    0.9392442 ]
 [-1.0982559  -0.21889535  0.7622093 ]
 [ 0.9688954   1.6680232  -0.09680232]]
Float16 MatMul Result:
 [[ 1.164   1.04   -1.676 ]
 [-0.295   4.88   -0.5146]
 [-0.4602  1.8125  0.5493]
 [ 1.479  -1.856  -1.543 ]
 [ 0.3606  0.6587 -1.435 ]
 [-0.3467 -3.469   0.933 ]
 [-1.119  -0.2177  0.7793]
 [ 0.954   1.668  -0.0791]]
Mean Squared Error between float16 and quantized matmul: 0.00018763145
```

8-bit quantization: zeropoint

- $\text{INT8} = \text{round}(s \cdot \text{FP}) + \text{offset}$
  - $s = \dfrac{255}{\max - \min}$
  - $\text{offset} = -\text{round}\left(\dfrac{255}{\max - \min}\min\right) - 128$
- $\text{FP} = \dfrac{\text{INT8} - \text{offset}}{s}$



- Good for asymmetric data (e.g., activations after a ReLU); absmax would not use any negative integers in that case.

8-bit quantization: zeropoint

```python
# zero point quantization example

import numpy as np
x = np.array([-3.0, 1.0, 2.0, 4.0])
x_min = np.min(x)
x_max = np.max(x)

qmin, qmax = -128, 127

if x_max - x_min != 0:
    scale = (qmax - qmin) / (x_max - x_min)
else:
    scale = 1.0

zero_point = np.round(qmin - x_min * scale)
zero_point = np.clip(zero_point, qmin, qmax).astype(np.int8)

x_quantized = np.round(x * scale + zero_point)
x_quantized = np.clip(x_quantized, qmin, qmax).astype(np.int8)

x_dequantized = (x_quantized.astype(np.float32) - zero_point) / scale

print("Original array:    ", x)
print("Quantized array:   ", x_quantized)
print("Dequantized array: ", x_dequantized)
print("Scale", scale)
print("Zero-point", zero_point)
mse = np.mean((x - x_dequantized) ** 2)
print("Mean Squared Error:", mse)
```

✓ 0.0s

```
Original array:     [-3.  1.  2.  4.]
Quantized array:    [-128   17   54  127]
Dequantized array:  [-2.99215686  0.98823529  2.00392157  4.00784314]
Scale 36.42857142857143
Zero-point -19
Mean Squared Error: 6.92041522491351e-05
```

Information loss

- As we observed, the mean-squared error is nonzero even in these simple examples. I.e., there is information loss.

- Since quantization depends on normalizing based on the min and max, information loss is particularly extreme in the presence of ***outliers***

With Outlier

```
With outlier:

Absmax Quantization:
Original array:     [ -0.3   0.1   0.2   0.4  -0.3   0.1   0.2   0.4  -0.3   0.1   0.2 100. ]
Quantized array:    [  0   0   0   1   0   0   0   1   0   0   0 127]
Dequantized array:  [  0.           0.           0.           0.78740157   0.
   0.           0.           0.78740157   0.           0.
   0.         100.        ]
Mean Squared Error: 0.060013

Zero-point Quantization:
Original array:     [ -0.3   0.1   0.2   0.4  -0.3   0.1   0.2   0.4  -0.3   0.1   0.2 100. ]
Quantized array:    [-128 -127 -126 -126 -128 -127 -126 -126 -128 -127 -126  127]
Dequantized array:  [-0.39333333  0.          0.39333333  0.39333333 -0.39333333  0.
  0.39333333  0.39333333 -0.39333333  0.          0.39333333 99.90666667]
Mean Squared Error: 0.014756
```

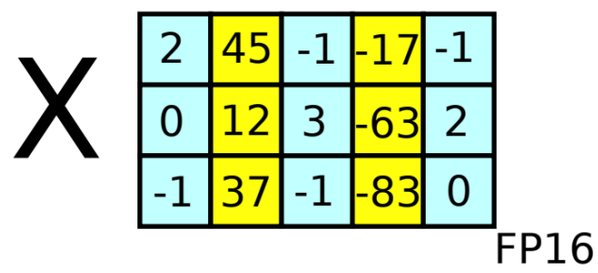
Without Outlier

```
Without outlier:

Absmax Quantization:
Original array:     [-0.3  0.1  0.2  0.4 -0.3  0.1  0.2  0.4 -0.3  0.1  0.2  0.4]
Quantized array:    [-95  32  64 127 -95  32  64 127 -95  32  64 127]
Dequantized array:  [-0.2992126  0.1007874  0.2015748  0.4       -0.2992126  0.1007874
  0.2015748  0.4       -0.2992126  0.1007874  0.2015748  0.4      ]
Mean Squared Error: 0.000001

Zero-point Quantization:
Original array:     [-0.3  0.1  0.2  0.4 -0.3  0.1  0.2  0.4 -0.3  0.1  0.2  0.4]
Quantized array:    [-128   17   54  127 -128   17   54  127 -128   17   54  127]
Dequantized array:  [-0.29921569  0.09882353  0.20039216  0.40078431 -0.29921569  0.09882353
  0.20039216  0.40078431 -0.29921569  0.09882353  0.20039216  0.40078431]
Mean Squared Error: 0.000001
```

LLM.int8()

[Dettmers et al 2022]

- Observed that Transformer LMs have *outlier features*
  - Columns of the activations that are unusually large



- In models with $\geq$ 6.7B parameters, observed that an outlier occurs in 100% of the layers

- Hypothesis: quantization results in large performance degradation due to outlier features

```python
W = np.random.randn(h, o).astype(np.float16)
X = np.random.randn(T, h).astype(np.float16)

if outliers:
    # Introduce outlier features in X
    X[:, 1] *= 100.0
```

```
W (Original):
 [[ 1.05     0.746    0.05606]
 [ 0.5435  -0.7725   0.8413 ]
 [ 1.889    0.2646   2.355  ]
 [-0.1483  -1.635    0.542  ]]
X (Original):
 [[-2.084e+00 -2.498e+01 -1.690e+00 -7.336e-02]
 [-1.869e-01 -3.581e+01 -1.252e+00 -5.234e-01]
 [-4.277e-01  2.052e+01 -6.880e-01  3.694e-01]
 [ 1.839e+00  1.438e+02  1.112e+00 -1.826e+00]
 [ 1.476e+00 -2.453e+01  1.535e+00 -6.396e-01]
 [-1.732e+00 -6.819e+01  1.178e+00 -2.742e-01]
 [ 1.381e+00 -6.009e+01  3.245e-01  9.492e-01]
 [ 1.128e+00 -7.227e-01  1.777e+00 -1.615e+00]]
W_quantized:
 [[ 57  40   3]
 [ 29 -42  45]
 [102  14 127]
 [ -8 -88  29]]
X_quantized:
 [[ -2 -22  -1   0]
 [  0 -32  -1   0]
 [  0  18  -1   0]
 [  2 127   1  -2]
 [  1 -22   1  -1]
 [ -2 -60   1   0]
 [  1 -53   0   1]
 [  1  -1   2  -1]]
```

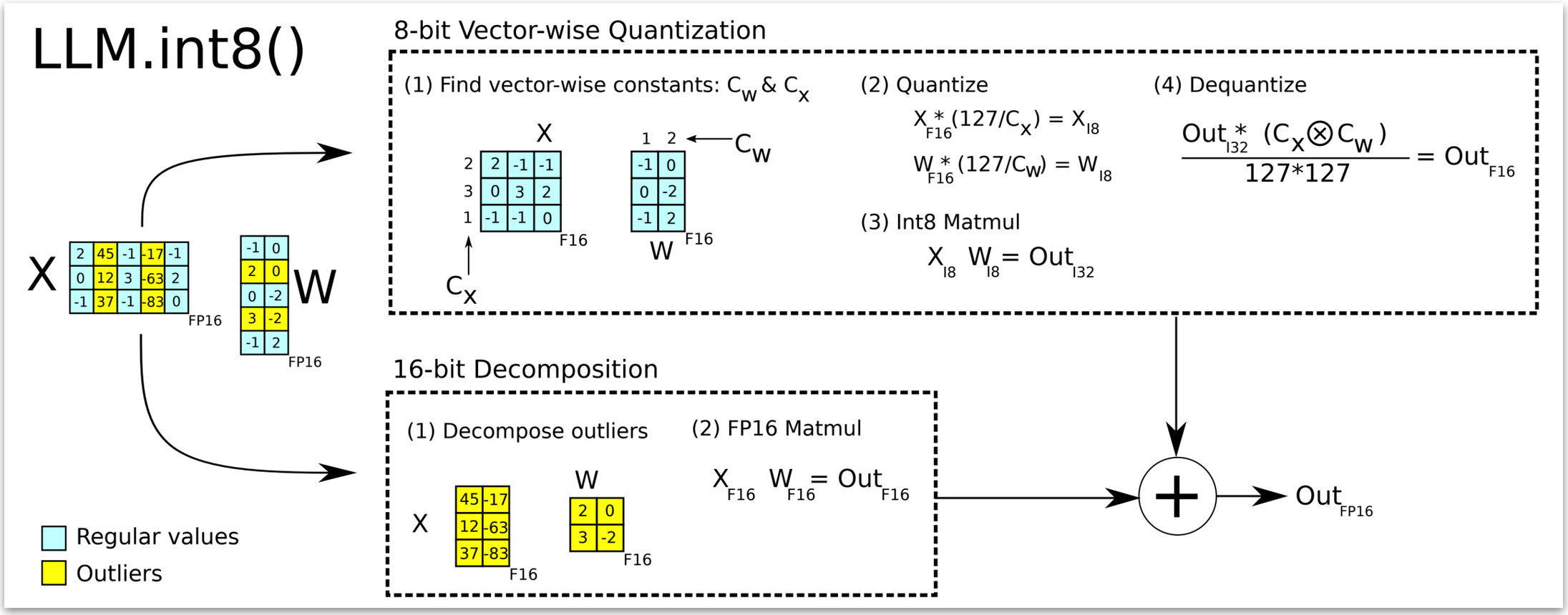
```
Y (Quantized MatMul Result before scaling):
 [[ -854   830 -1123]
 [-1030  1330 -1567]
 [  420  -770   683]
 [ 3915 -5064  5790]
 [ -471  1066  -889]
 [-1752  2454 -2579]
 [-1488  2178 -2353]
 [  240   198   183]]
Dequantized MatMul Result:
 [[ -17.931759    17.427822   -23.580053 ]
 [ -21.627296    27.92651    -32.902885 ]
 [   8.818897   -16.167978    14.3412075]
 [  82.20473   -106.33071    121.57481  ]
 [  -9.889764    22.383202   -18.666666 ]
 [ -36.787403    51.527557   -54.15223  ]
 [ -31.244095    45.732285   -49.406826 ]
 [   5.03937      4.1574802    3.8425198]]
Float16 MatMul Result:
 [[ -18.95    17.42   -25.16 ]
 [ -21.95    28.05   -33.38 ]
 [   9.34   -16.95    15.81 ]
 [  82.44  -106.4    122.7  ]
 [  -8.79    21.5    -17.28 ]
 [ -36.62    52.12   -54.84 ]
 [ -30.73    46.     -49.2  ]
 [   4.387    4.51     2.766]]
Mean Squared Error between float16 and quantized matmul: 0.6285514
```

LLM.int8()

[Dettmers et al 2022]

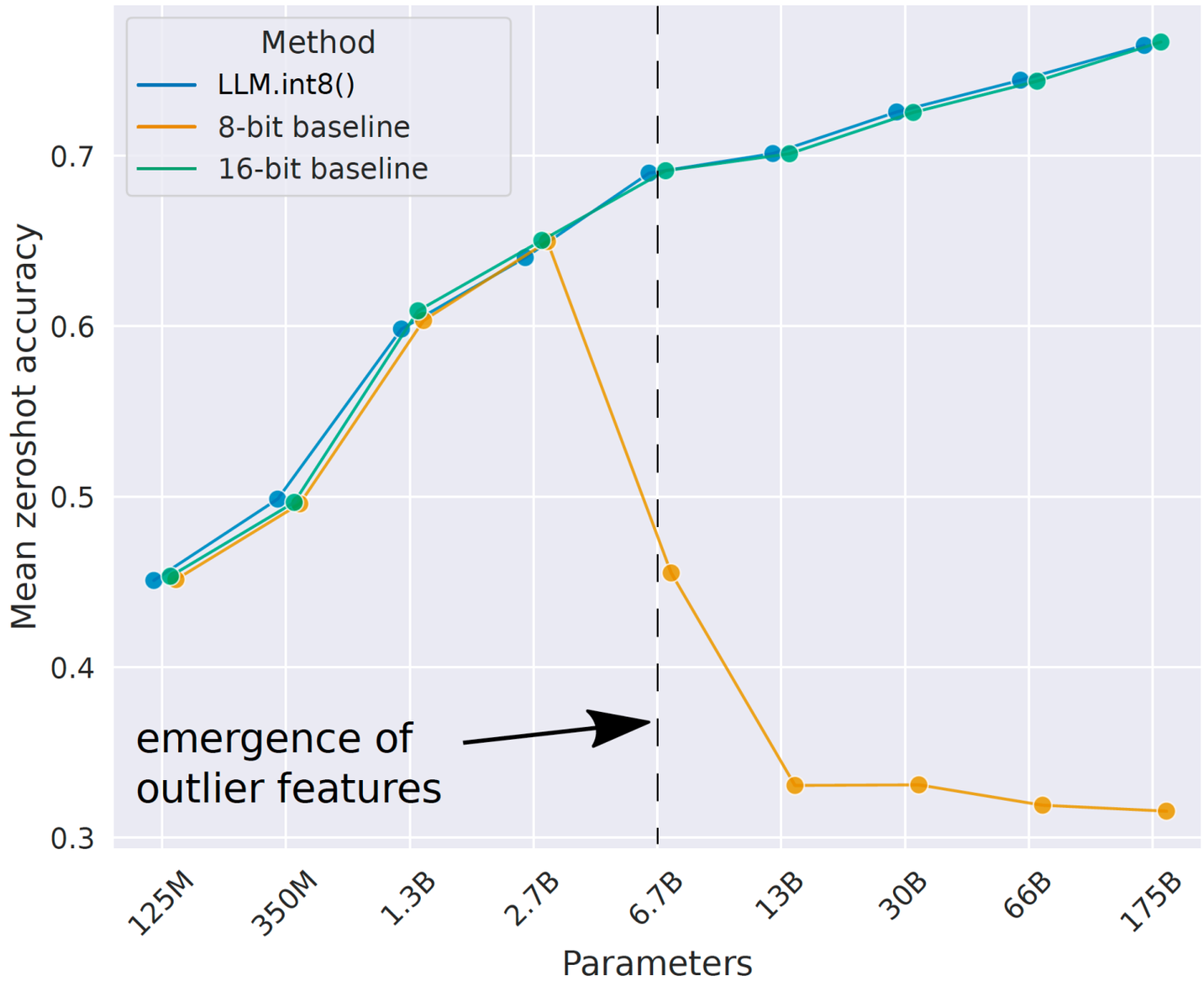
- Approach: perform computations that involve outliers in higher precision, perform other computations in lower precision, merge the results

  - Note: only ~0.1% of activation features are outliers



LLM.int8()

[Dettmers et al 2022]



Zeroshot accuracy

LLM.int8()

[Dettmers et al 2022]

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**

Validation perplexity

GGML/GGUF/Llama.cpp

- GGML: a library for tensor operations, including its own quantization methods
    - Example method: Q4_K https://github.com/ggml-org/llama.cpp/pull/1684
        - Groups weights into super-blocks containing 8 blocks, each block having 32 weights. Each weight is 4 bits.
        - Computes per-block scale and offset, stores as 6-bit
            - Q4_K_S
                - Use Q4_K for all tensors
            - Q4_K_M
                - Use Q4_6 (6-bit, 16x16 super blocks) for half of certain attention and feed forward tensors
    - Supports multiple backends, e.g. Metal, ARM, CUDA, …
- GGUF: a file format for storing neural network architectures
- Llama.cpp: a library for LLM inference that uses GGML and GGUF

GGML/GGUF/Llama.cpp

master ggml / src / ggml-quants.c

Code Blame

```
375
376 static void quantize_row_q4_K_impl(const float * GGML_RESTRICT x, block_q4_K
377     assert(n_per_row % QK_K == 0);
378     const int64_t nb = n_per_row / QK_K;
379
380     uint8_t L[QK_K];
381     uint8_t Laux[32];
382     uint8_t Ls[QK_K/32];
383     uint8_t Lm[QK_K/32];
384     float   weights[32];
385     float   sw[QK_K/32];
386     float   mins[QK_K/32];
387     float   scales[QK_K/32];
388
389     for (int i = 0; i < nb; i++) {
390
391         float sum_x2 = 0;
392         for (int l = 0; l < QK_K; ++l) sum_x2 += x[l] * x[l];
```

master ggml / src / ggml-metal / ggml-metal.metal

ode Blame

```
605 template <typename type4x4>
606 void dequantize_q4_K(device const block_q4_K * xb, short il, thread type4x4 & reg) {
607     device const uchar * q = xb->qs;
608
609     short is = (il/4) * 2;
610     q = q + (il/4) * 32 + 16 * (il&1);
611     il = il & 3;
612     const uchar2 sc = get_scale_min_k4_just2(is, il/2, xb->scales);
613     const float d   = il < 2 ? xb->d : xb->d / 16.h;
614     const float min = xb->dmin;
615     const float dl = d * sc[0];
616     const float ml = min * sc[1];
617
618     const ushort mask = il < 2 ? 0x0F : 0xF0;
619     for (int i = 0; i < 16; ++i) {
620         reg[i/4][i%4] = dl * (q[i] & mask) - ml;
621     }
622 }
623
```

Example

Qwen / **Qwen3-8B-GGUF**

Hardware compatibility — Log In to view the estimation

4-bit	Q4_K_M | 5.03 GB
5-bit	Q5_0 | 5.72 GB · Q5_K_M | 5.85 GB
6-bit	Q6_K | 6.73 GB
8-bit	Q8_0 | 8.71 GB

littlebird13 Update README.md 7c41481 VERIFIED

- .gitattributes
- LICENSE Safe
- Qwen3-8B-Q4_K_M.gguf Safe
- Qwen3-8B-Q5_0.gguf
- Qwen3-8B-Q5_K_M.gguf
- Qwen3-8B-Q6_K.gguf Safe
- Qwen3-8B-Q8_0.gguf
- README.md
- params Safe

Tensors	Shape	Precision
token_embd.weight	[4 096, 151 936]	Q4_K
blk (36)		
blk.0 (11)		
blk.0.attn_k.weight	[4 096, 1 024]	Q4_K
blk.0.attn_k_norm.weight	[128]	F32
blk.0.attn_norm.weight	[4 096]	F32
blk.0.attn_q.weight	[4 096, 4 096]	Q4_K
blk.0.attn_q_norm.weight	[128]	F32
blk.0.attn_v.weight	[4 096, 1 024]	Q6_K
blk.0.ffn_down.weight	[12 288, 4 096]	Q6_K
blk.0.ffn_gate.weight	[4 096, 12 288]	Q4_K
blk.0.ffn_norm.weight	[4 096]	F32
blk.0.ffn_up.weight	[4 096, 12 288]	Q4_K
blk.0.attn_output.weight	[4 096, 4 096]	Q4_K

Code example

```
!llama-server -hf Qwen/Qwen3-8B-GGUF:Q4_K_M --port 8080
15.0s

ggml_metal_device_init: tensor API disabled for pre-M5 device
ggml_metal_library_init: using embedded metal library
ggml_metal_library_init: loaded in 0.006 sec
ggml_metal_device_init: GPU name:   Apple M4
ggml_metal_device_init: GPU family: MTLGPUFamilyApple9  (1009)
```

localhost:8080/#

tell me about the Advanced NLP 11-711 course at CMU

Reasoning

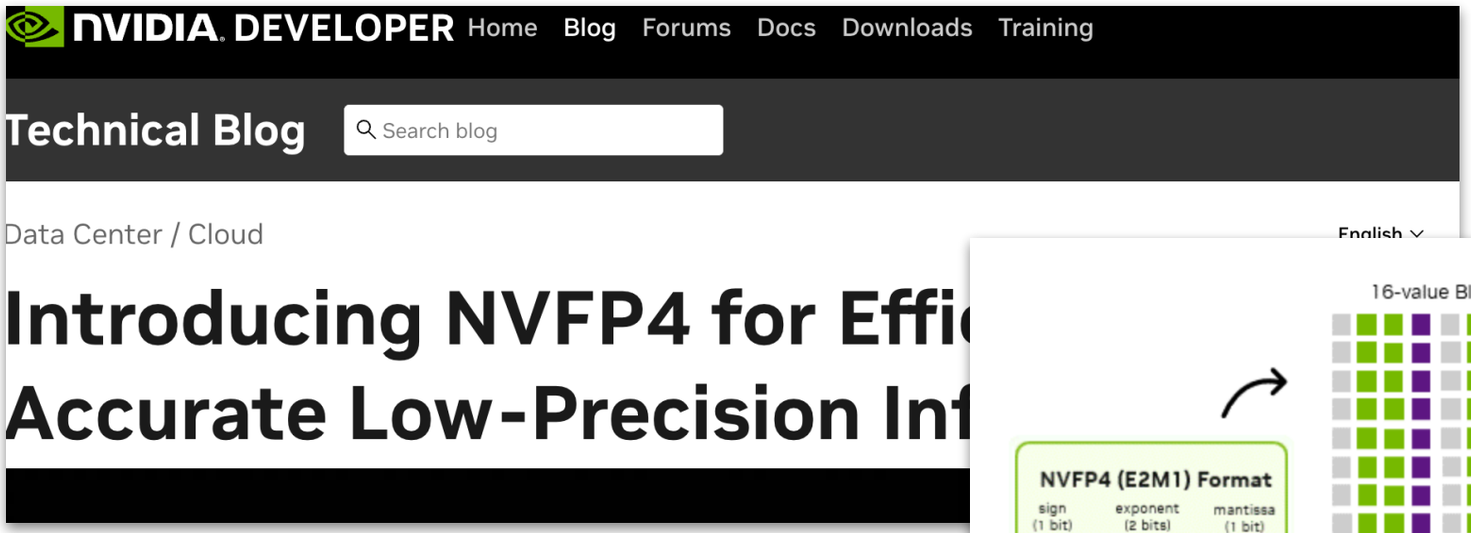
Okay, the user is asking about the Advanced NLP course at CMU, specifically 11-711. Let me start by recalling what I know about this course. I remember that CMU's School of Computer Science offers several NLP courses, and 11-711 is likely an advanced-level one.

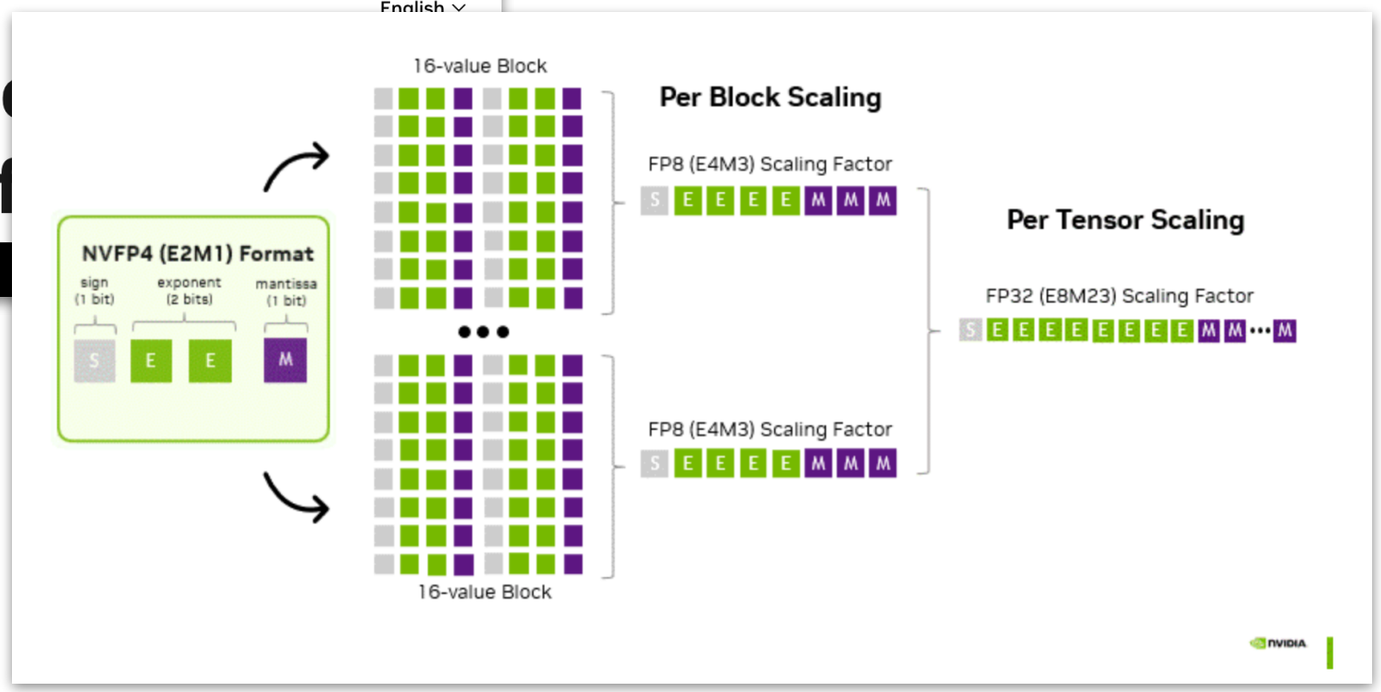
First, I need to confirm the course details. The course number 11-711 is probably a graduate-level course. I should check if there are any official course descriptions or syllabi available. Wait, the user might not have access to that, so I should provide a general overview based on common structures of such courses.

Advanced NLP courses typically cover topics beyond the basics, like deep learning for NLP, sequence modeling, transformer

Beyond integers

- New data types are being developed
  - NVFP4: 4-bit format introduced with the Blackwell GPU Architecture





Tradeoffs

- Quantization reduces the memory requirements for running a model

- In practice, may see performance degradation based on the quantization method

Today's lecture

- Representing numbers on a computer
- Quantizing for inference
- **Quantizing & training**

Quantization-Aware Training (QAT)

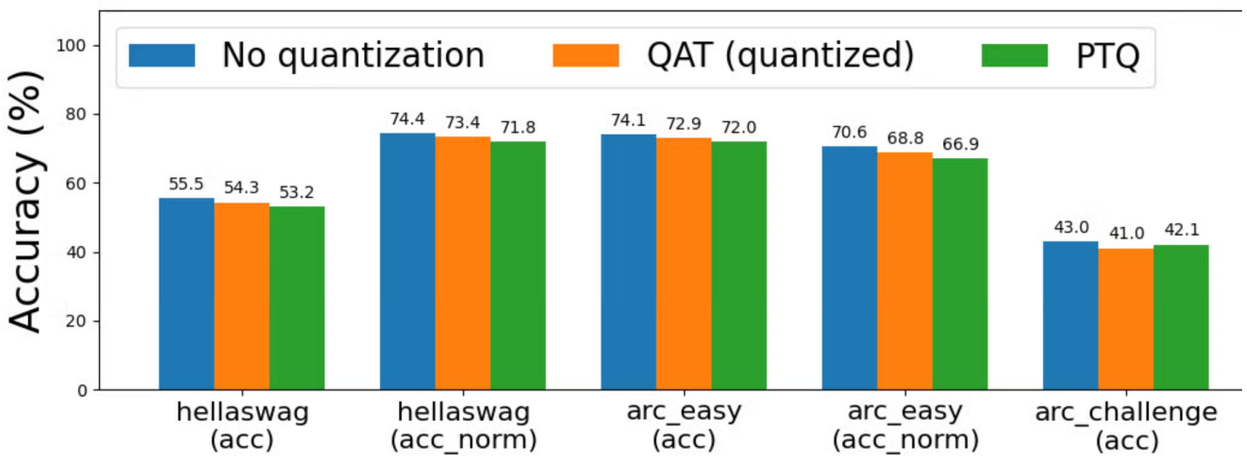
- Motivation: fine-tune a model so that it is “prepared” for quantization at inference time

- Basic idea: insert artificial quantization operations into the computational graph

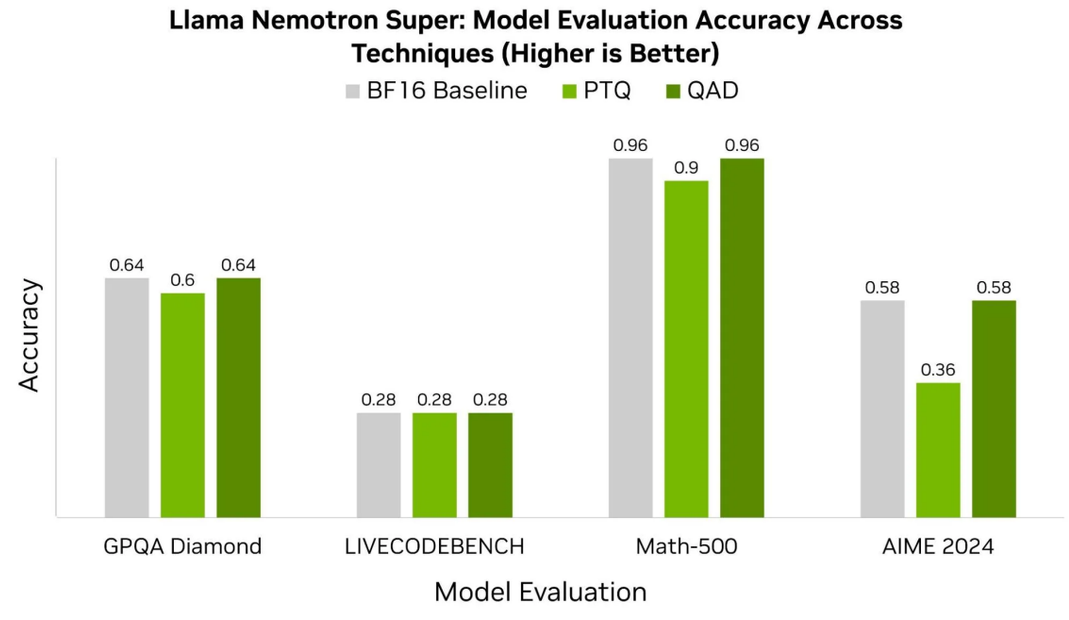
Quantization-Aware Training (QAT)

- Add a quantize-dequantize operation in model layers

```
# QAT: x_fq is still in float
# Fake quantize simulates the numerics of quantize + dequantize
x_fq = (x_float / scale + zp).round().clamp(qmin, qmax)
x_fq = (x_fq - zp) * scale
```

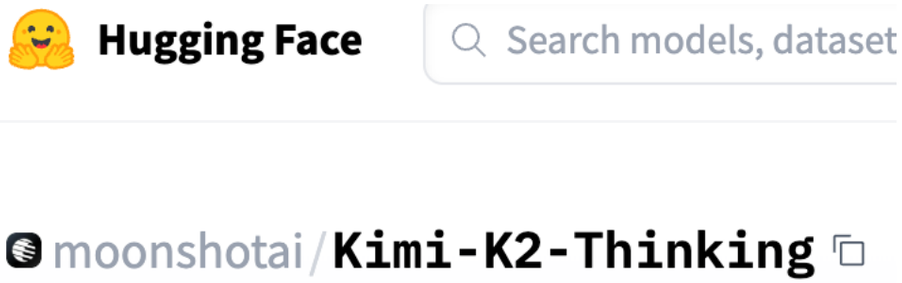


https://pytorch.org/blog/quantization-aware-training/



https://developer.nvidia.com/blog/how-quantization-aware-training-enables-low-precision-accuracy-recovery/

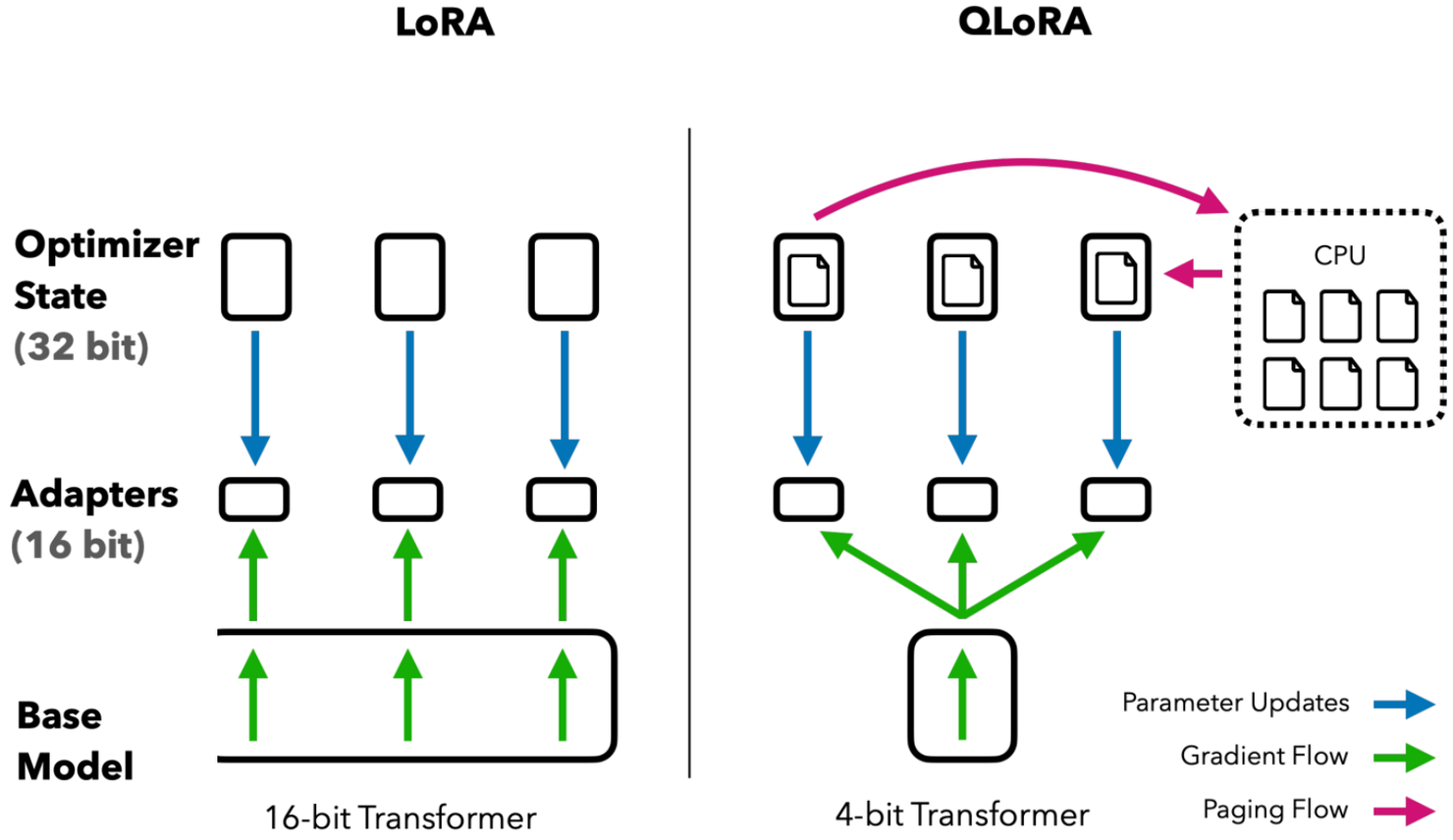
Example:


Hugging Face
Search models, dataset
moonshotai / Kimi-K2-Thinking
huggingface.co/moonshotai/Kimi-K2-Thinking
Key Features
Deep Thinking & Tool Orchestration: End-to-end trained to interleave chain-of-thought reasoning with function calls, enabling autonomous research, coding, and writing workflows that last hundreds of steps without drift.
Native INT4 Quantization: Quantization-Aware Training (QAT) is employed in post-training stage to achieve lossless 2x speed-up in low-latency mode.
Stable Long-Horizon Agency: Maintains coherent goal-directed behavior across up to 200–300 consecutive tool invocations, surpassing prior models that degrade after 30–50 steps.

Fine-tuning: QLora

[Dettmers et al 2023]

- Basic idea: use quantization and memory offloading to reduce the GPU memory requirements needed to fine-tune large models



Recap

- Representing numbers on a computer
    - Floating-point, integers
- Quantizing for inference
    - Absmax and zero-point quantization
    - Outliers and LLM.int8()
    - GGML and Llama CPP
- Quantizing & training
    - Quantization-aware training
    - Fine-tuning: QLora

Many other methods

Quantization Method	On the fly quantization	CPU	CUDA GPU	ROCm GPU	Metal (Apple Silicon)	Intel GPU	Torch compile()	Bits	PEFT Fine Tuning	Serializable with 🤗 Transformers	🤗 Transform Suppor
AQLM	🔴	🟢	🟢	🔴	🔴	🟢	🟢	1/2	🟢	🟢	🟢
AutoRound	🔴	🟢	🟢	🔴	🔴	🟢	🔴	2/3/4/8	🔴	🟢	🟢
AWQ	🔴	🟢	🟢	🟢	🔴	🟢	?	4	🟢	🟢	🟢
bitsandbytes	🟢	🟢	🟢	🟡	🟡	🟢	🟢	4/8	🟢	🟢	🟢
compressed-tensors	🔴	🟢	🟢	🟢	🔴	🔴	🔴	1/8	🟢	🟢	🟢
EETQ	🟢	🔴	🟢	🔴	🔴	🔴	?	8	🟢	🟢	🟢
FP-Quant	🟢	🔴	🟢	🔴	🔴	🔴	🟢	4	🔴	🟢	🟢
GGUF / GGML (llama.cpp)	🟢	🟢	🟢	🔴	🟢	🟢	🔴	1/8	🔴	See Notes	See Notes
GPTQModel	🔴	🟢	🟢	🟢	🟢	🟢	🔴	2/3/4/8	🟢	🟢	🟢
AutoGPTQ	🔴	🔴	🟢	🟢	🔴	🔴	🔴	2/3/4/8	🟢	🟢	🟢
HIGGS	🟢	🔴	🟢	🔴	🔴	🔴	🟢	2/4	🔴	🟢	🟢
HQQ	🟢	🟢	🟢	🔴	🔴	🟢	🟢	1/8	🟢	🔴	🟢
optimum-quanto	🟢	🟢	🟢	🔴	🟢	🟢	🟢	2/4/8	🔴	🔴	🟢
FBGEMM_FP8	🟢	🔴	🟢	🔴	🔴	🔴	🔴	8	🔴	🟢	🟢
torchao	🟢	🟢	🟢	🔴	🟡	🟢		4/8		🟢🔴	🟢
VPTQ	🔴	🔴	🟢	🟡	🔴	🔴	🟢	1/8	🔴	🟢	🟢
FINEGRAINED_FP8	🟢	🔴	🟢	🔴	🔴	🟢	🔴	8	🔴	🟢	🟢
SpQR	🔴	🔴	🟢	🔴	🔴	🔴	🟢	3	🔴	🟢	🟢
Quark	🔴	🟢	🟢	🟢	🟢	🟢	?	2/4/6/8/9/16	🔴	🔴	🟢

https://huggingface.co/docs/transformers/v4.57.1/quantization/overview

Thank you