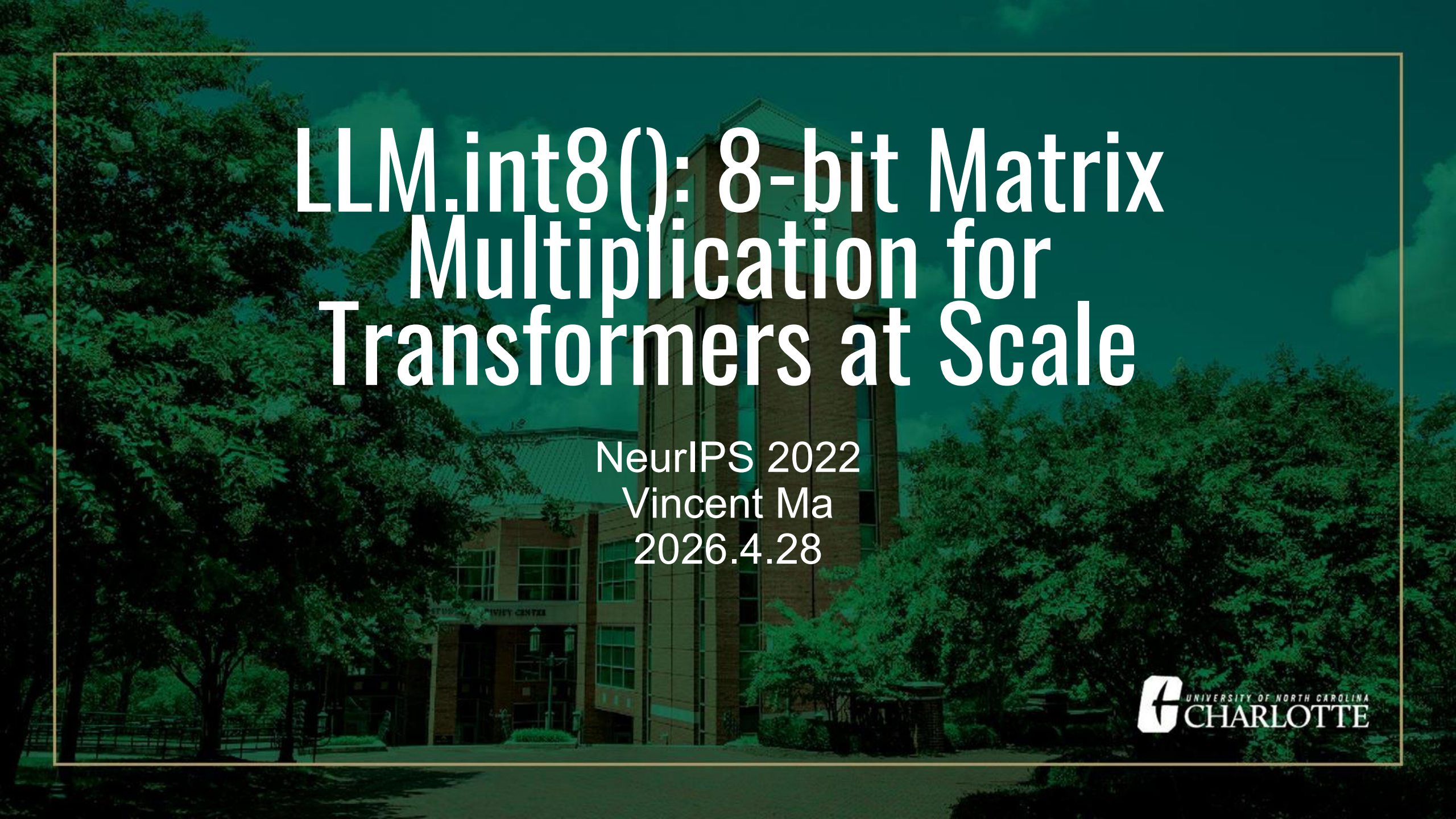


LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

NeurIPS 2022

Vincent Ma

2026.4.28

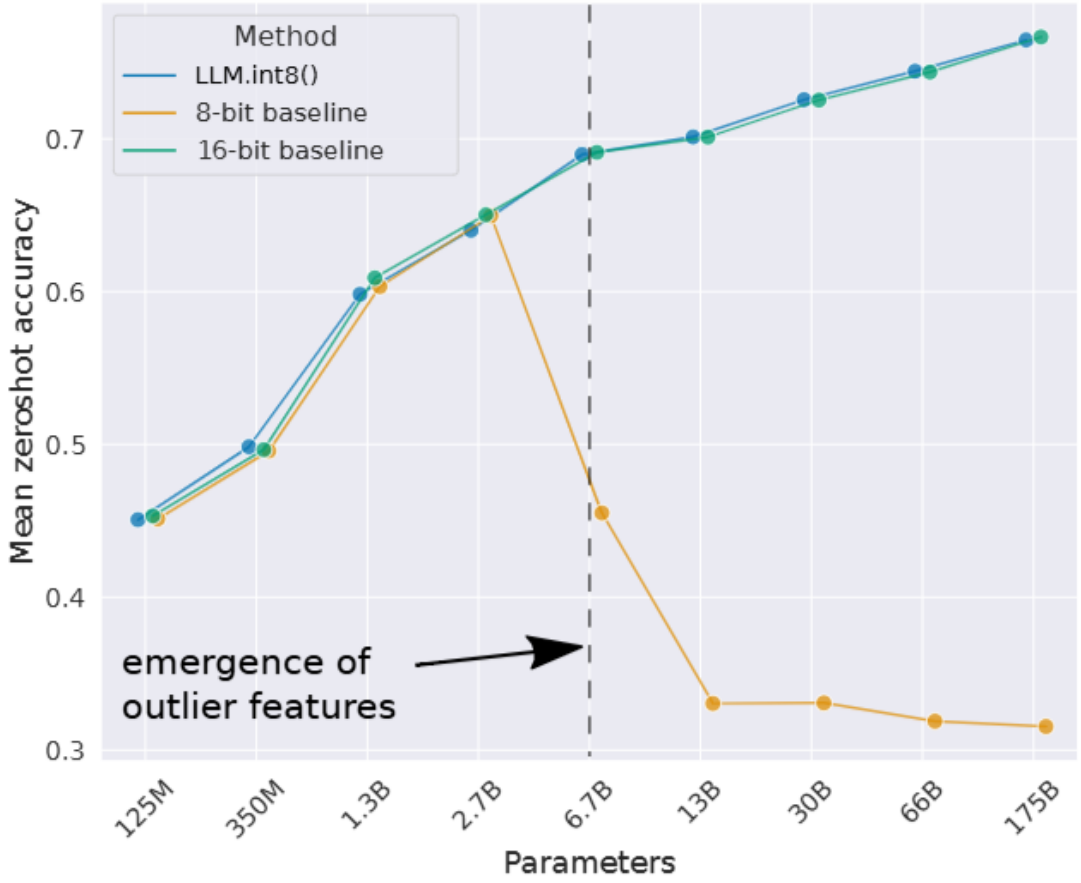
UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Quantization

- Neural network weights are typically stored in **full precision** (FP32 or BF16), which is **memory-intensive** and **computationally expensive**
- Quantization reduces the bit-width of model parameters — e.g., from 32-bit floats down to 8, 4 bits — by mapping continuous weight values to a discrete set using a quantization function $Q(\theta_j)$
- This introduces **quantization error**: the difference between the original weight and its quantized approximation, which accumulates across layers and can degrade model utility

Large Models Are Difficult to Use

- OPT-175B in 16-bit would take up 175B*2 byte=350GiB of GPU memory.
- With 8-bit weights, we can reduce memory usage from 350 GiB to 175 GiB. Does it work?
- LLM.int8() **reduces memory** and **preserves full-precision performance** even at multi-billion-parameter scale at the same time.



Absmax Quantization

- **Example:** to quantize a 16-bit Floating Point (FP16) tensor into a Int8 tensor with range [-128, 127]:
  - Given FP16 [1.2, -3.1, 0.8, 2.4, **5.4**]
  - Scale = 127 / 5.4 = 23.5
  - New Int8 [28, -73, 19, 56, **127**]

$$\mathbf{X}_{\text{quant}} = \text{round}\left(\frac{127}{\max|\mathbf{X}|} \cdot \mathbf{X}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\max|\mathbf{X}|}{127} \cdot \mathbf{X}_{\text{quant}}$$

Zeropoint Quantization

- **Example:** to quantize a FP16 tensor into a Int8 tensor with range [-128, 127]:
  - Given FP16 [1.2, **-3.1**, 0.8, 2.4, **5.4**]
  - Scale = 255 / (5.4-(-3.1)) = 30
  - Zeropoint = -round(30*-3.1)-128=-35
  - New Int8 [1, **-128**, -11, 37, **127**]

$$\text{scale} = \frac{255}{\max(\mathbf{X}) - \min(\mathbf{X})}$$

$$\text{zeropoint} = -\text{round}(\text{scale} \cdot \min(\mathbf{X})) - 128$$

$$\mathbf{X}_{\text{quant}} = \text{round}\left(\text{scale} \cdot \mathbf{X} + \text{zeropoint}\right)$$

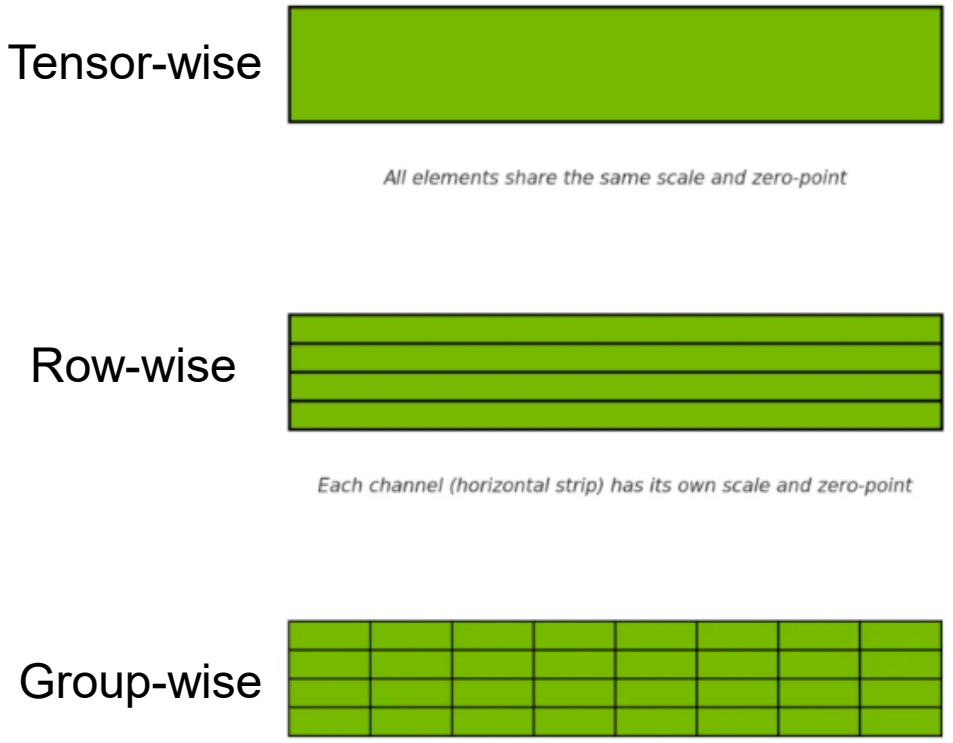
$$\mathbf{X}_{\text{dequant}} = \frac{\mathbf{X}_{\text{quant}} - \text{zeropoint}}{\text{scale}}$$

Problem with outliers

- **Example:** to quantize a FP16 tensor into a Int8 tensor with range [-128, 127] using Zeropoint quantization:
  - Scenario 1:
    - Given FP16 [1.2, **-3.1**, 0.8, 2.4, **5.4**]
    - New Int8 [1, **-128**, -11, 37, **127**]
  - Scenario 2:
    - Given FP16 [1.2, **-3.1**, 0.8, 2.4, **99.0**]
    - New Int8 [-117, **-128**, -118, -114, **127**]
    - **A single outlier can reduce the quantization precision of all other values!**

Problem with outliers

- Solution:
  - Block-wise quantization
  - Confine the effect of outliers to each block
- Finer granularity = smaller groups
  - More scaling factors → **lower quantization error**
  - More scaling factors → **higher memory overhead**
  - But memory overhead is small compared to the savings from quantization itself



Vector-wise Quantization(with absmax)

$$X_{f16} = \begin{bmatrix} 0.1 & 0.5 & 200.0 \\ 1.2 & 0.9 & 1.1 \end{bmatrix}, W_{f16} = \begin{bmatrix} 0.3 & 1.5 \\ 0.8 & 100.0 \\ 0.2 & 0.6 \end{bmatrix}$$

$$X_{f16} = \begin{bmatrix} 0.1 & 0.5 & 200.0 \\ 1.2 & 0.9 & 1.1 \end{bmatrix} \quad c_{x_1} = \frac{200}{127} \quad c_{x_2} = \frac{1.2}{127}$$

$c_{x_{f16}}$

$$W_{f16} = \begin{bmatrix} 0.3 & 1.5 \\ 0.8 & 100.0 \\ 0.2 & 0.6 \end{bmatrix}$$

$c_{w_1} = \frac{0.8}{127}$ $c_{w_2} = \frac{100}{127}$ $c_{w_{f16}}$

$$A_{i8} = \begin{bmatrix} 0 & 0 & 127 \\ 127 & 95 & 116 \end{bmatrix}, B_{i8} = \begin{bmatrix} 48 & 2 \\ 127 & 127 \\ 32 & 1 \end{bmatrix}$$

Vector-wise Quantization

$$C_{i32} = A_{i8} \cdot B_{i8}$$

- To dequantize:

$$S = \frac{1}{\mathrm{c_{x_{f16}}} \otimes \mathrm{c_{w_{f16}}}} = \frac{1}{\begin{bmatrix} c_{x_1} \\ c_{x_2} \end{bmatrix} \otimes [c_{\mathrm{w}_1} \quad c_{\mathrm{w}_2}]}$$

$$C_{f16} = S \odot C_{i32}$$

Mixed-precision Decomposition

- **Problem:**
  - Billion-scale transformers contain large-magnitude "outlier features" (specific columns) that are **critical for model performance**
  - Yet vector-wise quantization operates row-wise over hidden states, making it blind to outlier columns
- **Observation:**
  - Outliers are **extremely sparse** — only **~0.1%** of all feature dimensions
  - More importantly, they are **systematic**: outliers consistently appear at the same hidden dimensions h across almost all sequence positions s

Mixed-precision Decomposition

- **Solution:** Split by **Dimension**
  - Since outlier locations are fixed and identifiable, decompose the matrix multiplication column-wise
  - Outlier dimensions → FP16, preserving precision
  - Remaining → 99.9% of dimensions → INT8, saving memory

$$C_{f16} \approx \sum_{h \in O} X_{f16}^{h} W_{f16}^{h} + S_{f16} \cdot \sum_{h \notin O} X_{i8}^{h} W_{i8}^{h}$$

$$O = \{i | i \in \mathbb{Z}, 0 \leq i \leq h\}$$

which contains all dimensions of h which have **at least one outlier** with a magnitude larger than the threshold α. (α=6.0 is sufficient to reduce transformer performance degradation close to zero)

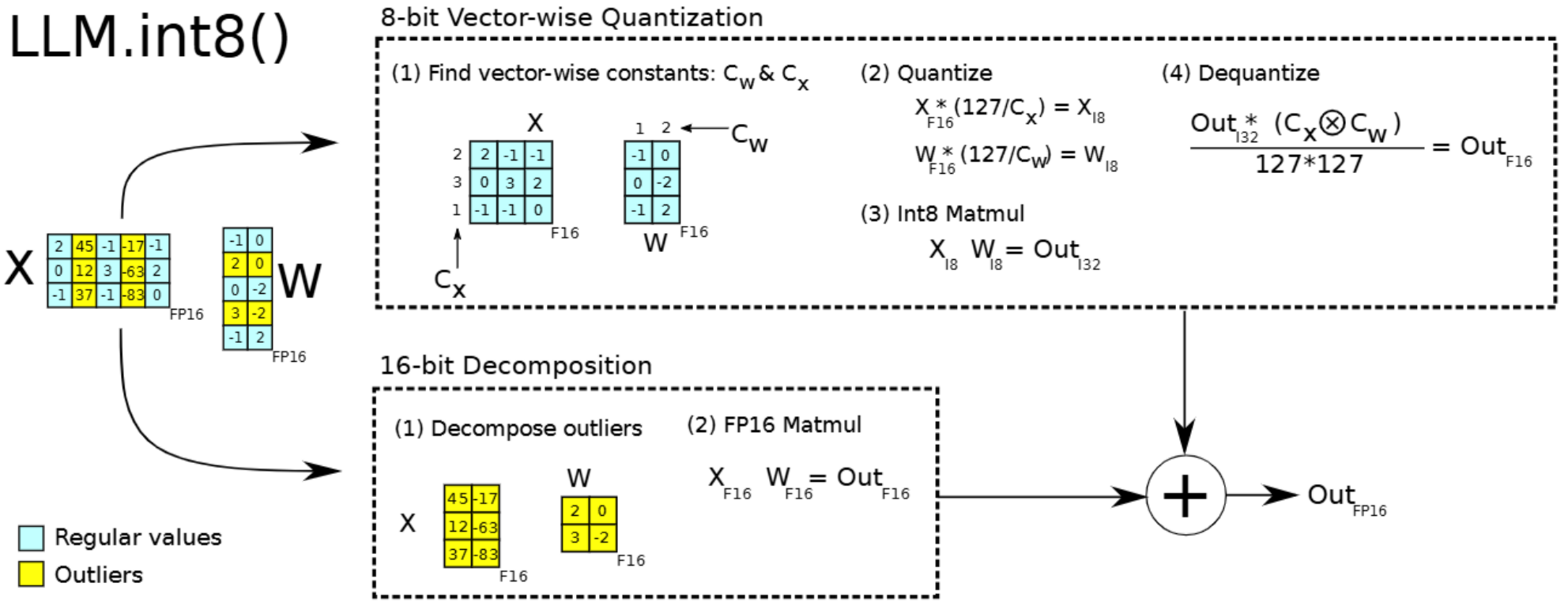


Figure 2: Schematic of LLM.int8(). Given 16-bit floating-point inputs $\mathbf{X}_{f16}$ and weights $\mathbf{W}_{f16}$, the features and weights are decomposed into sub-matrices of large magnitude features and other values. The outlier feature matrices are multiplied in 16-bit. All other values are multiplied in 8-bit. We perform 8-bit vector-wise multiplication by scaling by row and column-wise absolute maximum of $\mathbf{C}_x$ and $\mathbf{C}_w$ and then quantizing the outputs to Int8. The Int32 matrix multiplication outputs $\mathbf{Out}_{i32}$ are dequantization by the outer product of the normalization constants $\mathbf{C}_x \otimes \mathbf{C}_w$. Finally, both outlier and regular outputs are accumulated in 16-bit floating point outputs.

Experiment Setup

- Models & Scale
  - **Language modeling:** dense autoregressive transformers pretrained in fairseq, ranging from 125M to 13B parameters
  - **Zero-shot evaluation:** OPT models ranging from 125M to 175B parameters
- Evaluation Metrics
  - Language modeling: **perplexity** on C4 corpus validation set
  - Zero-shot: EleutherAI language model evaluation **harness**

LLM.int8() is the only quantization method that matches 32-bit float perplexity across all model scales up to 13B parameters

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.

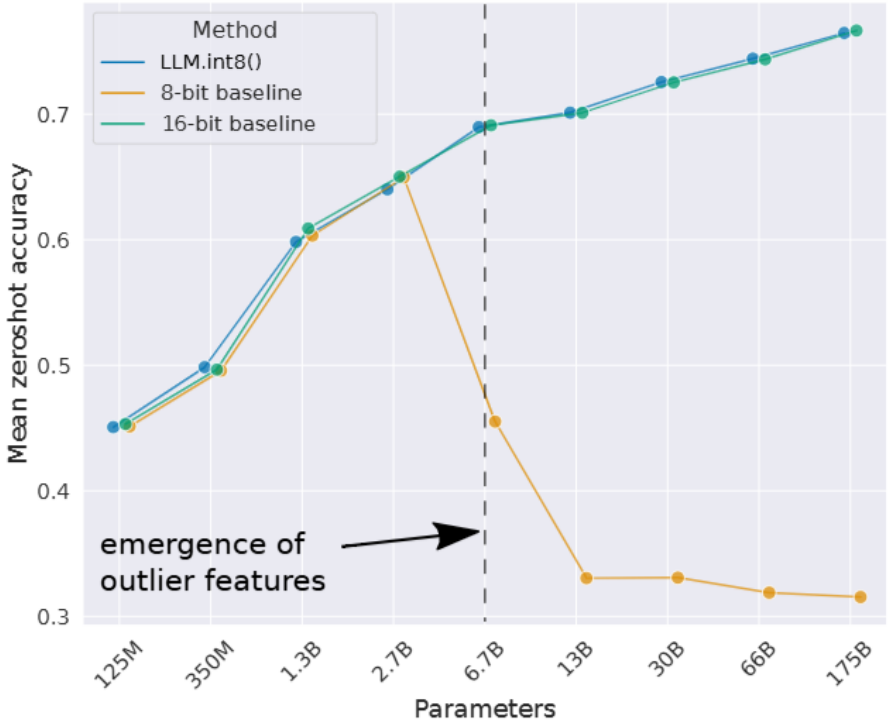
Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	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**

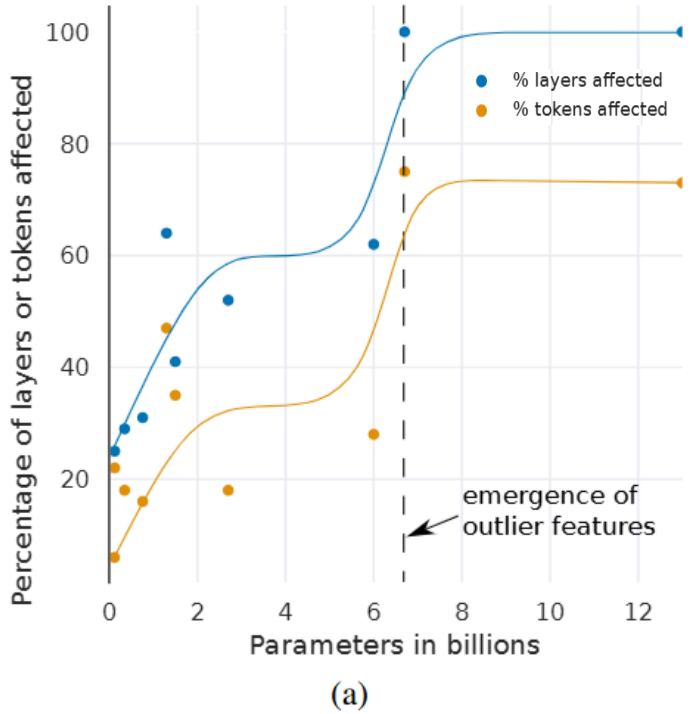
Finding Outlier Features

- Outlier Definition Criteria
  - 1. Feature magnitude ≥ 6.0
    - Perplexity **degradation stops** if we treat any feature with a magnitude **6 or larger** as an outlier feature.
  - 2. Affects at least 25% of transformer layers
    - The threshold is chosen **empirically** to capture only systematic outliers: in the 125M model, the most common outlier spans 25%+ of layers while the second most common appears in only 2%, confirming a clean separation between true outliers and incidental noise.
  - 3. Affects at least 6% of the sequence dimensions.

Measuring the Effect of Outlier Features

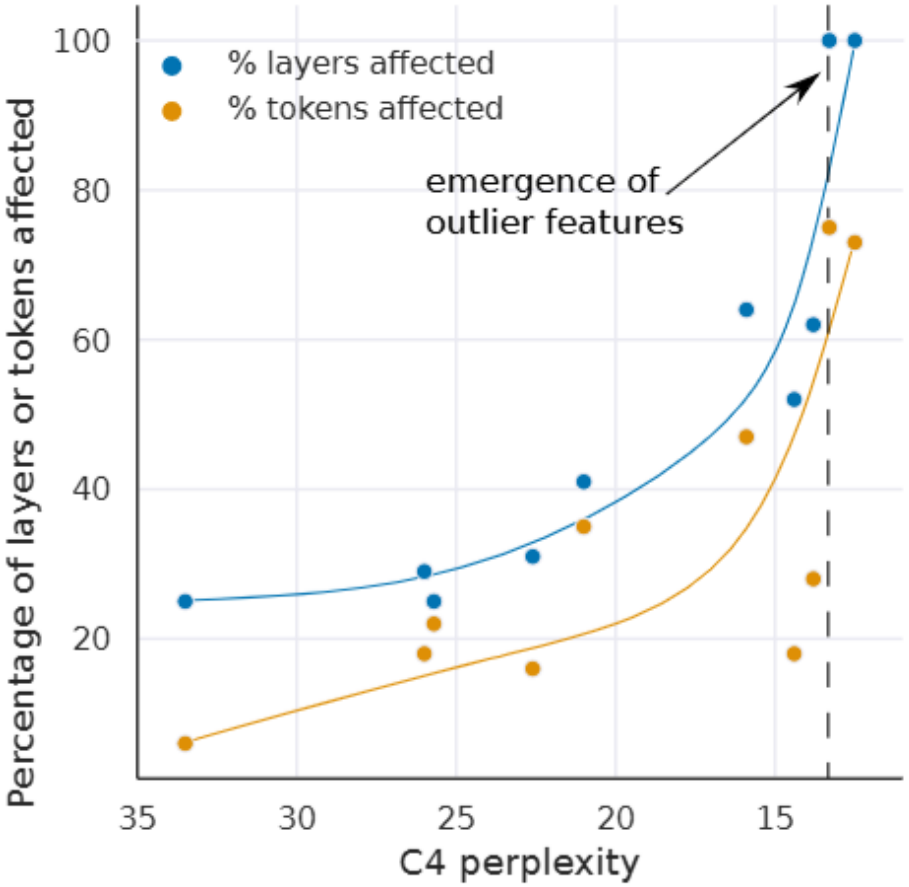
- Large magnitude outlier features emerge **suddenly** between 6B and 6.7B parameters, jumping from affecting 65% to 100% of layers — precisely where Int8 quantization begins to fail.





Measuring the Effect of Outlier Features

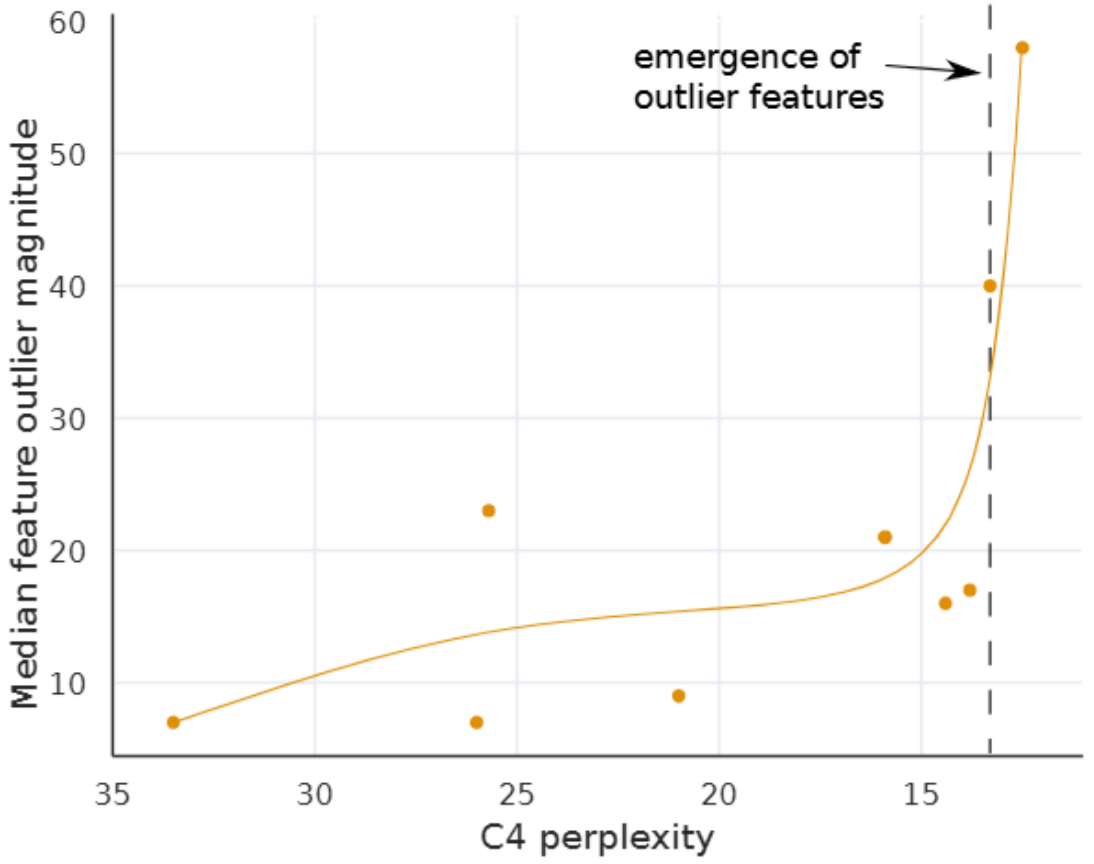
- When viewed through the lens of perplexity rather than parameter count, outlier feature emergence follows a **smooth exponential curve**, suggesting emergence is driven by perplexity (a proxy for data quantity and quality) rather than model size alone.



(b)

Measuring the Effect of Outlier Features

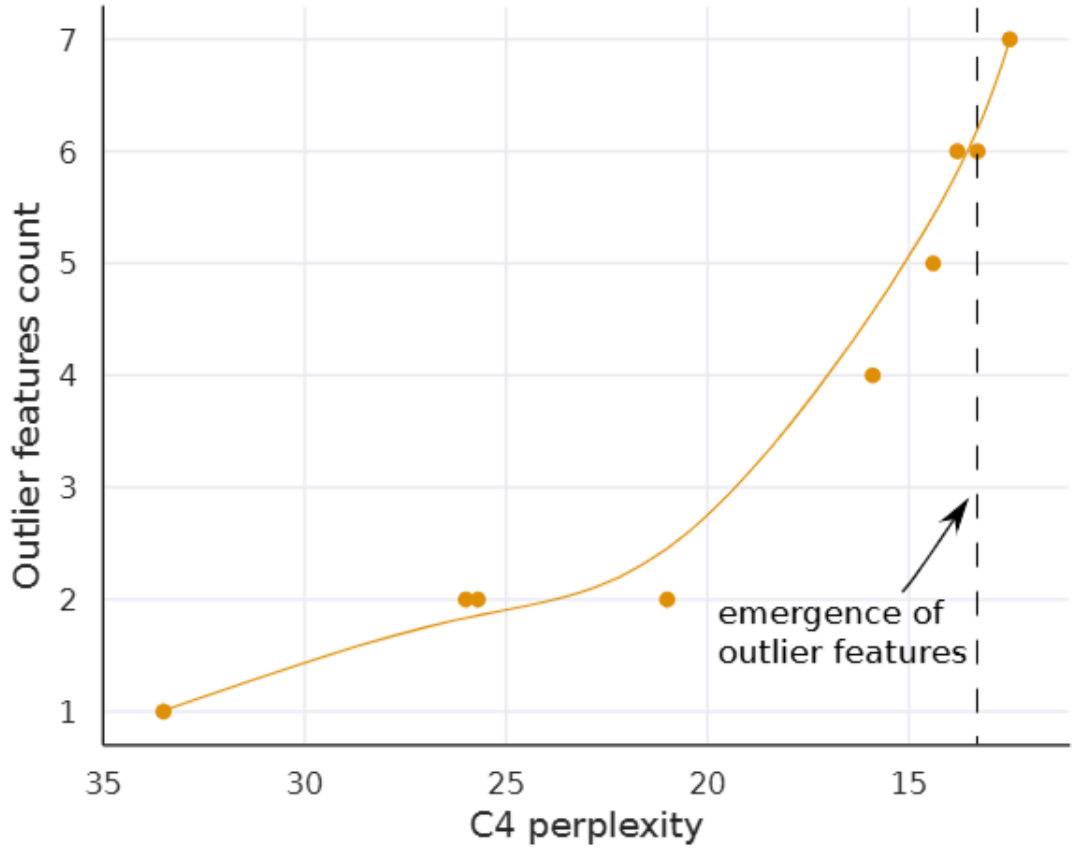
- Median outlier feature magnitude rapidly increases once outlier features occur in all layers of the transformer
- Once outlier features saturate all layers, their extreme magnitudes **overwhelm** the Int8 quantization range, collapsing most quantization bins and effectively zeroing out small values.



(a)

Measuring the Effect of Outlier Features

- Outlier feature count tracks monotonically with **perplexity** but not with model size, confirming that perplexity — not parameter count — is the true predictor of the quantization phase shift.



(b)

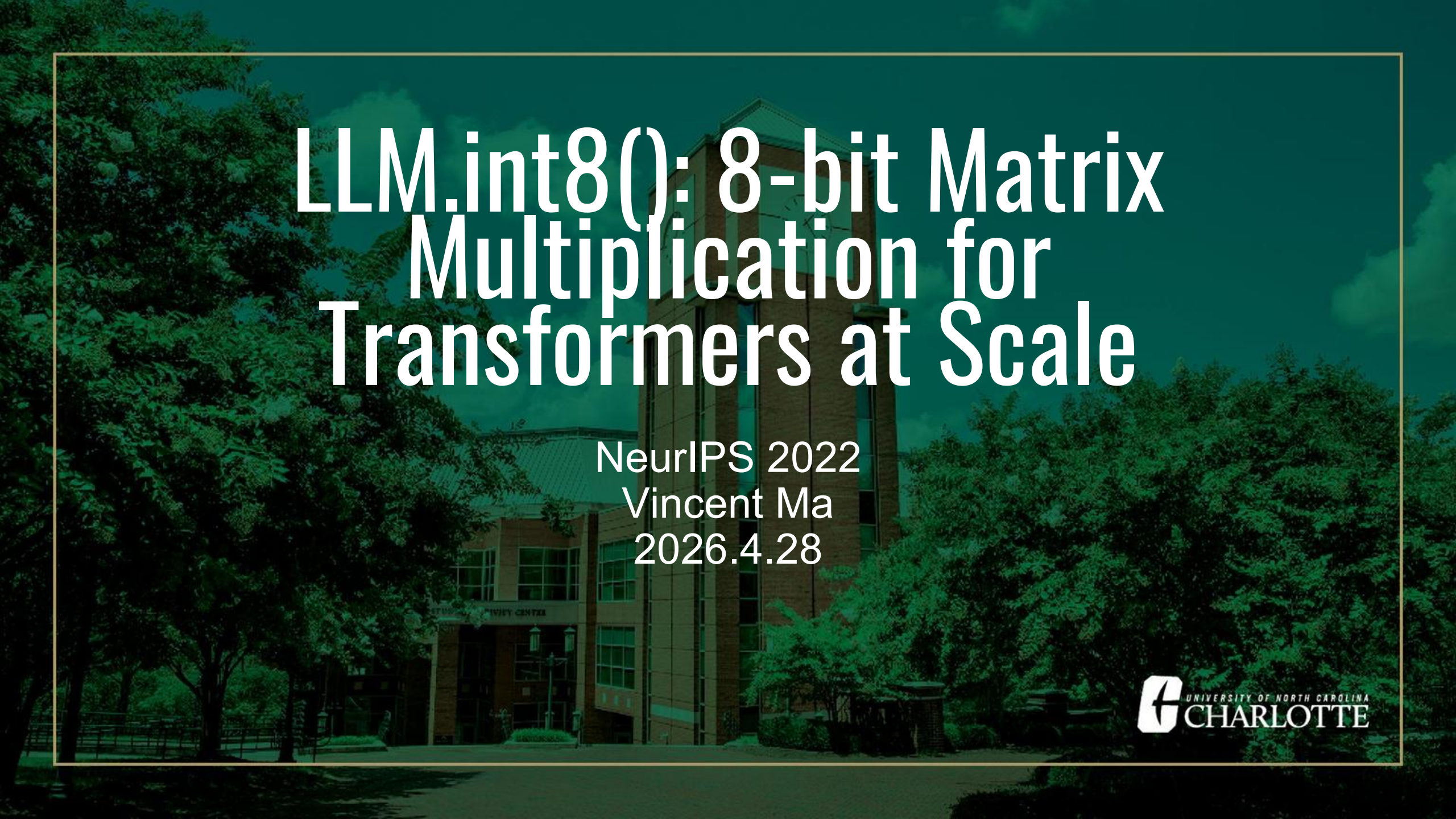
Interpretation of Quantization Performance

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**

Discussions and Limitations

- Our analysis is solely on the Int8 data type, and we do not study 8-bit floating-point (FP8) data types.
- We only study models with up to 175B parameters
- We do not use Int8 multiplication for the attention function.
- We focus on inference but do not study training or finetuning.



LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

NeurIPS 2022

Vincent Ma

2026.4.28

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Table 3: Different hardware setups and which methods can be run in 16-bit vs. 8-bit precision. We can see that our 8-bit method makes many models accessible that were not accessible before, in particular, OPT-175B/BLOOM.

			Largest Model that can be run	
Class	Hardware	GPU Memory	8-bit	16-bit
Enterprise	8x A100	80 GB	**OPT-175B / BLOOM**	**OPT-175B / BLOOM**
Enterprise	8x A100	40 GB	**OPT-175B / BLOOM**	OPT-66B
Academic server	8x RTX 3090	24 GB	**OPT-175B / BLOOM**	OPT-66B
Academic desktop	4x RTX 3090	24 GB	**OPT-66B**	OPT-30B
Paid Cloud	Colab Pro	15 GB	**OPT-13B**	GPT-J-6B
Free Cloud	Colab	12 GB	**T0/T5-11B**	GPT-2 1.3B