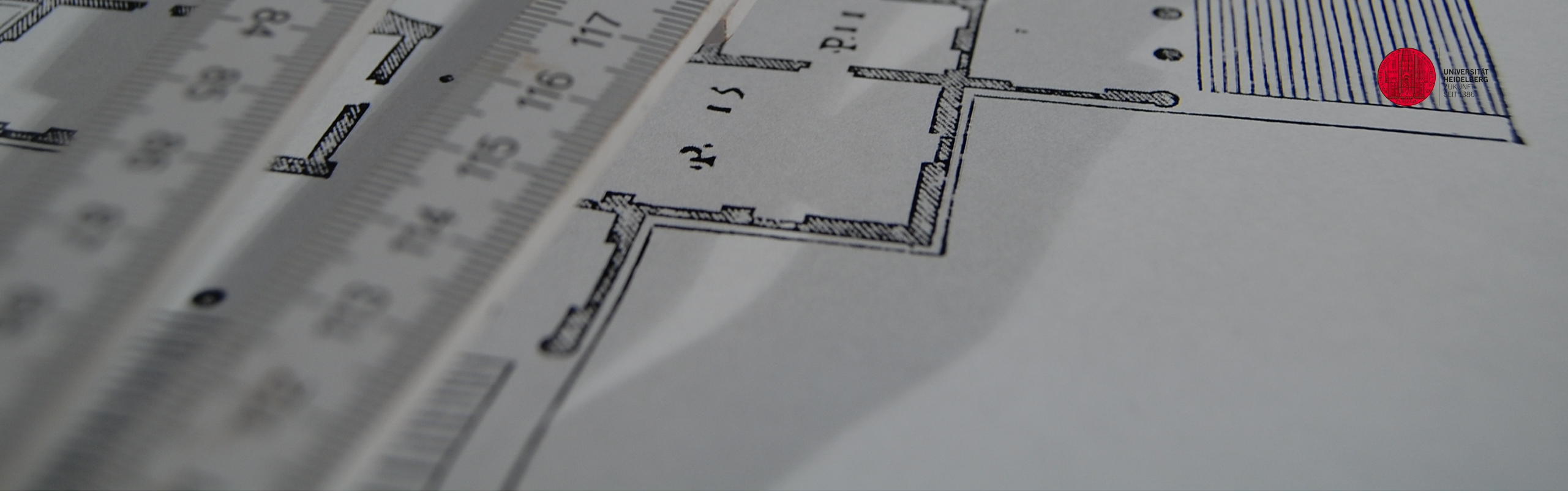


LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

Presenter: Zhaokun Wang | June 11, 2025

Content

LLMs are Powerful but Too Large

- LLMs like GPT-3, PaLM, and OPT have revolutionized NLP:
  - Zero-shot reasoning
  - Translation
  - Creative writing
- But at high costs:
  - Hundreds of billions of parameters
  - Massive GPU memory and compute

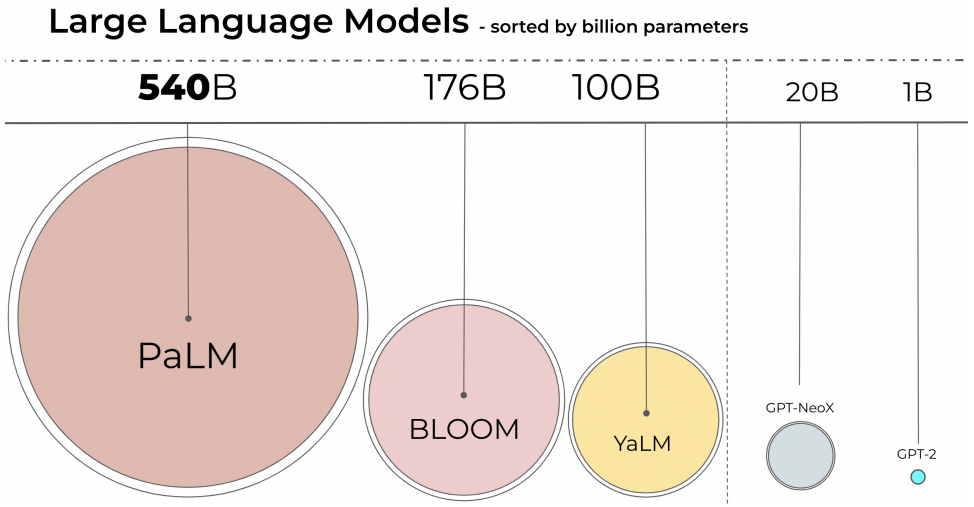


Figure 1: Illustration of why efficient inference matters.

Our Goal: Democratizing LLMs with Quantization

The Core Question:

How can we run these giant models on commodity hardware?

The Key Idea: Quantization

A technique to dramatically shrink models by converting their parameters from a high-precision, large-footprint format to a low-precision, small-footprint one.

FP32 $\rightarrow$ **INT8**

Numerical Representation Basics

- LLMs parameters typically use floating-point (FP32) representation:
  - FP32 consists of:
    - Sign bit
    - Exponent
    - Mantissa (fraction)

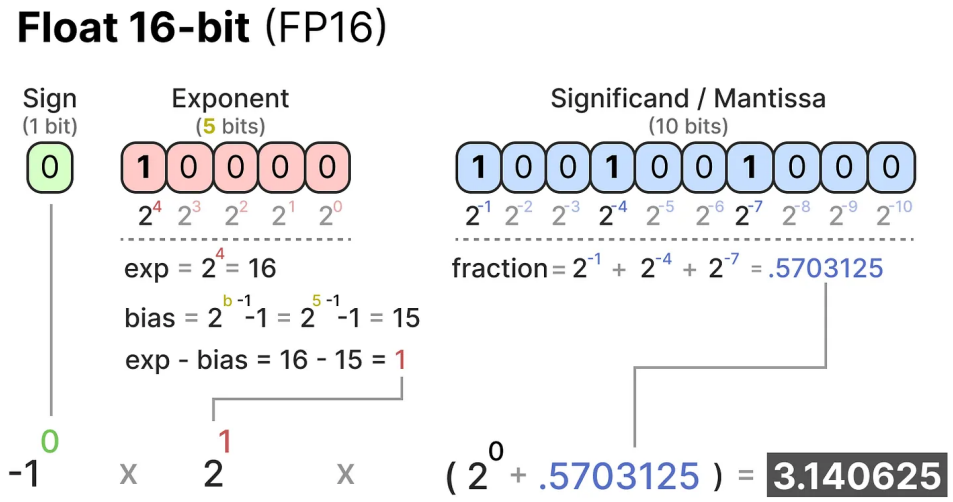


Figure 2: Floating-point representation basics.

Numerical Representation Basics

- LLMs parameters typically use floating-point (FP32) representation:
  - FP32 consists of:
    - Sign bit
    - Exponent
    - Mantissa (fraction)
  - Higher bit-width = greater precision & dynamic range but increased memory use:
    - FP32 provides high precision and dynamic range.
    - FP16 reduces memory by half at the cost of precision.

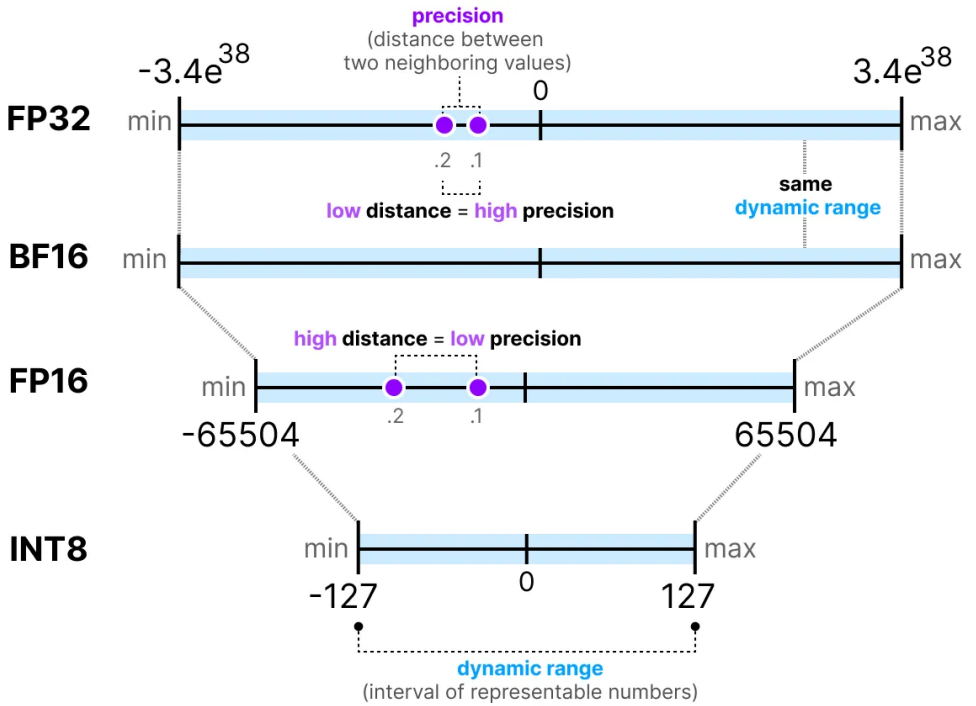


Figure 3: Floating-point representation basics.

Introduction to Quantization

- Quantization reduces precision of model parameters from FP32 to lower bit-width representations:
  - FP32 $\rightarrow$ INT8 (only 8 bits, 256 discrete values)
- Benefits:
  - Reduces memory, bandwidth, latency, and compute.
  - Small precision loss typically occurs but is minimized by careful methods.

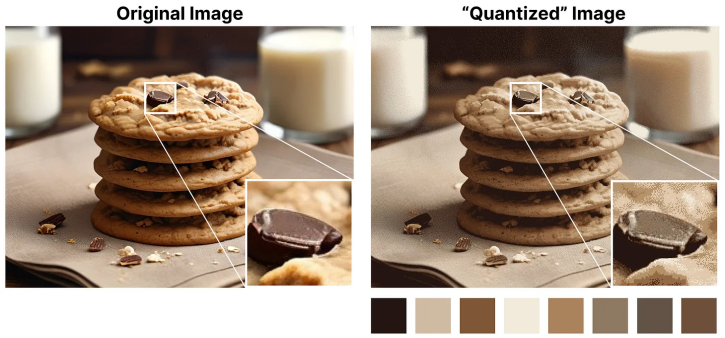


Figure 4: Quantization process illustration.

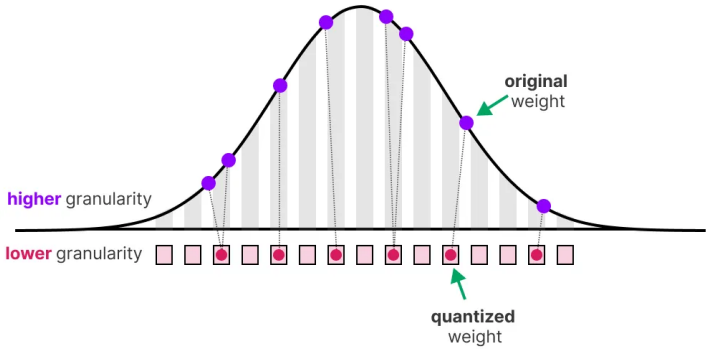


Figure 5: Distribution comparison before and after quantization.

How It Works: From Floating Points to Integers

FP32 (Floating Point)

- High precision & dynamic range.
- Composed of sign, exponent, and mantissa.
- **Large Memory Footprint.**

INT8 (Integer)

- Represents only 256 discrete values.
- Requires mapping via **Scale** and **Zero-Point**.
- **Small Memory Footprint.**

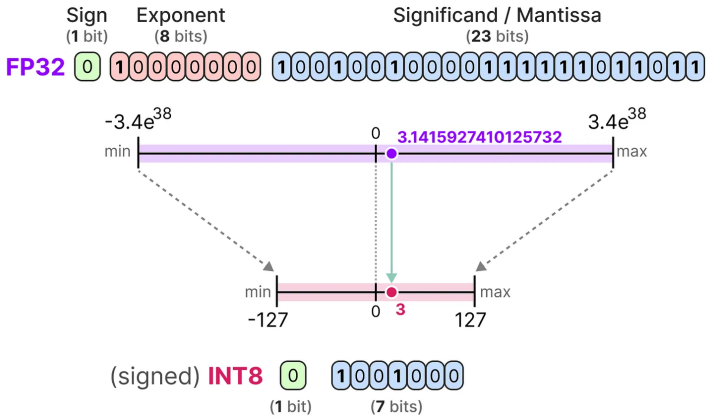


Figure 6: Quantization process details.

Absmax Quantization

This is the simplest method, scaling values symmetrically around zero.

How it Works

1. Find the maximum absolute value (α).
2. Compute scale: $s = \alpha/127$.
3. Quantize: $q = \text{round}(x/s)$.

Pros & Cons

Simple and fast.

Very sensitive to outliers. A single large value degrades precision for all others.

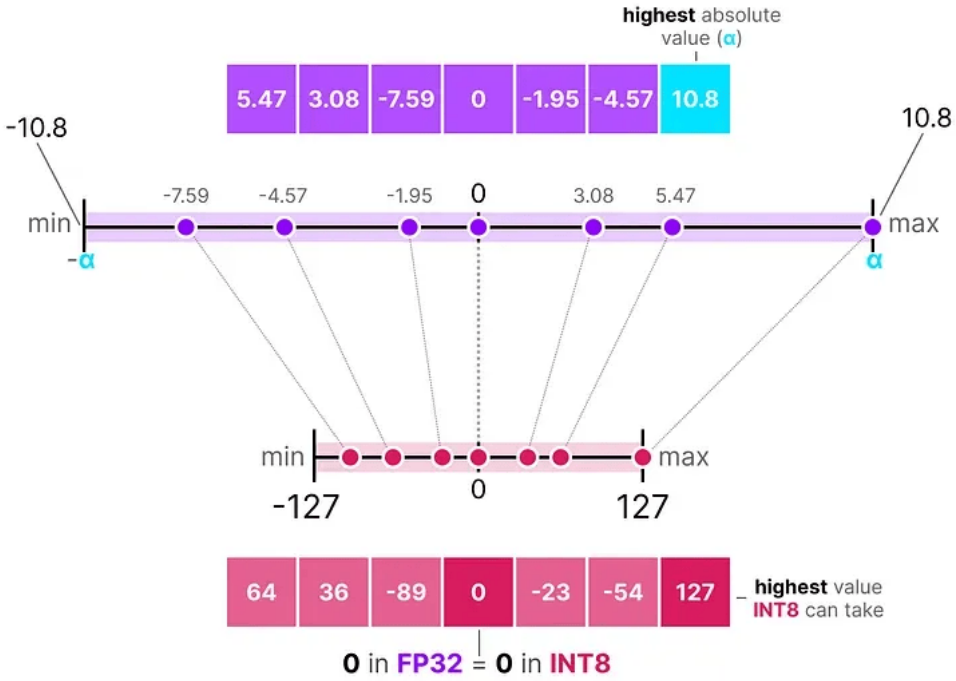


Figure 7: Illustration of absmax quantization.

Zeropoint Quantization

This method shifts the range to better utilize the 256 available integer values.

How it Works

1. Find the `min` and `max` values.
2. Compute scale: $s = (\max - \min)/255$.
3. Compute `zero_point` to map the original zero correctly.

Pros & Cons

More efficient for asymmetric data (e.g., after ReLU).

Computationally more expensive.

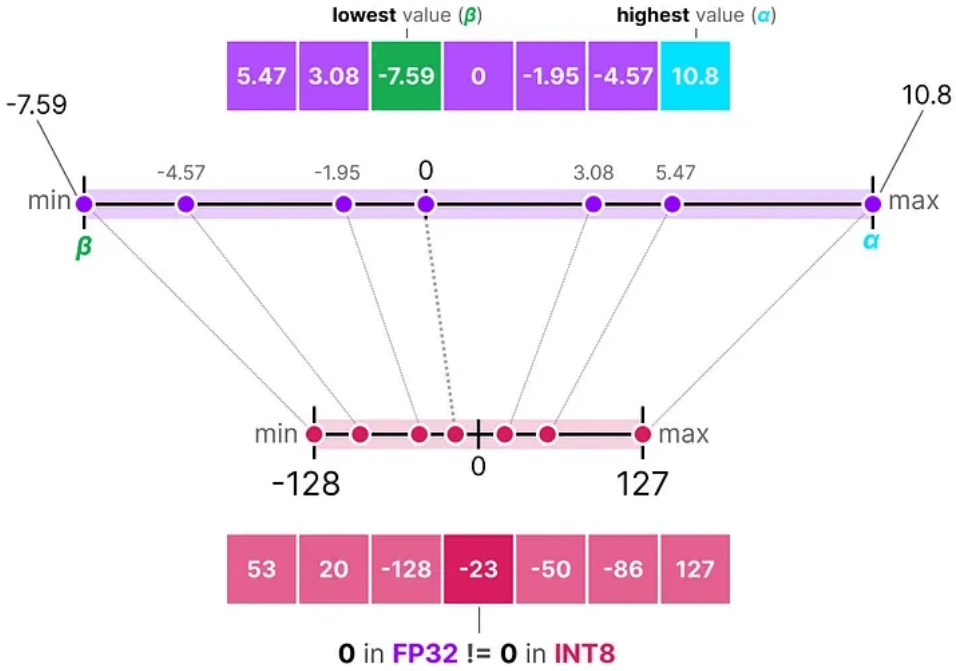


Figure 8: Illustration of zeropoint quantization.

Why Standard Quantization Fails for Large Models

As models exceed ∼6 billion parameters, a new problem emerges.

The Problem: Emergence of Strong Outliers

A tiny fraction of feature dimensions ($< 0.1\%$) have values that are **orders of magnitude larger** than all others. These are not noise; they are critical for model performance.

The Consequence: Standard quantization is completely

thrown off by these outliers. To accommodate them, the ‘scale‘ becomes huge, forcing all normal values to be quantized to near-zero, **destroying model accuracy**.

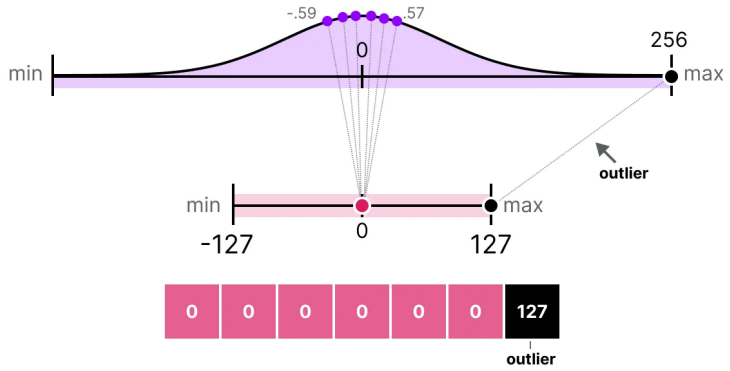


Figure 9: A few feature dimensions have extreme magnitudes.

Introducing LLM.int8(): A Hybrid, Two-Part Solution

Our approach discards the "one-size-fits-all" strategy. LLM.int8() is a hybrid method designed specifically for the outlier phenomenon in large Transformers.

Innovation 1: Vector-wise Quantization

Instead of one scale for the whole tensor, use fine-grained scales. This **localizes** the impact of outliers.

Innovation 2: Mixed-Precision Decomposition

Don’t quantize the extreme outliers at all. Process them in high precision, and everything else in low precision.

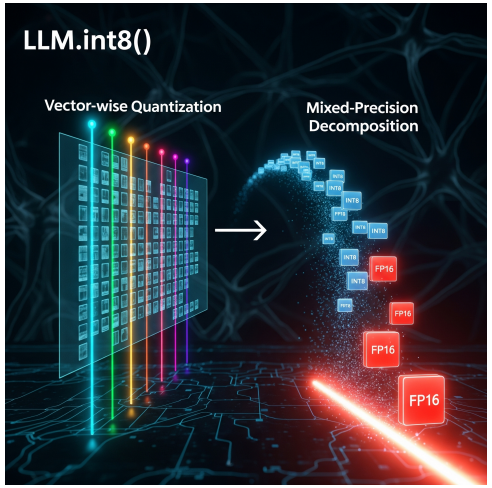


Figure 10: LLM.int8() combines two key ideas to preserve accuracy.

Part 1: Vector-wise Quantization

The Idea

Global scaling is too coarse. An outlier in one column shouldn't ruin the precision for all other columns.

The Technique:

- In a matrix multiplication, we calculate a separate 'scale' factor for each column of the weight matrix and each row of the input matrix.

The Benefit:

- Outlier effects are contained. An outlier in one vector only affects the scale for that specific vector, preserving the precision of all other data.

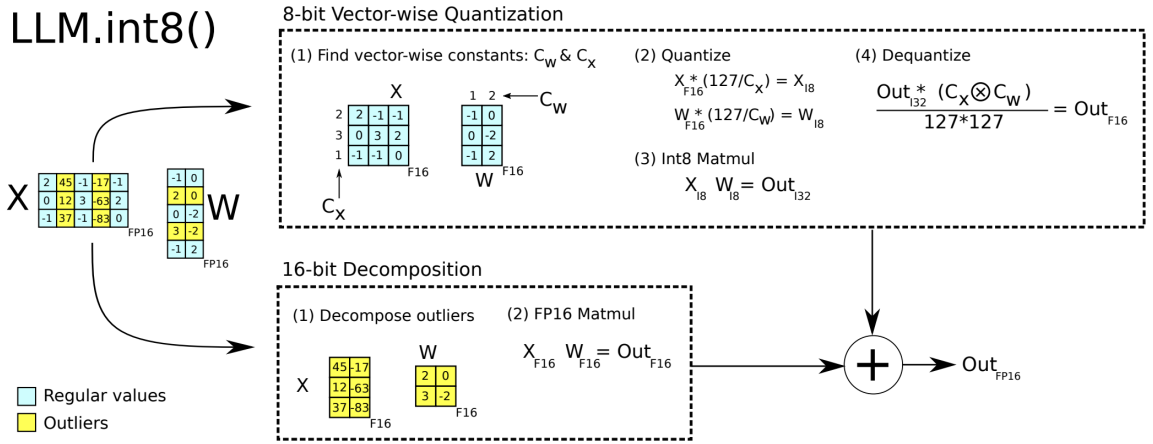


Figure 11: A separate scale for each vector localizes quantization errors.

Part 2: Mixed-Precision Decomposition

The Idea

For the most extreme outliers, even fine-grained quantization is not enough. The solution is to **not quantize them at all**.

The Technique:

1. **Detect:** Identify the few dimensions that contain systematic outliers.
2. **Decompose:** Split the matrix multiplication:
   - The vast majority (99.9%) is multiplied using efficient **INT8**.
   - The tiny fraction of outlier dimensions (0.1%) are multiplied in full **FP16** precision.
3. **Combine:** The results are added together.

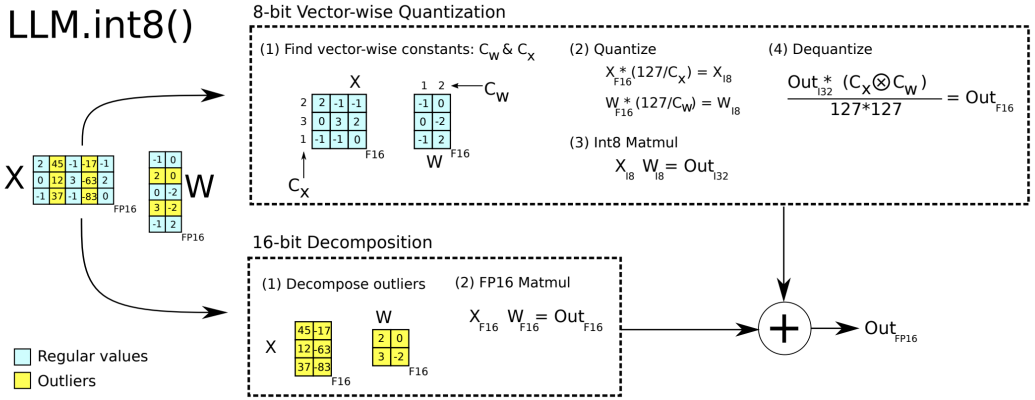


Figure 12: Isolating outliers in FP16 preserves their critical information.

The Complete Picture: How LLM.int8() Works

LLM.int8() seamlessly integrates these two techniques to quantize 175B+ parameter models with **zero performance degradation**.

The LLM.int8() Pipeline

1. Input Hidden States (X) arrive.
2. **Outlier Detection:** The 0.1% of outlier feature dimensions in X are identified.
3. **Decomposition:**
   - The "outlier" part of X is multiplied by its corresponding weights in **FP16**.
   - The "normal" part of X is quantized vector-wise and multiplied by the vector-wise quantized weights in **INT8**.
4. Results are added to produce the final, accurate output.

Fun Analogy: Quantization is like Packing for a Trip

- Absmax quantization: Stuff all clothes into a single-size bag — the bulkiest coat decides the size.
- Zeropoint: Fold and shift everything neatly, still same bag.
- Vector-wise: Pack different types of items in their own bags — one for shirts, one for shoes.
- Mixed-precision: Keep the formal suit in a garment bag (full-size), everything else goes in compressible packing cubes.

Figure 13: Analogy of quantization methods to packing strategies.

Experiments Overview

- **Objective:** Validate LLM.int8()
- **Questions Addressed:**
  - Can LLM.int8() prevent performance collapse typical in traditional 8-bit methods?
  - Does robustness scale with model size?
- Benchmarked extensively against strong baselines across model scales.

Experimental Setup

Evaluation Strategies:

- **Language Modeling (Perplexity)**
  - Dataset: C4
  - Lower perplexity indicates better performance.
  - Sensitive to quantization.
  - Models: 125M–13B parameters.
- **Zero-shot Downstream Tasks**
  - Tasks: WinoGrande, HellaSwag, PIQA, LAMBADA
  - Evaluated via EleutherAI harness.
  - Reflects practical application performance.
  - Models: 125M–175B parameters.

Figure 14: Language Modeling Setup

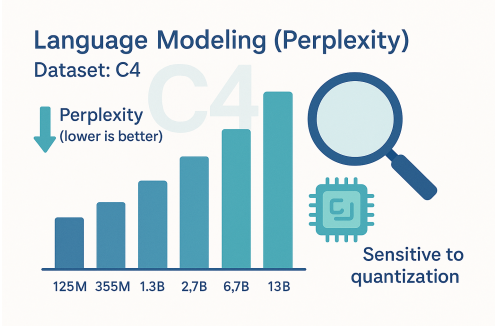
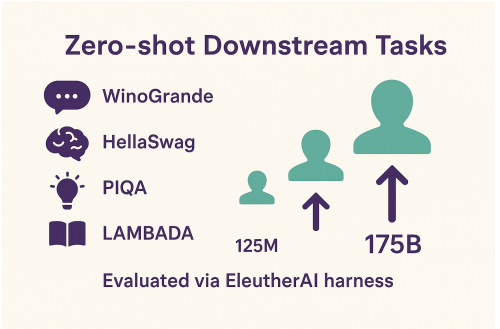


Figure 15: Zero-shot Setup



Perplexity Results

- Standard Int8 significantly degrades at larger scales (>6.7B).
- **LLM.int8() retains performance at all scales**.

Conclusion: LLM.int8 uniquely maintains perplexity performance.

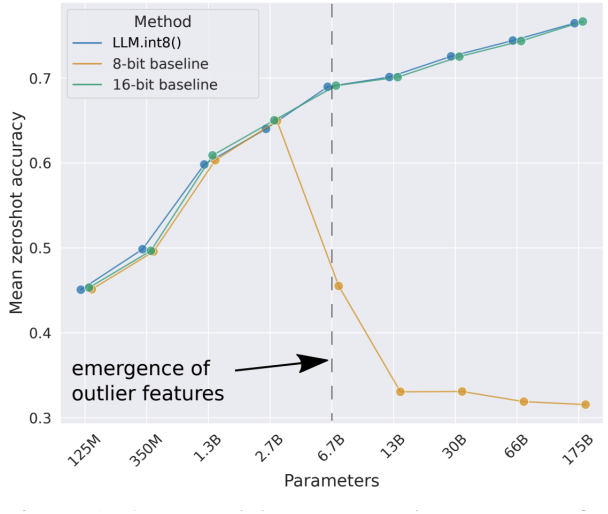
Table 1: Perplexity Results

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**

Zero-shot Task Performance

- **Metrics:** Mean zero-shot accuracy
- Standard Int8 deteriorates significantly after 6.7B parameters.
- LLM.int8() consistently matches FP16 performance up to 175B.
- Phase transition at 6.7B highlights the superiority of LLM.int8.

Figure 16: Zero-shot Task Performance



Inference Speed and Latency

- **Memory Efficiency:**
  - Approximately 2x reduction.
  - OPT-175B and BLOOM-176B feasible on a single server.

Table 2: Inference Speed and Latency

			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

Inference Speed and Latency

- **Memory Efficiency:**
  - Approximately 2x reduction.
  - OPT-175B and BLOOM-176B feasible on a single server.
- **Speed Improvements:**
  - Small models (<6.7B): Minimal impact.
  - Large models (13B): Int8 matrix multiplication 1.2x faster.
  - Latency per token matches FP16 at large batch sizes.

Table 3: Inference Speed and Latency

GPT-3 Size	Small	Medium	Large	XL	2.7B	6.7B	13B	175B
Model dimension	768	1024	1536	2048	2560	4096	5140	12288
FP16-bit baseline	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x
Int8 without overhead	0.99x	1.08x	1.43x	1.61x	1.63x	1.67x	2.13x	2.29x
Absmax PyTorch+NVIDIA	0.25x	0.24x	0.36x	0.45x	0.53x	0.70x	0.96x	1.50x
Vector-wise PyTorch+NVIDIA	0.21x	0.22x	0.33x	0.41x	0.50x	0.65x	0.91x	1.50x
Vector-wise	**0.43x**	**0.49x**	**0.74x**	**0.91x**	**0.94x**	**1.18x**	**1.59x**	**2.00x**
LLM.int8() (vector-wise+decomp)	0.14x	0.20x	0.36x	0.51x	0.64x	0.86x	1.22x	1.81x

Inference Speed and Latency

- **Memory Efficiency:**
  - Approximately 1.2x reduction.
  - OPT-175B and BLOOM-176B feasible on a single server.
- **Speed Improvements:**
  - Small models (<6.7B): Minimal impact.
  - Large models (13B): Int8 matrix multiplication 1.2x faster.
  - Latency per token matches FP16 at large batch sizes.

Table 4: Inference Speed and Latency

Batch Size	Hardware	1	8	32
bfloat16 baseline	8xA100 80GB	**239**	**32**	9.94
LLM.int8()	8xA100 80GB	253	34	10.44
LLM.int8()	4xA100 80GB	246	33	9.40
LLM.int8()	3xA100 80GB	247	33	**9.11**

Emergent Outlier Behavior

- Outliers become significant at 6.7B parameters:
  - 0.1% features represent 20% of softmax mass.
  - Removing outliers drastically reduces accuracy and increases perplexity.
- **Conclusion:** Handling outliers critical for maintaining performance.

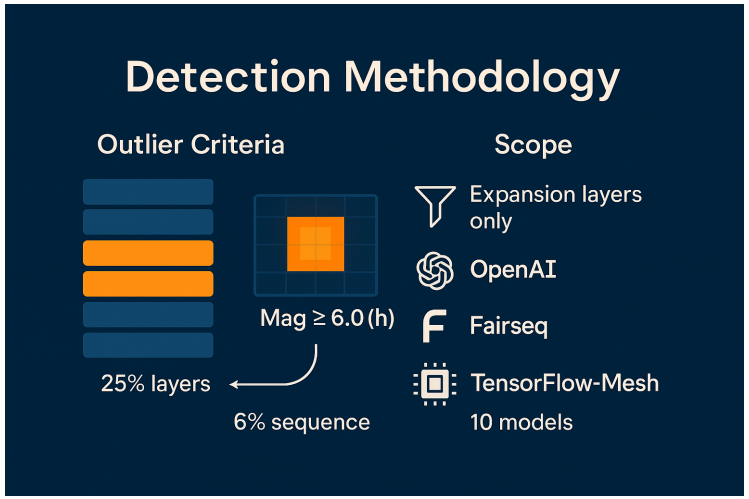
Table 5: Emergent Outlier Behavior

			Outliers		Frequency			Top-1 softmax p	
Model	PPL↓	Params	Count	1-sided	Layers	SDims	Quartiles	w/ Outlier	No Outlier
GPT2	33.5	117M	1	1	25%	6%	(-8, -7, -6)	45%	19%
GPT2	26.0	345M	2	1	29%	18%	(6, 7, 8)	45%	19%
FSEQ	25.7	125M	2	2	25%	22%	(-40, -23, -11)	32%	24%
GPT2	22.6	762M	2	0	31%	16%	(-9, -6, 9)	41%	18%
GPT2	21.0	1.5B	2	1	41%	35%	(-11, -9, -7)	41%	25%
FSEQ	15.9	1.3B	4	3	64%	47%	(-33, -21, -11)	39%	15%
FSEQ	14.4	2.7B	5	5	52%	18%	(-25, -16, -9)	45%	13%
GPT-J	13.8	6.0B	6	6	62%	28%	(-21, -17, -14)	55%	10%
FSEQ	13.3	6.7B	6	6	100%	75%	(-44, -40, -35)	35%	13%
FSEQ	12.5	13B	7	6	100%	73%	(-63, -58, -45)	37%	16%

Detection Methodology

- **Outlier Criteria:**
  - Magnitude 6.0 in feature dimension h.
  - Present in 25% of layers (systematic emergence).
  - Affects 6% of sequence dimensions (s).
- **Scope:**
  - Analyzed attention/FFN expansion layers (ignored contraction layers).
  - Validated across 3 frameworks (OpenAI, Fairseq, TensorFlow-Mesh) and 10 models (125M–13B params).

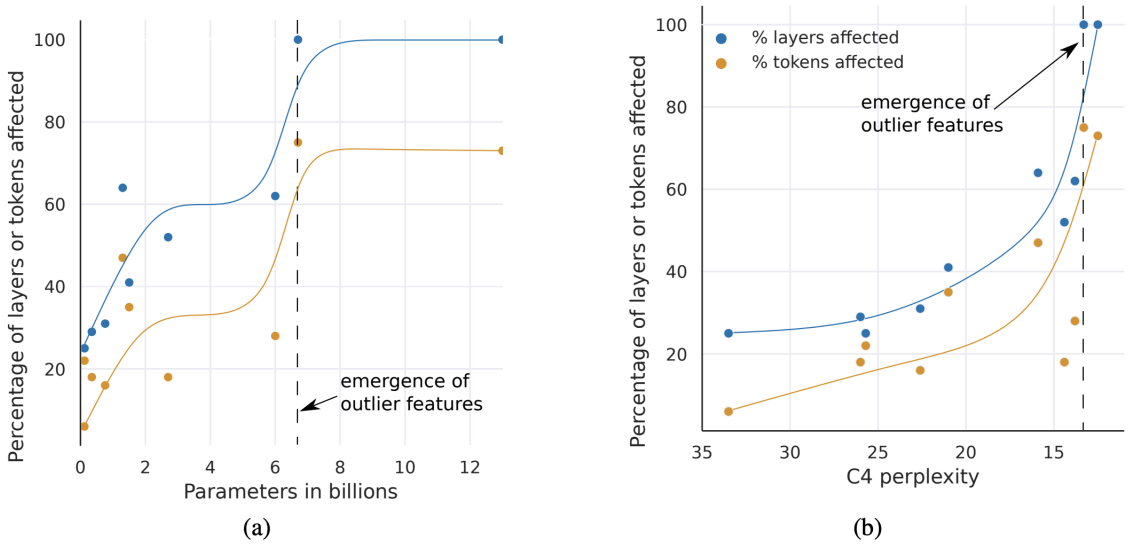
Figure 17: Detection Methodology



Emergence Patterns

- **By parameter count:** sudden phase shift between 6 B $\rightarrow$ 6.7 B
  - Layers affected: 65 % $\rightarrow$ 100 %
  - Tokens affected: 35 % $\rightarrow$ 75 %
- **By perplexity (PPL):** smooth exponential growth—outliers rise steadily as PPL falls
  - Implies emergence is performance-driven, not size-driven alone
- **Result:** onset of quantization failure aligns with this phase shift

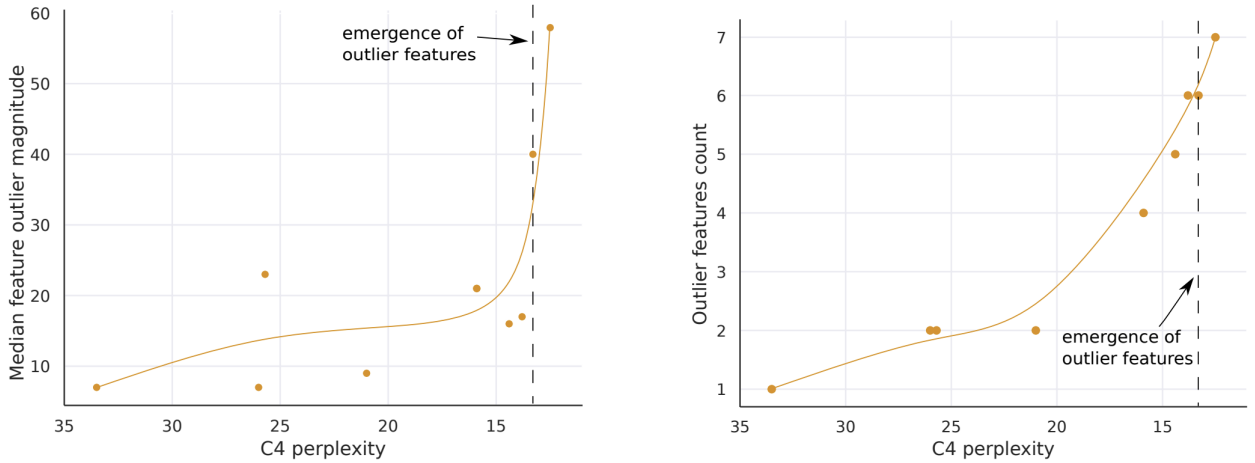
Figure 18: Emergence Patterns



Magnitude & Frequency Dynamics

- **Median outlier magnitude jumps sharply once every layer hosts outliers**
  - 6.7 B model: median |x| 40
  - 13 B model: median |x| 60 – 65
- **Count trend:** strictly monotonic w.r.t. decreasing PPL, non-monotonic w.r.t. size
- **Despite** 150 k outlier activations per 2048-token pass (13 B), they occupy 7 hidden dims

Figure 19: Magnitude & Frequency Dynamics



Impact on Prediction

- **Zeroing 7 outlier dims →**
  - Top-1 softmax p: 40 % → 20 %
  - Validation PPL: +600 – 1000 %
- **Zeroing 7 random non-outlier dims →**
  - Top-1 softmax p: -0.02 – 0.3 %
  - PPL: +0.1 %
- **Take-away:** these few dims carry disproportional semantic load—precision here is critical.

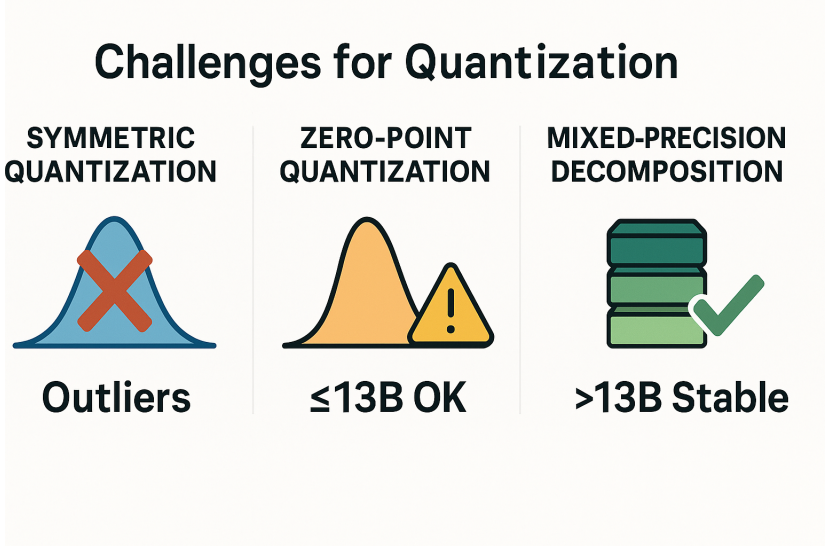
Table 6: Impact on Prediction

			Outliers		Frequency			Top-1 softmax p	
Model	PPL↓	Params	Count	1-sided	Layers	SDims	Quartiles	w/ Outlier	No Outlier
GPT2	33.5	117M	1	1	25%	6%	(-8, -7, -6)	45%	19%
GPT2	26.0	345M	2	1	29%	18%	(6, 7, 8)	45%	19%
FSEQ	25.7	125M	2	2	25%	22%	(-40, -23, -11)	32%	24%
GPT2	22.6	762M	2	0	31%	16%	(-9, -6, 9)	41%	18%
GPT2	21.0	1.5B	2	1	41%	35%	(-11, -9, -7)	41%	25%
FSEQ	15.9	1.3B	4	3	64%	47%	(-33, -21, -11)	39%	15%
FSEQ	14.4	2.7B	5	5	52%	18%	(-25, -16, -9)	45%	13%
GPT-J	13.8	6.0B	6	6	62%	28%	(-21, -17, -14)	55%	10%
FSEQ	13.3	6.7B	6	6	100%	75%	(-44, -40, -35)	35%	13%
FSEQ	12.5	13B	7	6	100%	73%	(-63, -58, -45)	37%	16%

Challenges for Quantization

- **Outliers break symmetric quantization (e.g., absmax)**
  - Result: precision loss, cascading inference failure
- **Zeropoint quantization:**
  - Handles asymmetry better by mapping to [-127, 127]
  - Works well up to 13B, then fails due to magnitude growth
- **Mixed-Precision Decomposition:**
  - Resolves failures post-13B
  - Removes need for asymmetric handling $\rightarrow$ improved stability

Figure 20: Challenges for Quantization



Small Summary

- Outliers are few but powerful — tiny subsets control performance.
- They systematically emerge with scaling and lower perplexity.
- Effective handling (e.g., via mixed-precision) is essential for:
  - Stable quantization
  - Scalable inference and training
- Critical to modern LLM design and optimization.

Core Contributions and Challenges

- **LLM.int8: The First Degradation-Free 8-bit Inference at 175B Scale**
- **Key Innovations:**
  - **Vector-wise quantization:** Fine-grained scale control for weights/activations.
  - **Mixed-precision decomposition:** Isolates outlier features in FP16 (99.9% values in INT8).
- **Limitations:**
  - **Inference-only (no training acceleration).**
  - **Attention layers remain unquantized.**
  - **No online speedup (no hardware-native INT8 compute).**

Future Directions

- **Next Frontiers in LLM Quantization:**
- **Attention Layer Quantization:**
  - Critical for end-to-end efficiency; requires new methods.
- **FP8 Adoption:**
  - Floating-point 8-bit formats for better accuracy/compatibility trade-offs.
- **8-bit Training & Finetuning:**
  - Early success with FFN layers; attention projections still challenging.
- **Extreme Memory Optimization:**
  - KV cache quantization for long sequences/large batches.

Figure 21: Future directions in LLM quantization research.

Evolution of Quantization Methods

- **Post-LLM.int8 Advancements:**
- **SmoothQuant (2022):**
  - Migrates activation outliers to weights. $\rightarrow$ Enables W8A8 & hardware acceleration.

Figure 22: Evolution of quantization methods over time.

Evolution of Quantization Methods

- **Post-LLM.int8 Advancements:**
- **AWQ (2023):**
  - Protects salient weights via activation-aware scaling. → W4A16 with 1.85× speedup.

Figure 23: Evolution of quantization methods over time.

Evolution of Quantization Methods

- **Post-LLM.int8 Advancements:**
- **GPTQ (2023):**
  - Hessian-based optimization for minimal loss. $\rightarrow$ INT8/INT4 support; 2× speedup.

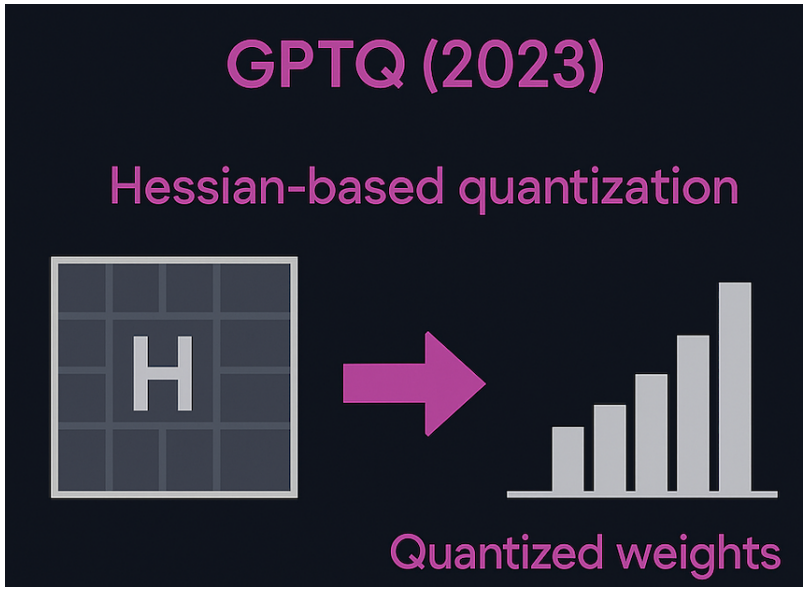


Figure 24: Evolution of quantization methods over time.

Practical Impact & Conclusion

- **Democratizing Large-Scale Inference:**
  - Enables 175B models on consumer-grade GPUs (e.g., OPT-175B, BLOOM).
  - 2× memory reduction → Wider accessibility for academia/low-resource labs.
  - Integrated into HuggingFace, BitsAndBytes, AutoGPTQ, AutoAWQ.
- **Conclusion:**
  - Quantization is no longer a compromise. With methods like LLM.int8 and its successors, we enable real-world deployment of massive models without sacrificing performance.

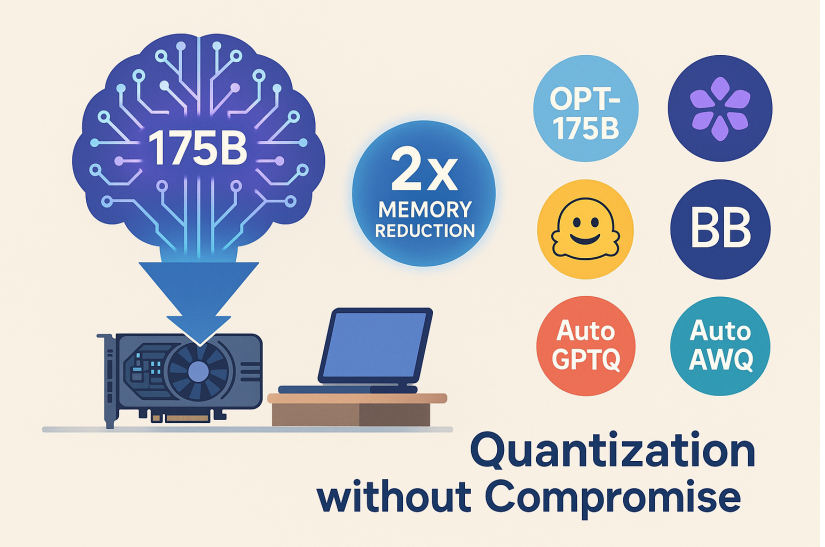


Figure 25: Practical impact and conclusion of LLM.int8.

References

[1] T. Dettmers, M. Lewis, Y. Belkada, L. Zettlemoyer (2022).
LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale.

[2] M. Grootendorst (2023).
A Visual Guide to Quantization.

[3] Y. Belkada T. Dettmers (2022).
A Gentle Introduction to 8-bit Matrix Multiplication...

[4] G. Xiao, J. Lin, et al. (2022).
SmoothQuant: Accurate and Efficient Post-Training Quantization...

[5] J. Lin, J. Tang, et al. (2023).
AWQ: Activation-aware Weight Quantization...

[6] E. Frantar, S. Ashkboos, et al. (2023).
GPTQ: Accurate Post-Training Quantization...

Image Table Attribution

- **Tables 1-6:** Adapted from Dettmers et al. [1].
- **Figure 1:** Sourced from the Hugging Face Blog [3].
- **Figures 2-9:** Adapted from Maarten's AI Newsletter [2].
- **Figures 10, 11, 12, 16:** Adapted from Dettmers et al. [1].
- **Figures 22-24:** These visuals illustrate concepts from their respective papers on SmoothQuant [4], AWQ [5], and GPTQ [6].
- **Other auxiliary images** (e.g., Figs. 13, 14, 15, 17-21, 25) were created by the presenter or generated via GPT-4o for illustrative purposes.

Q&A

Thank You! Questions?