MLOPs Meetup Helsinki

4th Feb 2026

LLM Compression:

How Far Can We Go in Balancing Size and Performance?

Dr. Shantipriya Parida
Senior AI Scientist

AMD Silo AI, Helsinki, Finland

Agenda

- Motivation & Challenges
- Compression Methods
- Use Cases
- Takeaways

Motivation and Challenges

- LLMs provide powerful capabilities but require high memory and compute
- Resource constraints hinder deployment on edge and mobile devices
- Model compression is essential for real-world applicability
- Naïve low-bit quantization degrades model accuracy
- Need for accuracy-aware, efficient compression methods

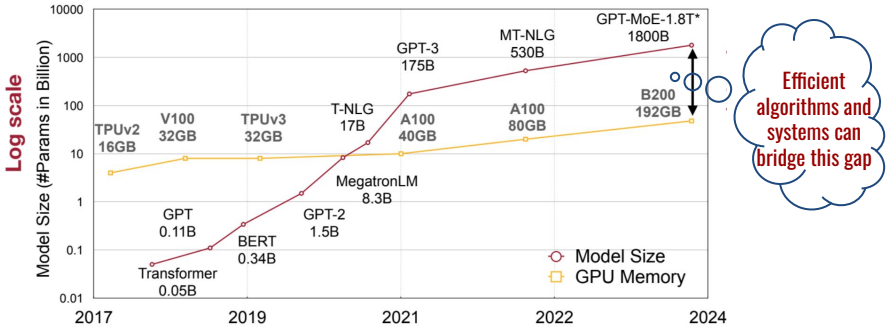


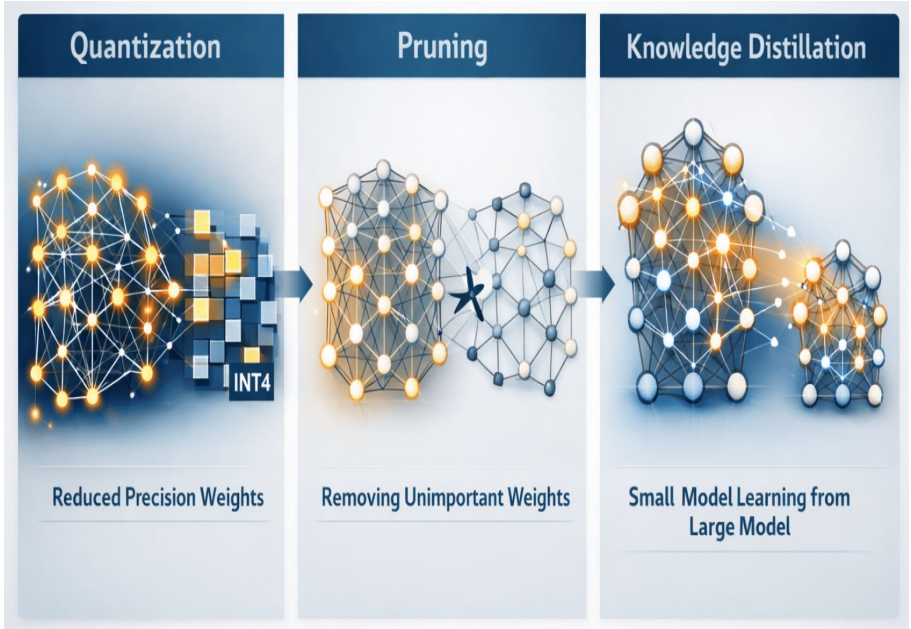
Fig 1: Exponential Growth of LLM Parameters Outpacing GPU Memory Capacity
Image source: AWQ: Activation-aware Weight Quantization

Motivation and Challenges

- **Edge Computing Applications**
  - Deploy sophisticated computer vision and NLP models directly on IoT devices and sensors, enabling real-time processing without cloud connectivity requirements.
- **Mobile AI Development**
  - Integrate complex deep learning capabilities into smartphone applications while maintaining battery efficiency and performance responsiveness.
- **Resource-Limited Research Environments**
  - Enable AI research teams with limited computational resources to work with state-of-the-art models through efficient compression techniques.

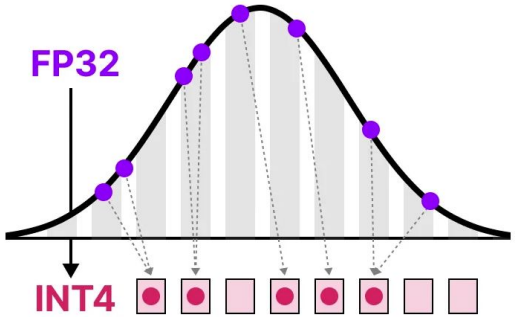
Model Compression Techniques

- **Quantization**
  - Reduce numerical precision of model weights
  - Lowers memory footprint and inference cost
  - Enables deployment on resource-constrained hardware
- **Pruning**
  - Remove redundant or less important weights
  - Preserves performance while reducing model size
  - Improves inference efficiency
- **Knowledge Distillation**
  - Train a smaller model using a large teacher model
  - Retains key behaviors of the original LLM
  - Produces compact, high-performing models



What is Quantization ?

- Quantization aims to reduce the precision of a model's parameter from higher bit-widths (like 32-bit floating point) to lower bit-widths (like 8-bit integers).
- Reduce numerical precision of model weights from 32-bit floating point to lower bit representations (8-bit, 4-bit), dramatically decreasing memory requirements while maintaining functional accuracy.



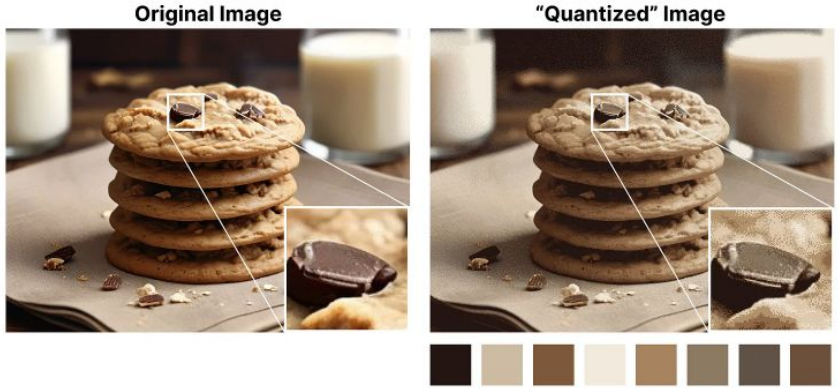


Image source:
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Why Memory Matters in LLMs

- **Model size = memory footprint**
  Number of parameters directly determines RAM/GPU memory needs (FP32/FP16 models quickly exceed hardware limits).

- **Precision defines memory efficiency**
  Moving from FP32 → FP16 → INT8 → INT4 reduces memory by **2×–8×** with minimal accuracy loss.

- **Quantization is the key enabler**
  Large models (e.g., 70B) become deployable on a **single GPU** when stored in 4-bit representations.

- **Memory vs performance trade-off**
  Larger models offer better reasoning and accuracy, but smaller or quantized models deliver **higher throughput and lower latency**.

- **Perplexity ≠ real-world quality**
  Slight increases in perplexity after compression often **do not hurt downstream task performance**.

- **Optimal choice is task-dependent**

Numerical Representations in LLMs

(IEEE-754)

Floating-Point Formats

FP32 (Single Precision – 32 bits)

- Sign: 1 bit, Exponent: 8 bits, Mantissa (Fraction): 23 bits
- High numerical precision and wide dynamic range
- High memory and compute cost

FP16 (Half Precision – 16 bits)

- Sign: 1 bit, Exponent: 5 bits, Mantissa (Fraction): 10 bits
- Lower precision and dynamic range
- Reduced memory footprint and faster computation

BF16 (BFloat16 – 16 bits)

- Sign: 1 bit | Exponent: 8 bits | Mantissa: 7 bits
- Maintains FP32-like dynamic range
- Slightly lower precision, ideal for stable training
- Widely adopted in TPU & GPU training

Integer & Low-Bit Formats (Used in LLM Compression)

INT8 (8 bits)

- Fixed-point integer (no exponent or mantissa)
- Typically uses per-tensor or per-channel scaling factors
- Significantly reduces memory (4× vs FP32)
- Widely used for post-training quantization and inference

INT4 (4 bits)

- Extremely low-bit integer representation
- Requires careful scaling, grouping (e.g., GPTQ, AWQ, GSQ)
- Provides major memory and bandwidth savings
- Popular for LLM deployment on GPUs and edge devices

INT2 / Binary (Research & Experimental)

- Ultra-low precision formats
- Aggressive compression with higher accuracy risk
- Mostly experimental for LLMs

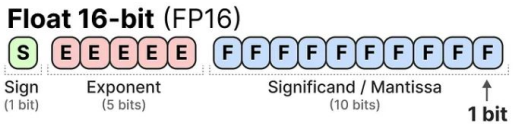


Fig 2: FP16 Example

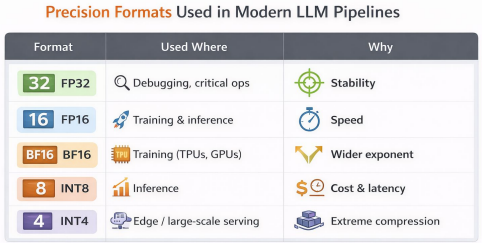


Fig 3: Precision Formats Used in Modern LLM Pipelines.

LLM Precision & Quantization

Value Representation Across Precisions

- Same value → different representations across precisions (Fig 4)
- **FP32**: high precision, high memory & compute
- **FP16**: faster, cheaper, lower numerical accuracy

Conversion to Lower Precision (Quantization)

- FP32 values mapped to discrete integer levels (Fig 5)
- Reduces model size and memory bandwidth
- Commonly applied during **inference**, not full training

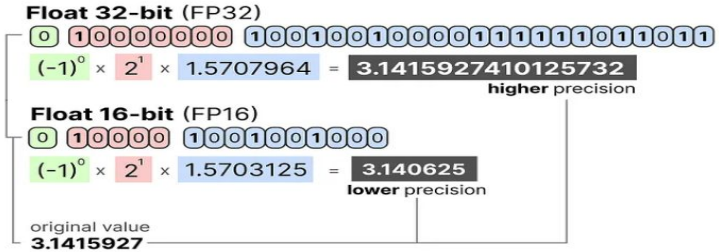


Fig 4: Floating-Point Precision Comparison (FP32 vs FP16). Reduced mantissa in FP16 leads to rounding error but faster computation

Image Source: https://medium.com/data-science/a-visual-guide-to-quantization-930ebcd9be94

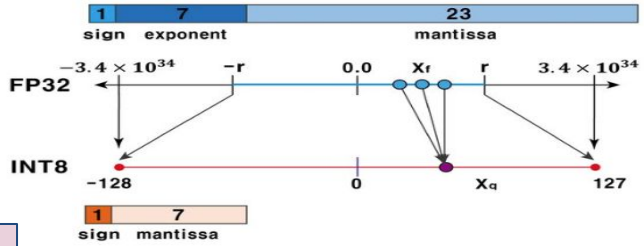


Fig 5: Quantization from FP32 to INT8.
Continuous FP32 values mapped to discrete integer levels

Image Source: https://www.researchgate.net/publication/363946099_A_Method_of_Deep_Learning_Model_Optimization_for_Image_Classification_on_Edge_Device

Why This Matters ?

Enables larger batch sizes, reduces GPU memory footprint, lowers latency and serving cost, and makes LLMs deployable at scale

Other Quantization Approaches - Post Training Quantization (PTQ)

Post-training quantization occurs when quantization is applied to an existing model. This converts the model from a floating-point representation to a lower precision fixed-point integer without the need for any retraining. This method does not require as much data as quantization aware training and is much faster.

Advantages:

- **Simplified training**: PTQ eliminates the need for modifications during model training, making it a more straightforward approach.
- **Flexibility**: PTQ allows for the use of pre-trained models, which can be beneficial when working with existing models or when the original training data is unavailable.

Challenges:

Accuracy loss: PTQ can introduce accuracy loss, particularly for smaller networks, due to the reduced precision of model weights and activations

Quantization Approaches - Quantization Aware Training (QAT)

- Quantization-aware training incorporates the conversion of weight during the pre-training or fine-tuning of an LLM.
- This allows for enhanced performance, but it demands a large amount of computational power and requires representative training data.
- QAT will usually produce a model with higher performance, but it is more expensive and will demand far more computing power.

Key Components

- **Fake Quantization**: Simulate quantization noise by applying quantization and dequantization operations to the model's weights and activations during training.
- **Quantization Configuration**: Specify the quantization parameters, such as bit-width, scaling, and zero-point, to control the quantization process.

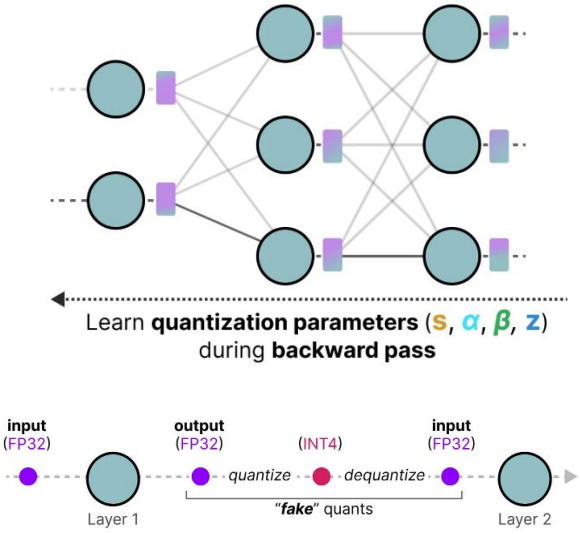


Fig 6: Quantization-Aware Training (QAT) Workflow

Quantization Steps

- **Weight Quantization**: Convert model weights from FP32 to a lower precision data type (e.g., INT8). This involves:
  - Determining the range of weight values
  - Mapping the weight values to a smaller set of discrete values (e.g., integers)
  - Minimizing information loss through techniques like: Linear quantization & Non-linear quantization (e.g., histogram-based)
- **Activation Quantization**: Convert model activations (output of each layer) from FP32 to a lower precision data type (e.g., INT8). This involves:
  - Estimating the range of activation values
  - Mapping the activation values to a smaller set of discrete values (e.g., integers)
  - Minimizing information loss through techniques like:
    - Linear quantization
    - Non-linear quantization (e.g., histogram-based)
    - Learned quantization (e.g., using neural networks
- **Model Calibration**: Adjust the quantized model to minimize errors and maintain accuracy. This may involve:
  - Fine-tuning the model with the quantized weights and activations
  - Adjusting the quantization parameters (e.g., number of bits, quantization intervals)

Popular Quantization Techniques

Generative Pretrained Transformer Quantization (GPTQ)

- One-shot post-training quantization method for large language models
- Converts model weights from FP32/FP64 to low-bit precision (e.g., 8-bit or 4-bit)
- Performs row-wise weight quantization, optimizing each row independently
- Minimizes accuracy loss using second-order error minimization and dynamic error compensation
- Adjusts later weights to offset quantization errors introduced earlier
- Enables faster and more memory-efficient inference due to low-precision arithmetic

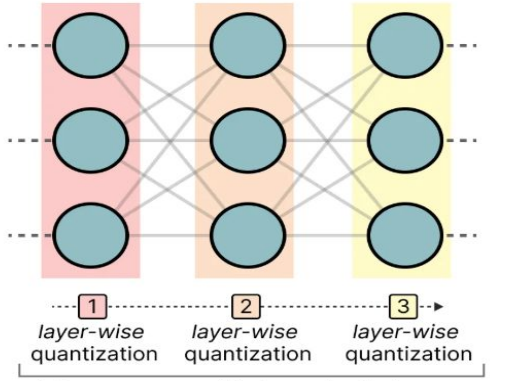


Fig 7: GPTQ quantization process

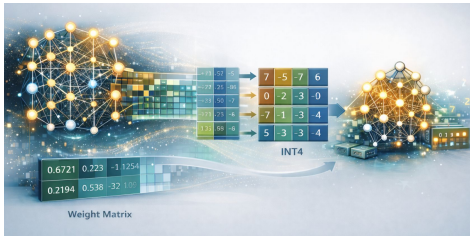


Image source: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Popular Quantization Techniques

Group Scaling Quantization (GSQ)

- A post-training quantization method inspired by **Activation-Aware Quantization (AWQ)**
- **Divides weight matrices into groups**, assigning a **shared scaling factor** per group
- Ensures all quantized weights fit within the **INT4 range**, reducing precision loss
- **Selects important weights based on activation impact**, not just weight magnitude
- Preserves **fine-grained activation information**, improving post-quantization accuracy
- Particularly effective for **4-bit quantization** of LLMs

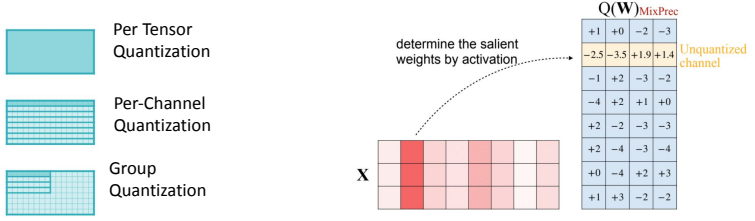


Image source: AWQ: Activation-aware Weight Quantization

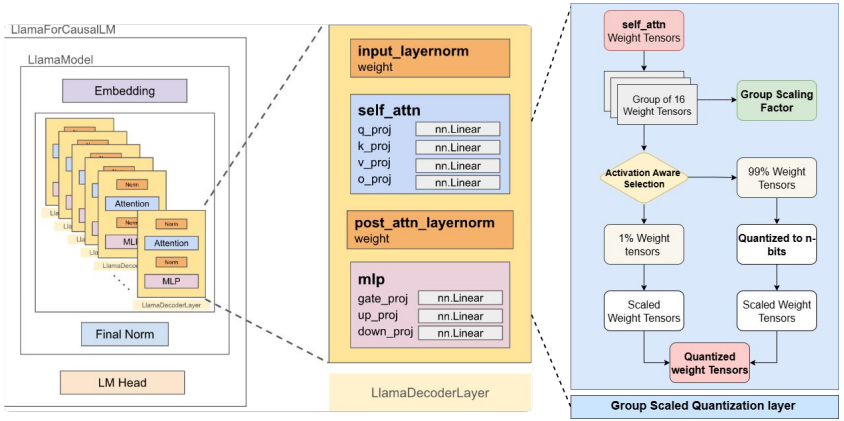


Fig: 8: Detailed view of the architecture behind GSQ quantization process.

Use Case

Objective: Evaluate GPTQ for post-training compression of small LLMs under realistic inference workloads.

Models Quantized
- LLaMA 1B
- Qwen 0.5B
- Phi 1.5B

Quantization Method
- GPTQ (Post-Training Quantization)
- Weight-only INT4 quantization
- One-shot calibration (no retraining)

Evaluation Tasks
- Information Retrieval
- Question Answering
- Mathematical Reasoning

Datasets
- MS MARCO (IR)
- BoolQ (QA)
- GSM8K (Math)

Metrics
- Accuracy, Perplexity, latency, throughput, Memory

Work Published: Sk, S., Dhal, D., Khosla, S., Dhaka, A., Parida, S., Shahid, S., ... & Bojar, O. (2025, September). LLM Compression: How Far Can We Go in Balancing Size and Performance?. In *Proceedings of the 15th International Conference on Recent Advances in Natural Language Processing-Natural Language Processing in the Generative AI Era* (pp. 1183-1187).

Use Case:

Key Results & Trade-offs

- **LLaMA-1B:** Quantization significantly improves MS MARCO (81.1 → 99.9%) and BoolQ (40.1 → 62.2%) with negligible impact on GSM8K
- **Qwen-0.5B:** More sensitive to quantization; MS MARCO drops, while BoolQ remains stable
- **Phi-1.5B:** Highly robust; performance remains **almost unchanged** after quantization
- **Retrieval & QA tasks** tolerate low-bit quantization better than **reasoning tasks (GSM8K)**
- **Larger models** show greater resilience to aggressive quantization

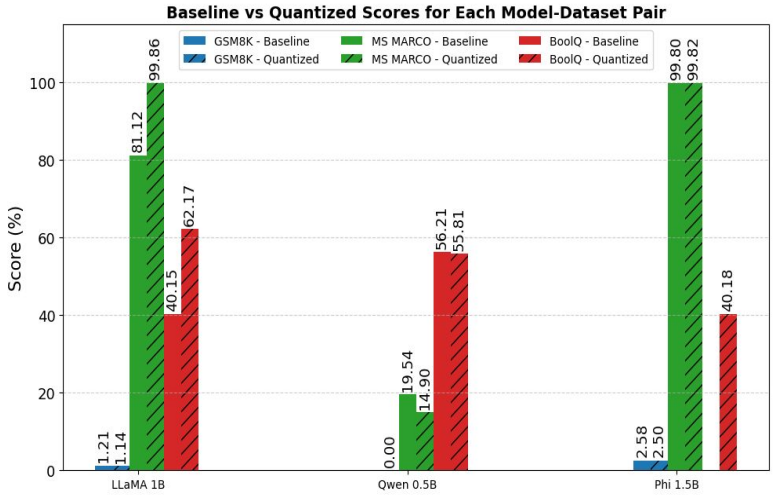


Fig: 9: Full-Precision vs. Quantized Model Performance

Trade-offs

- Up to **13× model size reduction** with minor accuracy loss
- Structured quantization preserves critical parameters
- Slight latency increase from group-level scaling

Takeaways

- Quantization enables deployment of LLMs on resource-limited hardware.
- **GPTQ** and **GSQ** provide different accuracy–efficiency trade-offs.
- Activation-aware and structured quantization preserve critical model behaviors.
- Model size can be reduced **up to 13×** with minimal performance loss.
- Quantization strategy should be **task-dependent** (e.g., reasoning vs. retrieval).
- Careful selection of methods is crucial for balancing **accuracy, latency, and memory**.

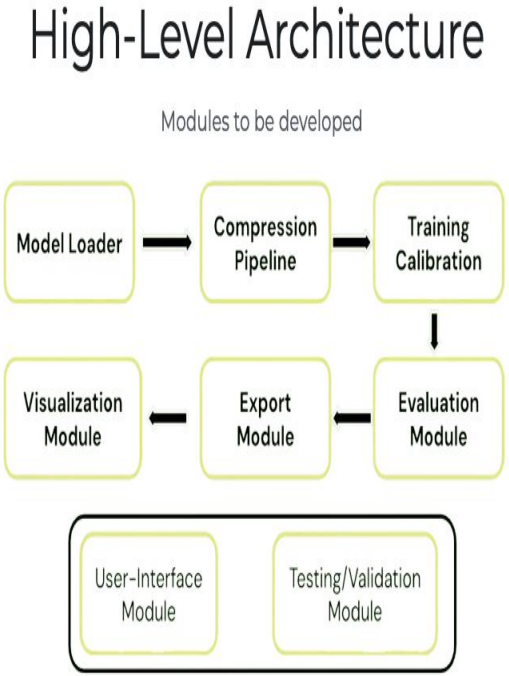
What Next...

Framework for Easy Deployment of Compressed and Optimized Models (FEDCOM)

Goal: FEDCOM simplifies deep learning model compression and deployment by providing a structured workflow for quantization, pruning, distillation, and optional fine-tuning — making efficient models accessible for resource-constrained environments.

Features: It supports multimodal models, configurable compression techniques, performance evaluation (size, speed, accuracy), visual insights into trade-offs, and export of deployment-ready compressed models.

Purpose: Designed to reduce technical complexity and enable users of varied expertise to efficiently compress and deploy optimized models for real-world scenarios



Open to research ideas and open-source contributions.

References

- Sk, S., et al. “LLM Compression: How Far Can We Go in Balancing Size and Performance?” *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2025)*, 2025, pp. 1183–1187. ACL Anthology, https://aclanthology.org/2025.ranlp-1.136.pdf.
- Frantar, Elias, et al. “GPTQ: Accurate Post-Training Quantization for Generative Pretrained Transformers.” *arXiv*, 2024, https://arxiv.org/abs/2210.17323.
- Lin, Ji, et al. “Group Scaling Quantization (GSQ) for Large Language Models: Activation-Aware Quantization.” *arXiv*, 2024, https://arxiv.org/abs/2306.00978.
- Lee, Byungsoo, et al. “LittleBit: Ultra Low-Bit Quantization via Latent Factorization.” *arXiv*, 2025, https://arxiv.org/abs/2506.13771.
- Grootendorst, Maarten. “A Visual Guide to Quantization.” *Maarten Grootendorst*, 2024, https://maartengrootendorst.com.
- Grootendorst, Maarten. “Quantization in vLLM: From Zero to Hero.” *Maarten Grootendorst*, 2024, https://maartengrootendorst.com.
- Grootendorst, Maarten. “Quantization Notes.” *Maarten Grootendorst*, 2024, https://maartengrootendorst.com.
- Grootendorst, Maarten. “Optimizing LLMs: Does Size Really Matter?” *Maarten Grootendorst*, 2024, https://maartengrootendorst.com.
- Grootendorst, Maarten. “How to Quantize an LLM with GGUF or AWQ.” *Maarten Grootendorst*, 2024, https://maartengrootendorst.com.

Acknowledgement

Acknowledgement is given to **Sahil Sk, Debasish Dhal, Sonal Khosla, Sk Shahid, Sambit Shekhar, Akash Dhaka, Shantipriya Parida, Dilip K. Prasad, and Ondřej Bojar** for their work titled **“LLM Compression: How Far Can We Go in Balancing Size and Performance?**

Thank you

Scan to access the Presentation slides