Quantization in Large Language Models: Problem and Solution

Raghad Mohamed

September.2025

Introduction and Motivation

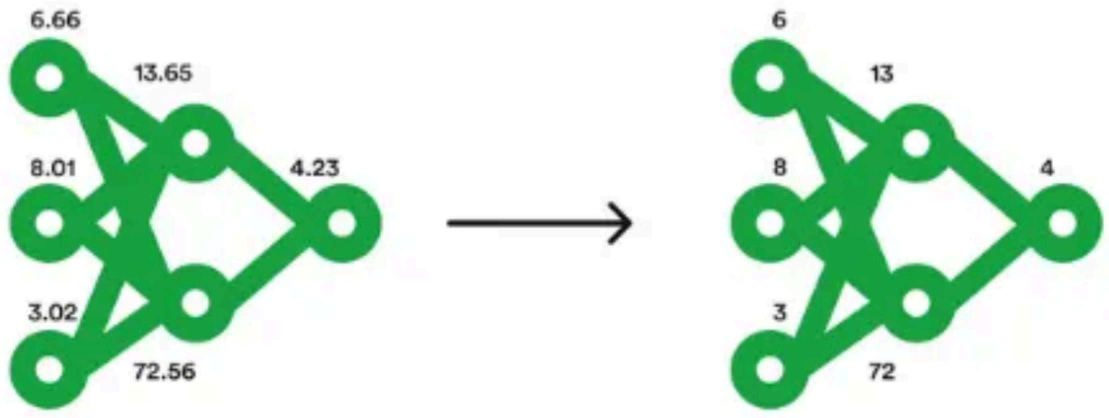
Large language models (LLMs) have earned their name from two defining characteristics: the enormous size of their parameter space and the vast amounts of training data used to build them. What concerns us here is the first aspect, the billions of parameters that these models contain. Each parameter requires storage and computational resources. To make this easier to imagine, think of each parameter as a person: every person needs a place to live and food to survive. Similarly, each parameter requires memory (its "house," in RAM) and energy (its "food," in GPU resources). As the number of parameters grows into the billions, the strain on hardware resources becomes overwhelming.

The Problem: Memory and Precision

Models today are typically trained using the FP32 data type, or 32-bit floating point numbers. This choice is not arbitrary. High precision is crucial because even tiny decimal differences can alter the model's predictions. Small fluctuations across billions of parameters add up, influencing outcomes such as whether a word is classified one way or another. However, storing every parameter as a 32-bit floating point number is expensive. A model with seventy billion parameters would require seventy billion times thirty-two bits of memory, a staggering demand that few systems can handle.

Quantization as a Solution

This is where quantization emerges, both as a challenge and as a solution. Quantization reduces the memory footprint of parameters by mapping them to smaller data types, typically INT8, which requires only eight bits per parameter instead of thirty-two. Naturally, this comes at the cost of reduced precision. The model may become slightly less accurate, but it becomes much more efficient, running faster and consuming far less memory and energy. The trade-off is crucial: without sacrificing some accuracy, these models would remain too large to be practical. With quantization, however, they become accessible on smaller devices such as mobile phones and even wearable technology.



Types of Quantization

There are two primary approaches to quantization. The first is **post-training quantization** (**PTQ**), where a model is trained in full FP32 precision and then quantized afterward. PTQ can be performed in different ways. Dynamic quantization converts the weights to INT8 while keeping the activations in FP32, only quantizing them dynamically during inference. Static quantization, also known as full integer quantization, reduces both weights and activations to Int8, leading to stronger efficiency gains but greater risk of accuracy loss. A middle ground is Float16 quantization, which maps values into 16-bit floating points, offering a compromise between precision and efficiency.

The second approach is **quantization-aware training** (**QAT**), which integrates quantization into the training process itself. Unlike PTQ, here the model is trained to adapt its weights to lower-precision formats. This awareness allows the model to minimize the degradation in accuracy that might otherwise occur, since it "learns" to live within the tighter precision constraints. While more computationally demanding during training, QAT often produces better results for tasks where precision is especially critical.

Quantization Techniques

Beyond when quantization is applied, there are different techniques for how it is performed. Uniform quantization divides the entire range of parameter values into equal intervals, mapping each interval to a representative quantized value. Non-uniform quantization instead gives more granularity to values that occur more frequently or are more important to model performance, thus preserving critical information. Min-max quantization defines the quantization range using the observed minimum and maximum parameter values, while logarithmic quantization maps values according to a logarithmic scale, providing finer precision for very small or very large numbers.

Use Cases and Best Practices

The choice of quantization strategy depends on the use case. For general deployment on cloud servers, post-training quantization with Float16 is often effective, providing a balance between efficiency and accuracy. For mobile devices, static Int8 quantization is usually preferred to maximize memory savings. In sensitive fields such as healthcare, where accuracy is non-negotiable, quantization-aware training combined with Float16 may be more suitable. For tiny devices like wearables or IoT sensors, dynamic Int8 quantization offers the smallest memory footprint, even if it comes with a sharper drop in precision.

Use Case	Best Quantization Type	Why
General deployment on cloud servers	PTQ + Float16	Good balance
Mobile devices	PTQ + Int8 Static	Strong memory savings
Healthcare (sensitive predictions)	QAT + Float16	Precision critical
Tiny devices (wearables, IoT)	PTQ + Int8 Dynamic	Smallest footprint

Conclusion

In conclusion, quantization sits at the heart of the challenges and opportunities in modern language models. It is both a problem, because reducing precision can hurt accuracy—and a solution, because without it, LLMs would remain too massive for most practical use cases. By carefully balancing precision with efficiency, researchers and engineers can ensure that these models are not only powerful but also usable, making them accessible across a wide range of devices and applications.

Resources

- Quantization in Deep Learning - GeeksforGeeks
- Optimize Your AI - Quantization Explained