Quantization in Large Language Models: A Student-Friendly Overview

What is Quantization?

Quantization can be compared to reducing the number of colors in an image: the “quantized” image on the right uses only 8 colors instead of the full spectrum, resulting in some loss of detail but a much smaller file size . In a similar way, quantizing a large language model means using fewer bits to represent the model’s numbers (its parameters), trading a bit of precision for big gains in efficiency.

In technical terms, **quantization** is a model compression technique that converts a model’s weights (and sometimes activations) from a high-precision format to a lower-precision format . For example, we might convert 32-bit floating-point values (FP32) into 8-bit or 4-bit integers (INT8 or INT4) . This drastically reduces the amount of data needed to store each number. In fact, converting from 32-bit to 8-bit representation cuts memory usage by about **75%**, and going to 4-bit cuts it by roughly **87.5%** . In other words, an INT8 model can be around **4× smaller** than the same model in FP32, and an INT4 model about **8× smaller**. Quantization achieves this compression by **rounding and mapping the original values** into a smaller set of possible values. There is some loss of numerical precision in this process, but the key is that the loss is kept within an “acceptable” range so that the model’s behavior remains nearly the same.

Why Quantize LLMs? (Goals & Benefits)

The primary goal of quantization is to make large models **more efficient** and easier to deploy. Large Language Models (LLMs) often have billions of parameters, so reducing the bits for each parameter yields big savings in memory and computation . Here are the main benefits and reasons we quantize LLMs:

- **Smaller Memory Footprint:** Lower-precision weights mean the model consumes far less memory/storage . This makes it feasible to load or deploy LLMs on devices with limited RAM. For instance, a model with 1 billion parameters would require about 4 GB in FP32, but only ~1 GB in INT8 or ~0.5 GB in INT4 . Quantized models are simply *lighter* to handle.

- **Faster Inference Speed:** Using fewer bits can speed up model inference, because there's less data to move and compute. Integer math (INT8/INT4 operations) can be executed very efficiently on modern hardware (like GPUs with tensor cores) . In practice, quantization can lead to **2–4× faster** inference on hardware that supports low-precision computation . Even on CPUs or older hardware, the reduced memory bandwidth requirements mean the model can process data more quickly .
- **Lower Power Consumption:** With fewer computations and reduced memory access, quantized models often use less energy for inference . This is especially important for battery-powered devices or large-scale deployments where energy efficiency translates to cost savings. INT8 and INT4 operations tend to be less power-hungry than full 32-bit operations .
- **Wider Deployment (Flexibility):** By shrinking models and making them more efficient, quantization allows LLMs to run in environments that would otherwise be impossible. A quantized LLM can potentially run on a **single GPU** instead of several, or even on a **CPU-only system**, without exhausting memory . This enables deploying advanced language models on **edge devices** like smartphones, browsers, or IoT hardware, as well as on modest servers and laptops . In short, quantization increases the portability and accessibility of LLMs.

*(In summary, quantization helps us **reduce the memory and computation demands** of large models while aiming to **maintain their accuracy** as much as possible. Next, we'll look at how it's done.)*

Common Quantization Techniques

There are several strategies to quantize a neural network, ranging from simple post-processing steps to more involved training procedures. Below are some common quantization techniques applied to LLMs, along with their high-level characteristics:

- **Post-Training Quantization (PTQ):** This approach quantizes a model *after* it has been fully trained, without further retraining . The model's weights (and optionally activations) are converted to lower precision in a one-time procedure. PTQ is relatively easy and fast since it doesn't require retraining the model on data . However, it may introduce some drop in accuracy if not done carefully, because the model wasn't originally optimized for the lower precision. Techniques like calibration (using a small dataset to find optimal scaling) are often used to mitigate accuracy loss in PTQ.

- **Quantization-Aware Training (QAT):** In QAT, the model is trained or fine-tuned with quantization in the loop, *aware* of the lower precision from the outset . This means during training (or a later fine-tuning stage), we simulate the effects of quantization (rounding to int8, etc.) so the model can adjust its weights to minimize any errors caused by lower precision. QAT generally yields **higher accuracy** in the final quantized model compared to naive PTQ, because the model learns to handle the reduced precision . The trade-off is that QAT requires more computation and time – essentially an extra training phase – and access to training data. It's often used when maximum accuracy is needed despite low-bit deployment.
- **GPTQ (GPT Quantization):** *Generative Pre-trained Transformer Quantization* (GPTQ) is an advanced **post-training quantization** algorithm specifically designed for efficiently compressing large transformers . It quantizes model weights to very low bit-widths (sometimes 3-bit or 4-bit) in a **layer-by-layer** fashion while using optimization to minimize the error introduced at each layer. GPTQ is notable because it can quantize extremely large models (tens or hundreds of billions of parameters) quickly, without needing to retrain the model . According to its authors, GPTQ achieves greater compression than earlier one-shot methods while keeping accuracy high . It's become a popular choice for compressing LLMs like OPT, BLOOM, and Llama models after training, due to its balance of speed and accuracy. In practice, GPTQ can compress a model like BLOOM (176B parameters) to 4-bit in just a few GPU hours , enabling much faster inference (3–4× speedups observed on modern GPUs) .
- **AWQ (Activation-Aware Weight Quantization):** AWQ is another recent quantization technique that focuses on *weight-only* post-training quantization while being mindful of activations. Unlike standard PTQ which treats weights in isolation, AWQ uses a small set of data to analyze the model's **activation distribution** during inference . With those activation statistics, it then chooses how to quantize weights such that the impact on the model's outputs is minimized . In essence, AWQ "protects" the most important weights (based on their effect on activations) and finds optimal

scaling factors for quantization. The result is that AWQ can quantize models to very low precision (often 4-bit) **with almost no loss in accuracy** . This technique was shown to preserve model performance by focusing on the tiny fraction of critical weights and allowing slight error on less critical ones . Like GPTQ, AWQ is a post-training method – it doesn't require full retraining, only a brief calibration with some sample data (e.g. a few hundred sentences for a 70B model) . AWQ has proven effective in compressing big LLMs (such as Llama 2 70B) to 4-bit weights that run efficiently on a single GPU, without significant accuracy drops .

> Note:
>
> **quantization-aware fine-tuning**
>
> *after*
>
> *during*

Bit-Width Options and Their Implications

When we talk about "8-bit" or "4-bit" quantization, we're referring to the number of bits used to store each weight/activation value:

- **INT8 (8-bit integers):** Using 8 bits means each number can represent 256 discrete values. An 8-bit model uses one **quarter** of the memory of the same model in 32-bit floats . In practice, 8-bit quantization is considered quite *safe* for many LLMs – it usually preserves model accuracy to within about 1% of the original performance . Most modern hardware (GPUs, NPUs, and even CPUs with AVX512/VNNI instructions) have special support for fast INT8 arithmetic, so INT8 models can take advantage of that for speed. Because it offers a good balance of compression and accuracy, INT8 is a popular choice for quantizing large models.
- **INT4 (4-bit integers):** Using 4 bits per value means only 16 possible levels, so the model is extremely compressed – about **8× smaller** than the FP32 version . INT4 quantization can dramatically shrink model size (for example, a 30 GB model in FP32 might become around 3.75 GB in 4-bit form). The challenge is that with so few bits, there is a higher risk of model quality degradation. A straightforward INT4 quantization might lead to a noticeable drop in accuracy (e.g. a few percentage points off the original accuracy on some benchmarks) . However, with advanced methods like GPTQ or AWQ

(which handle outliers and find optimal scales), many LLMs can be quantized to 4-bit with only minimal loss in performance . INT4 is often used when maximum compression is needed, but it may require more careful calibration to get good results. Not all hardware natively supports 4-bit math (some GPUs do, or they pack 4-bit values into higher-bit operations), so the actual speedup from INT4 can vary, but the memory savings are significant.

- **Other bit-widths:** In between 8 and 4 bits, sometimes **INT16** (or half-precision floats FP16) is used as a simpler form of quantization – many training processes already use FP16 to save memory. INT16 or FP16 usually has **negligible impact** on accuracy but only halves the memory usage relative to FP32. Going lower, **2-bit or 3-bit** quantization is an active research area; these offer even greater compression but the accuracy trade-offs become more severe (and hardware support is limited). In practice, 8-bit and 4-bit are the most common targets for LLM quantization today, with 3-bit being experimental in some cutting-edge research.

Implications: The choice of bit-width is a trade-off between *model efficiency* and *model fidelity*. Higher bit-width (e.g. 8-bit) preserves more precision (hence usually higher accuracy), but doesn't reduce size as much. Lower bit-width (e.g. 4-bit or below) squeezes the model more and saves more memory, at the risk of greater information loss. Moreover, the bit-width interacts with hardware capabilities: if your accelerator supports fast INT8 operations but not INT4, you might find 8-bit quantization gives a bigger speed boost, even if 4-bit uses less memory. The goal is to choose the lowest precision that still keeps the model's accuracy within an acceptable range for your application.

Trade-offs: Performance vs. Accuracy

Quantization involves a fundamental trade-off between **performance (efficiency)** and **accuracy (model quality)**. The good news is that for many large models, you can compress them *significantly* with only a minor impact on accuracy – but push it too far, and the model's responses or predictions can start to degrade.

- **Efficiency Gains vs. Quality Loss:** As we reduce the precision, we gain efficiency: models run faster and use less memory. However, using fewer bits also introduces approximation error (quantization error). For example, 8-bit quantization might change model outputs only very slightly (often <1% difference in accuracy) , while 4-bit quantization can cause a bit more drop in fidelity (perhaps a few percent drop in accuracy) . If the quantization is done carefully (using proper techniques and perhaps a little fine-tuning), the accuracy loss can be minimized to an acceptable level. The *sweet spot* is often INT8 for a near-zero loss, but INT4 can work with advanced methods. Ultimately, there's a balancing act: too much compression will hurt model performance on tasks, so we choose a precision that gives the best of both worlds.

- **Hardware and Compatibility:** Another trade-off factor is how well the target platform supports the chosen quantization. A quantized model only delivers speed benefits if the hardware can exploit those lower precision computations. For instance, many GPUs can execute INT8 arithmetic at higher throughput than FP32, but if a certain processor has no INT4 support, a 4-bit model might not run any faster (despite being smaller). In some cases, using a slightly higher precision (like 8-bit) yields a better overall performance because it aligns with hardware capabilities. It's important to consider the deployment environment: quantization is most beneficial when it matches the hardware's sweet spot (be it 8-bit, 4-bit, etc.) . Otherwise, the main benefit in those cases might be memory savings alone.

In summary, quantization asks the question: *How much can we shrink this model before it starts making too many mistakes?* Finding the right level often requires experimentation and validation. Fortunately, many LLMs have proven to be quite robust to quantization, maintaining high accuracy even at 8 or 4 bits per weight, especially when using the latest quantization algorithms.

Use Cases for Quantized LLMs

Quantization is widely used whenever we need to deploy or run large models under constrained resources or at lower cost. Some typical use cases include:

- **Edge and Mobile Deployment:** Running LLMs on edge devices (like smartphones, tablets, or embedded systems) is made possible through quantization. These devices have limited memory and compute power, so a smaller model is essential. A quantized model can fit into the memory of a phone or a single-board computer and run with tolerable latency. For example, developers use 8-bit or 4-bit quantization to compress models so that features like on-device chatbots or smart assistants can work without offloading to a server. The lower power consumption of quantized models is also crucial for battery-powered devices .

- **Inference Optimization in the Cloud:** Even in server and cloud settings where hardware is powerful, quantization helps to cut costs and improve scalability. By reducing model size, you can serve more concurrent users or run more model instances on the same GPU/TPU hardware. Quantization also speeds up response times, which is important for real-time applications. Importantly, it enables **larger models to run on fewer GPUs**. For instance, a 70-billion-parameter model in 4-bit can run on a single 80GB GPU , whereas in 16-bit it might require multiple GPUs or be impossible to load on one. This means organizations can deploy advanced LLMs with less specialized hardware, making deployment more cost-effective. Quantized models are also easier to deploy in containerized or serverless environments due to their smaller footprint .
- **Resource-Constrained Environments:** In research, personal projects, or non-profit endeavors, one might not have access to high-end hardware at all. Quantization allows large models (like those powering GPT-style applications) to be run on commodity hardware, including standard CPUs (albeit at slower speeds). Projects like LLaMA.cpp famously demonstrated that with 4-bit quantization, a 7B or 13B parameter model can run on a typical laptop CPU. This opens up accessibility: more people can experiment with and use LLMs without needing top-tier GPUs. In enterprise scenarios, quantization helps in deploying LLMs in edge servers or hybrid cloud setups where optimizing for lower resource usage is important .

In conclusion, quantization is a key technique in the LLM optimization toolkit. It enables the **scaling-down** of model size and compute requirements by using clever approximations, all while aiming to **keep the model's answers and behavior nearly as good as before**. By understanding and applying quantization, we can bring the power of large language models to more environments – from a cloud data center down to a cell phone – making AI both more efficient and more widely accessible.

📚 Sources (APA 7 format)

- Ainslie, J., Narang, S., Wang, L., Krikun, M., Song, H., Wang, Z., ... & Shoeybi, M. (2023). *LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale*. arXiv. https://arxiv.org/abs/2208.07339
- Bondarenko, A., Dettmers, T., Zettlemoyer, L., & Gupta, S. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs*. arXiv. https://arxiv.org/abs/2305.14314

- Frantar, E., & Alistarh, D. (2022). *GPTQ: Accurate Post-training Quantization for Generative Pre-trained Transformers*. arXiv. https://arxiv.org/abs/2210.17323
- Frantar, E., Xu, S., Stock, P., & Alistarh, D. (2023). *SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression*. arXiv. https://arxiv.org/abs/2306.03078
- Lin, Y., Chiang, W. L., Lin, Y. C., Xu, S., & Kolter, J. Z. (2023). *AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration*. arXiv. https://arxiv.org/abs/2306.00978
- Mishra, A., & Marr, D. (2017). *Apprentice: Using Knowledge Distillation Techniques To Improve Low-Precision Network Accuracy*. arXiv. https://arxiv.org/abs/1711.05852
- Park, J., Ahn, S., Ryu, H., Yoo, S., & Shin, J. (2022). *Faster and Better: A Survey of Quantized Neural Networks*. arXiv. https://arxiv.org/abs/2111.05691
- Zhang, Z., Park, J., Zhang, Y., & Fu, Y. (2023). *SqueezeLLM: Dense-and-Sparse Quantization*. arXiv. https://arxiv.org/abs/2306.07629
- Zhao, W., Chiang, W. L., Huang, B., & Kolter, J. Z. (2023). *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. arXiv. https://arxiv.org/abs/2211.10438