Lecture 11.3: Efficient LLM Deployment

Quantization, QLoRA, and Running Models at Home

Heman Shakeri

The Democratization Challenge

From billion-parameter models to your laptop

Topics:

1. The memory bottleneck: Why 7B models need 66+ GB for training
2. Quantization fundamentals: Representing weights with 4 bits instead of 16
3. QLoRA: Fine-tuning 7B models on consumer GPUs (8–12 GB)
4. Hands-on: Fine-tune and deploy models on your machine
5. Trade-offs: Speed, quality, and hardware constraints

Practical Outcome

By the end, you will fine-tune and serve models on consumer hardware!

The Billion-Parameter Reality

Modern language models are remarkably capable but extraordinarily large:

Open models:

- **Gemma 3**: 270M, 2B, 9B, 27B (Apache-2.0)
- **LLaMA 3.1**: 8B, 70B, 405B (Apache-2.0)
- **Mistral**: 7B (Apache-2.0)
- **Qwen 2.5**: 7B, 14B, 72B (Apache-2.0)

Proprietary models:

- **GPT-4**: Rumored 1.7 trillion parameters
- **Claude 3**: Multiple variants (sizes undisclosed)

Question: Can you run these on your laptop? Let us do the math.

Memory Requirements: Just Storing Weights

Each parameter is a number. Standard storage formats:

Format	Bits/param	Bytes/param	7B model
FP32 (float32)	32	4	28 GB
FP16 (float16)	16	2	14 GB
BF16 (bfloat16)	16	2	14 GB
INT8	8	1	7 GB
INT4	4	0.5	**3.5 GB**

Example: 7 billion parameters in FP16:

$$7 \times 10^9 \text{ params} \times 2 \text{ bytes/param} = 14 \text{ GB}$$

This is just the **storage cost**. **Inference** requires *more* memory to hold the **KV-Cache** (the "memory" of the conversation), which can be several GBs itself for long contexts.

Your hardware:

- Consumer laptop: 8–16 GB RAM (shared with OS, browser)
- RTX 3060/4060 GPU: 8–12 GB VRAM
- MacBook Pro M3: 16–32 GB unified memory

What About Smaller Models?

Gemma 3 270M in different precisions:

Format	Inference	Training (w/ Adam)
FP32	1.08 GB	13 GB
FP16	540 MB	6.5 GB
INT8	270 MB	–
INT4	135 MB	–

Much more manageable! But what if you want to use larger, more capable models?

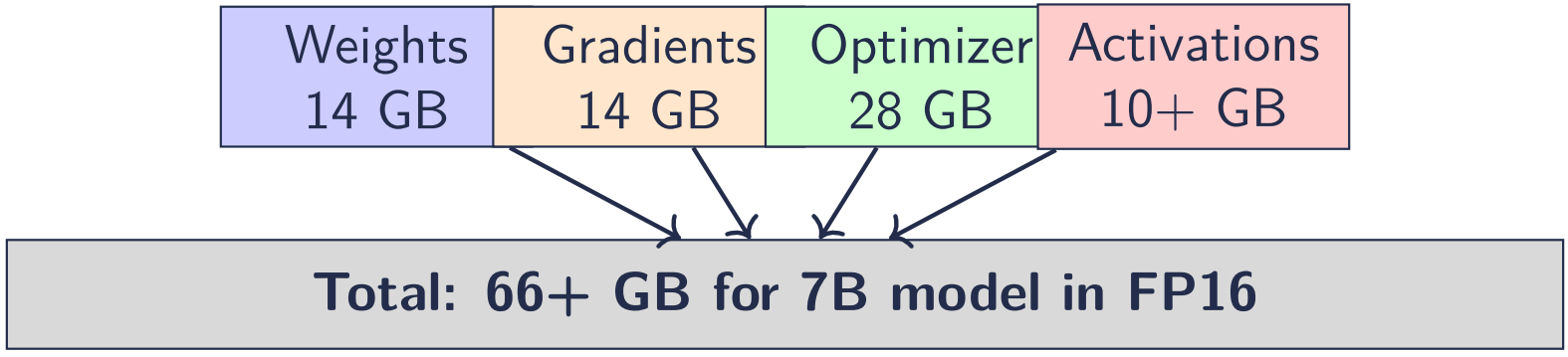
Trade-off:

- 270M: Fits easily, limited capability
- 7B: Better capability, needs optimization
- 70B+: State-of-art capability, requires advanced techniques

Today's goal: Make 7B models as accessible as 270M models used to be!

Training Makes It Worse

During training, memory requirements explode:



Why so much?

- **Gradients**: Same size as weights (14 GB for FP16)
- **Optimizer states**: Adam stores first and second moments in FP32 (28 GB!)
- **Activations**: Stored during forward pass for backpropagation (10+ GB)

This requires enterprise GPUs (A100 80GB, H100 80GB) costing \$10,000–\$40,000!

The Challenge and Solution

The Challenge

Goal: Enable researchers and practitioners to:

- **Run** 7B models for inference on 8 GB GPUs
- **Fine-tune** 7B models on 12 GB GPUs
- **Deploy** models on CPUs and consumer hardware

The Solution

Three-pronged approach:

1. **Quantization**: Fewer bits per parameter
2. **Efficient fine-tuning**: QLoRA (quantization + LoRA)
3. **Optimized serving**: Specialized inference engines

Result: Democratize access to powerful LLMs!

The Core Idea

Quantization: Represent weights with fewer bits while preserving performance

Analogy: Image compression

- Raw photo: 24 bits per pixel (16 million colors)
- JPEG: Approximate values, much smaller file
- Visual quality: Often indistinguishable

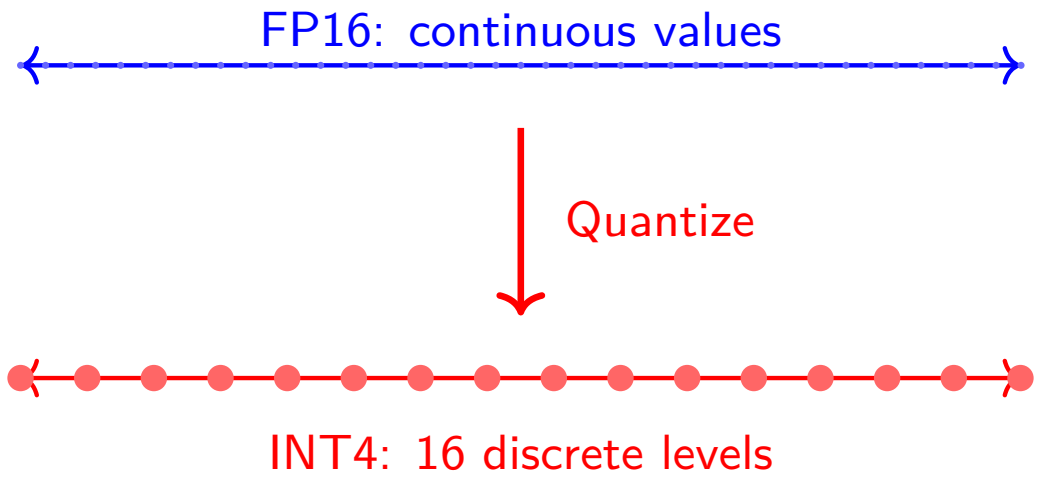
For neural networks:

- Original weights: FP16 or FP32 (continuous values)
- Quantized weights: INT8 or INT4 (discrete levels)
- Model quality: Minimal degradation with proper techniques!

Key insight: Neural networks are remarkably robust to reduced precision

How Quantization Works

Basic quantization maps floating-point weights to integers



The quantization formula:

Given floating-point weight $w \in \mathbb{R}$, quantize to integer:

$$w_{\text{quant}} = \text{round}\left(\frac{w - z}{s}\right)$$

where s = scale, z = zero-point

To reconstruct (dequantize):

$$\hat{w} = w_{\text{quant}} \cdot s + z$$

Example: 8-bit Quantization

Task: Quantize weights in range $[-0.5, 0.5]$ to INT8 (256 values)

Steps:

1. **Determine range**: $w_{\text{min}} = -0.5$, $w_{\text{max}} = 0.5$
2. **Compute scale**:

$$s = \frac{w_{\text{max}} - w_{\text{min}}}{2^8 - 1} = \frac{1.0}{255} \approx 0.00392$$

3. **Quantize** $w = 0.12$:

$$w_{\text{quant}} = \text{round}\left(\frac{0.12}{0.00392}\right) = \text{round}(30.6) = 31$$

4. **Dequantize**:

$$\hat{w} = 31 \times 0.00392 \approx 0.1215$$

Error: $|0.12 - 0.1215| = 0.0015$ (very small!)

Quantization Strategies

Method	Description	Bits
PTQ	Post-Training Quantization	4–8
QAT	Quantization-Aware Training	4–8
Weight-Only	Quantize weights, keep activations FP16	3–4
Weight + Act	Quantize both	8

For LLMs, weight-only quantization dominates:

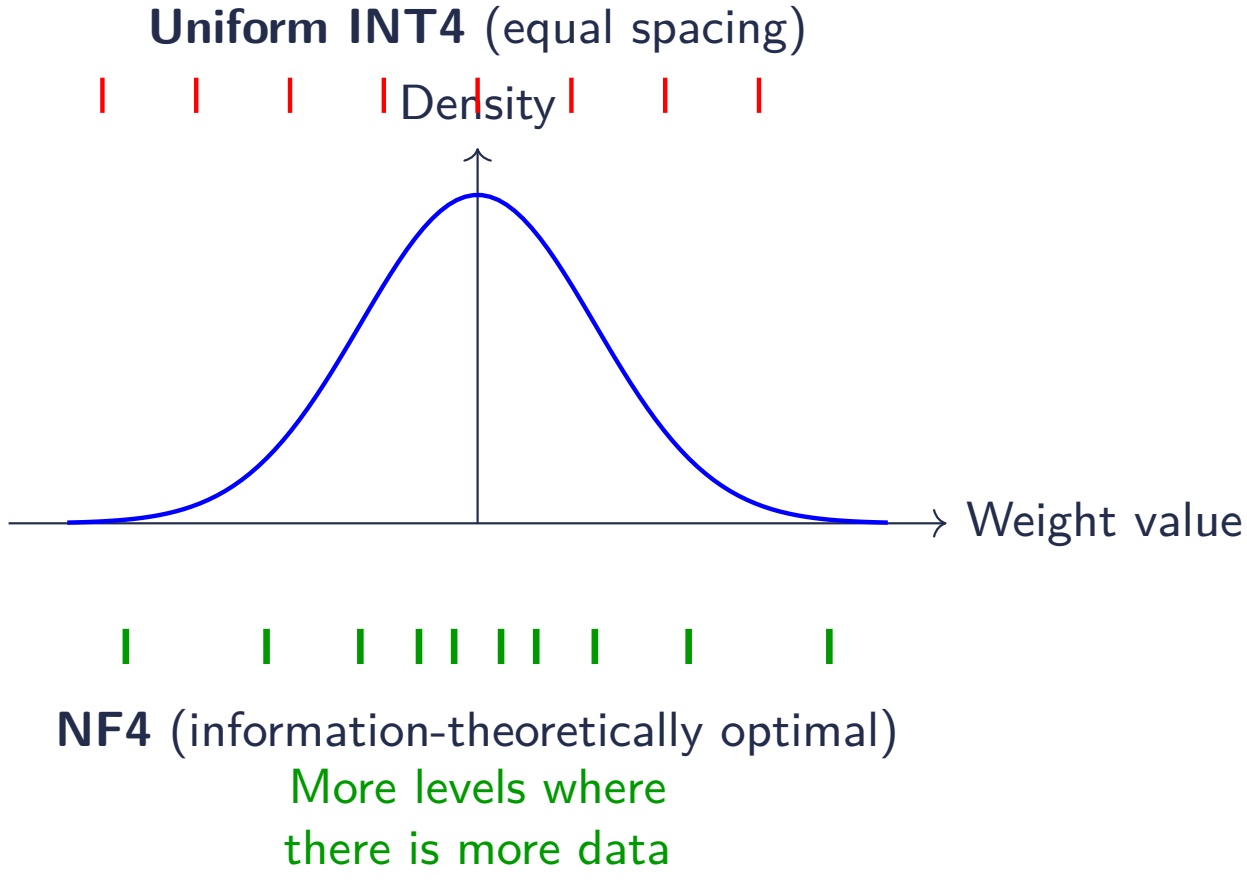
- Quantize weights to INT4 (16 values)
- Keep activations in FP16 during computation
- **4× memory reduction** with minimal quality loss!

Weight-only works because weights (W) are static and have a nice, stable distribution. Activations (x) are dynamic (change with every input) and have large outliers. Quantizing these outliers (e.g., in W_x) is what causes performance degradation.

NF4: Normal Float 4

Standard INT4 uses uniform spacing: $\{0, 1, 2, \ldots, 15\}$

But neural network weights follow a **normal distribution**!



NF4 (used in QLoRA): More quantization levels near zero, fewer in tails

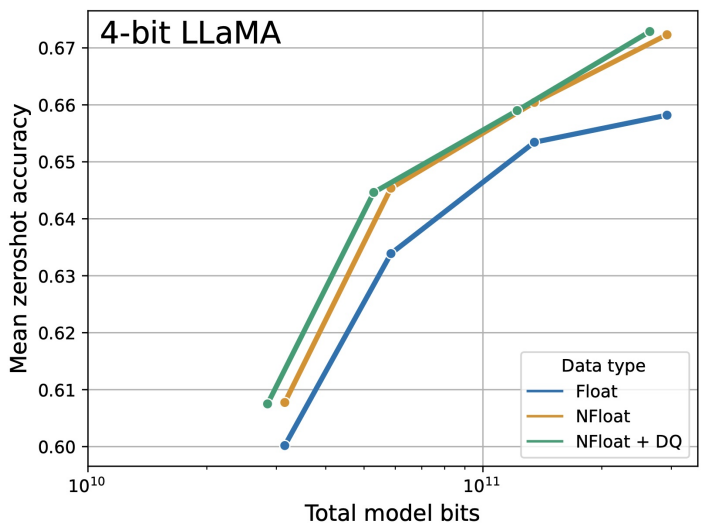
This gives **better precision where it matters**!

Quantization Methods for LLMs

Method	Description	Use
GPTQ	Post-training using Hessian info	**3** – **4**-bit **inference** (static quantiza
AWQ	Activation-aware (protects salient weights)	Fast inference
GGUF	Format for llama.cpp (CPU/Metal)	Laptops, no GPU
bitsandbytes	On-the-fly 4/8-bit with NF4	**Training** (**QLoRA**) & inference

Bottom line: With 4-bit quantization, the **16**-bit model size is reduced by $\approx$ **4**$\times$: e.g. 70B model: $\sim$ 140 GB $\rightarrow$ **41** GB.

Quality Loss: **4**-bit QLoRA with NF4 is shown to **match 16-bit performance** [Fig2 of QLoRA paper:].



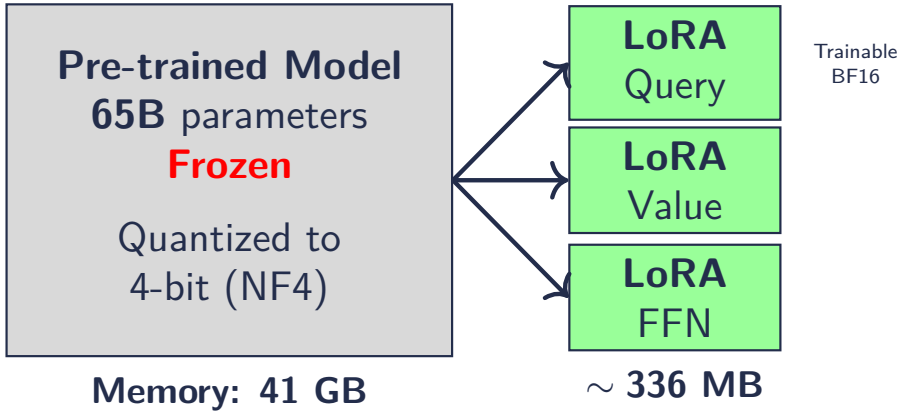
QLoRA

Recall Memory savings for adapters:

- Original trainable params: $d \times k$
- LoRA trainable params: $r(d + k)$ (typically $r = 8$ or $r = 16$)

The problem: Even with LoRA, the **65**B model still requires $>$ **780** GB in **16**-bit for finetuning!

QLoRA = **Quantized** base model + **LoRA** adapters



Total: $<$ **48 GB**
Fits on a single professional GPU!

QLoRA: Visualizing the Memory Savings

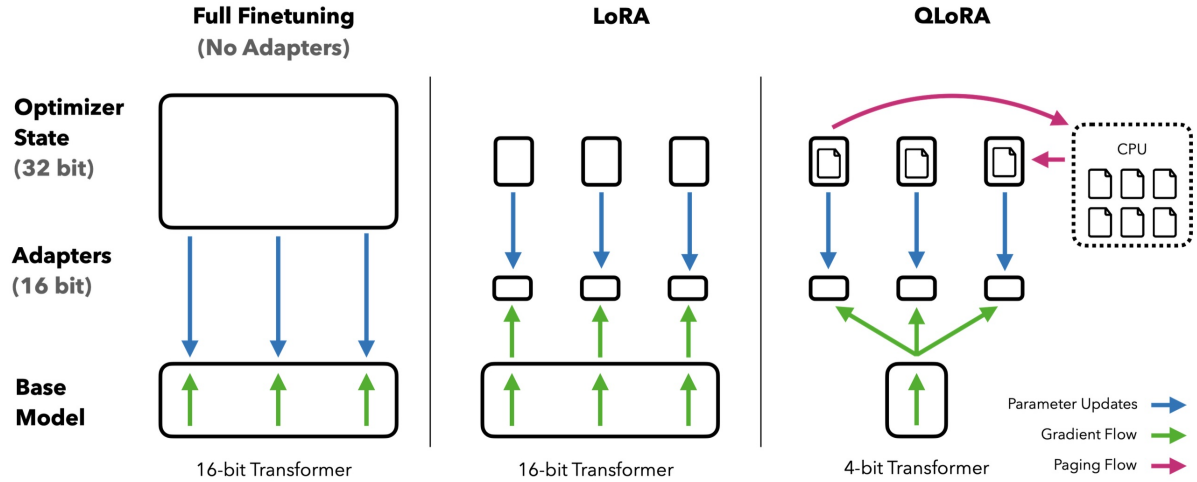


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

- Full Finetuning updates a huge **16**-bit model and stores a massive **32**-bit Optimizer State.
- LoRA reduces the Optimizer State slightly by adding small **16**-bit Adapters, but the base **16**-bit model remains the memory bottleneck.
- QLoRA fundamentally solves the bottleneck by quantizing the base model to **4**-bit, while retaining the small **16**-bit Adapters and using Paged Optimizers (Paging Flow) to manage the Optimizer State with the **CPU**.
- *Paging* is an old memory management scheme that allows a system to store and retrieve data from secondary storageand the main memory

QLoRA: Key Innovations

1. **4-bit NormalFloat (NF4)**: Quantize base to 4 bits optimally
   - Information-theoretically optimal for normally distributed weights
   - Better empirical results than 4-bit Integers and 4-bit Floats
2. **Double Quantization (DQ)**: Quantize the quantization constants
   - Quantization constants are typically 32-bit floats
   - DQ quantizes these constants, saving an average of **0.37** bits per parameter
   - Saves $\sim$ **3** GB for a **65**B model
3. **Paged Optimizers**: Handle memory spikes via unified memory
   - Uses **NVIDIA** unified memory for automatic page-to-page transfers between **CPU** and **GPU**
   - Prevents out-of-memory errors during gradient checkpointing peaks
4. **LoRA adapters in BF16**: Keep trainable parameters in high precision
   - The frozen base is 4-bit; the trainable adapters are **16**-bit **BrainFloat** (**BF16**)
   - Training with high precision is critical for stable gradient updates

QLoRA for Gemma 3 270M

Can we use QLoRA for smaller models?

Yes, but it's overkill!

Method	Memory	When to Use
Full FT (FP16)	6.5 GB	Plenty of VRAM
LoRA (FP16)	4 GB	Save some memory
QLoRA (4-bit)	2.5 GB	Extreme constraints

For Gemma 3 270M:

- Regular LoRA already very efficient
- QLoRA adds complexity without much benefit
- Only use if you have $<$4 GB VRAM

Sweet spot for QLoRA: 7B–70B models on consumer GPUs

For larger models (70B+):

- 70B in 4-bit: $\sim$35 GB (still needs high-end GPU)
- May need model parallelism across multiple GPUs

Beyond Training: Efficient Inference

You've fine-tuned your model. Now what?

Deployment requirements:

- Low latency (time to first token $<$100ms)
- High throughput (many requests/second)
- Memory efficiency (serve multiple users)
- Hardware flexibility (GPU, CPU, edge devices)

Challenge: Standard HuggingFace Transformers is not optimized for production

Solution: Specialized inference engines

Inference Engine Landscape

Engine	Specialty	Hardware
vLLM	High-throughput GPU serving	NVIDIA GPU
llama.cpp	CPU/edge deployment	CPU, Metal, CUDA
MLX-LM	Apple Silicon optimization	M1/M2/M3
TGI	Production-ready API server	NVIDIA GPU
TensorRT-LLM	Maximum GPU performance	NVIDIA GPU

Key innovation across all engines:

- Quantization (INT4/INT8)
- KV-cache optimization
- Batching and scheduling
- Kernel fusion and optimization

The Democratization Vision

The Future is Accessible

Three years ago: Running LLMs required million-dollar infrastructure
Today: With the techniques you learned:

- Run 7B models on $300 consumer GPUs
- Fine-tune on laptops with 16 GB RAM
- Deploy on edge devices and phones

Tomorrow: Even more efficient methods will emerge!

- Better quantization (sub-4-bit without quality loss)
- Sparse models (Mixture-of-Experts at edge)
- Hardware-software co-design

Your opportunity: Use these tools to build LLM applications that were impossible just a few years ago!

Real-World Impact

The techniques you learned today are **actively used in production**:

Startups: Deploy LLM services without massive GPU clusters

Research labs: Enable more scientists to experiment with large models

Edge AI: Run models on robots, drones, IoT devices

Privacy applications: Keep sensitive data local with on-device inference

Developing regions: Access AI without expensive cloud services

"The best model is not the largest model—
it is the model you can actually deploy and use."

Further Reading

Core papers:

- Dettmers et al. (2023). "QLoRA: Efficient Finetuning of Quantized LLMs"
- Frantar et al. (2022). "GPTQ: Accurate Post-Training Quantization for GPT"
- Lin et al. (2023). "AWQ: Activation-aware Weight Quantization"
- Kwon et al. (2023). "Efficient Memory Management for LLM Serving" (vLLM)
- Dettmers et al. (2022). "LLM.int8(): 8-bit Matrix Multiplication"

Tools and frameworks:

- vLLM: `docs.vllm.ai`
- llama.cpp: `github.com/ggerganov/llama.cpp`
- AutoAWQ: `github.com/casper-hansen/AutoAWQ`
- MLX: `ml-explore.github.io/mlx`
- bitsandbytes: `github.com/TimDettmers/bitsandbytes`
- PEFT library: `huggingface.co/docs/peft`

Emerging techniques:

- GPTQ-R: Refined GPTQ with better Hessian approximation
- QuIP: Quantization with Incoherence Processing (2-bit!)
- SpQR: Sparse Quantized Representation
- SmoothQuant: Migrate difficulty from activations to weights