Efficient Finetuning of Quantized LLMs

QLoRA

HUMANE Lab

허윤서

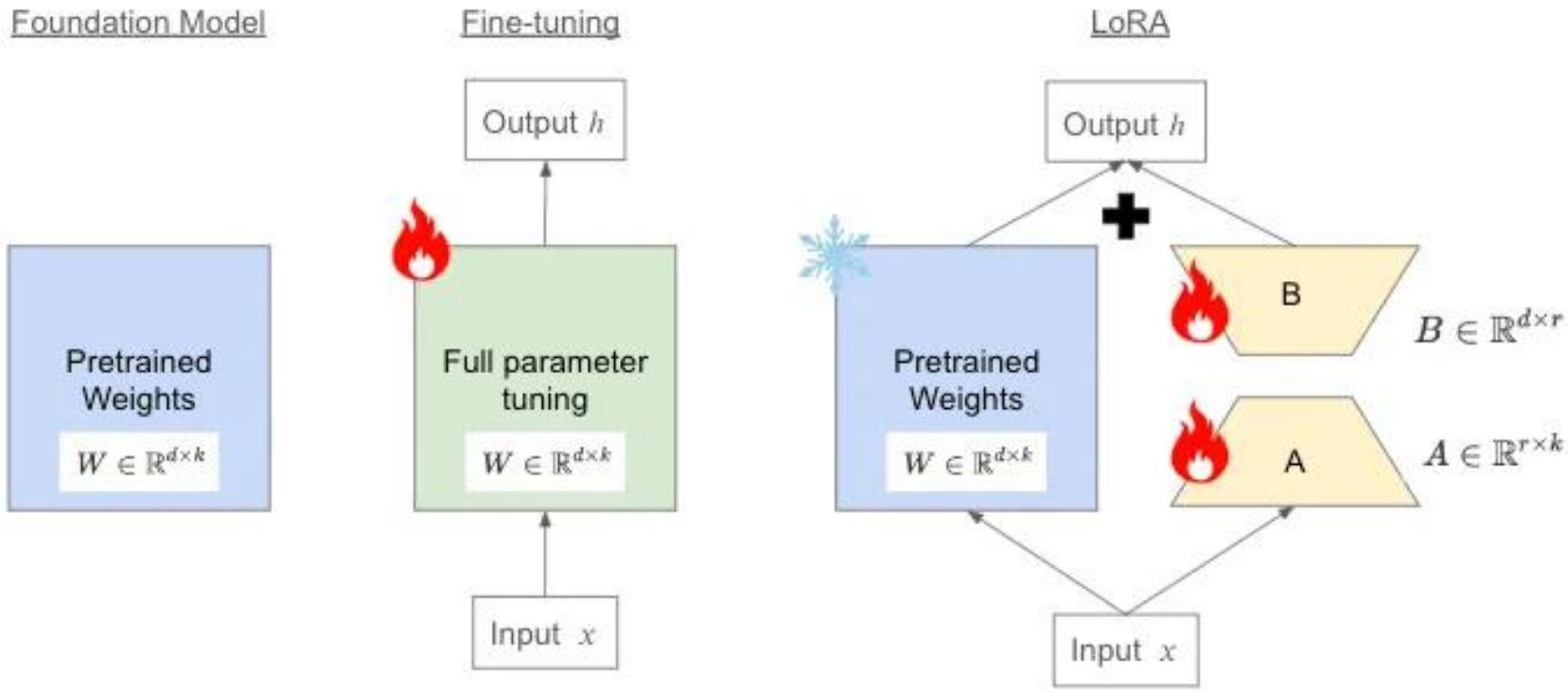
2025.03.07

What is QLoRA? – Overview

- **QLoRA = Quantized Low-Rank Adaptation** – A method to fine-tune large language models using **4-bit quantization + LoRA adapters**.
- **Goal:** Finetune a 65B LLM on a single GPU (48 GB) *without* performance loss. Achieve 16-bit fine-tuning quality with a fraction of the memory.
- **Key Innovations:** 4-bit NormalFloat (NF4) quantized weights, Double Quantization of scales, Paged Optimizer memory management.
- **Big Result:** QLoRA (65B Guanaco model) reached **99.3%** of ChatGPT's performance on Vicuna benchmark in 24h on one GPU. Enabled democratization of LLM fine-tuning.

LoRA Recap – Low-Rank Adaptation (1)

- **LoRA adds trainable low-rank matrices** A, B to mimic weight updates. Original weight W_0 is frozen; model output = $\mathrm{W_0x} + (\mathrm{B} \times \mathrm{A})\mathrm{x}$



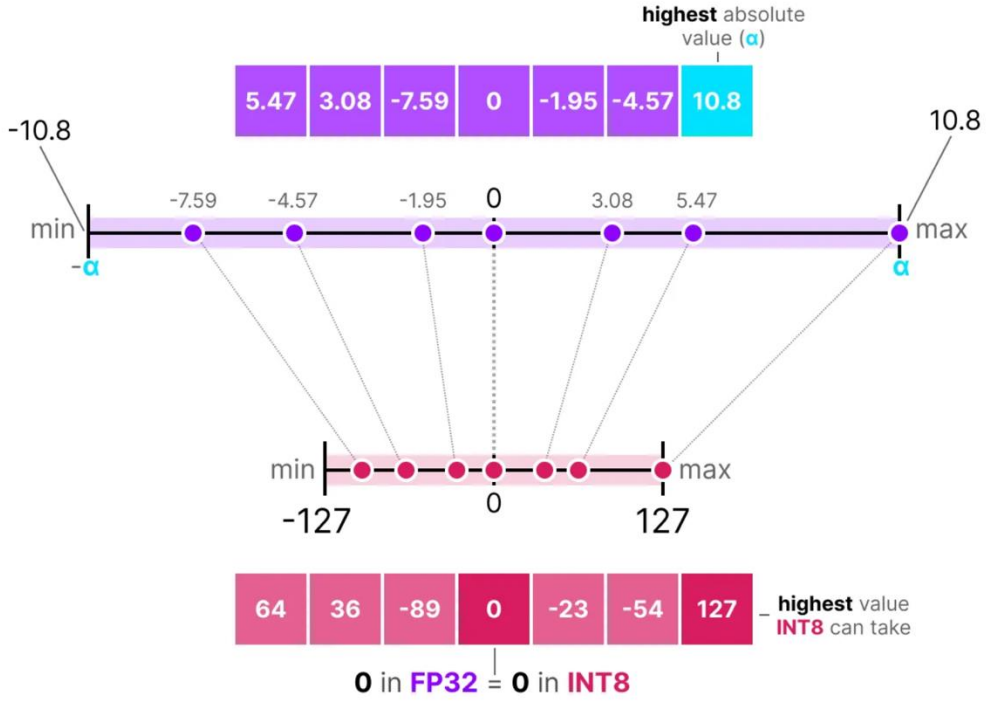
LoRA Recap – Low-Rank Adaptation (2)

- Only A and B (the LoRA adapter params) are trained (much fewer params). *E.g.*, if W is 1000×1000 (1M params), a LoRA rank $r = 8$ introduces $A(1000 \times 8)$ and $B(8 \times 1000)$ = 16k params (1.6% of original).

- **Memory win:** LoRA drastically reduces gradients and optimizer state size – no need to update all 1M params, just 16k. The **frozen base weights** also act as a regularizer, preventing overfitting to small data. LoRA fine-tuning has shown *no loss in performance* vs full fine-tuning in many cases. **It's the backbone of QLoRA** (QLoRA uses LoRA for all major weight matrices).

Quantization Recap – Why & How (1)

- **Why quantize?** Memory scales with precision. 16-bit weights → 4-bit = 4× less memory. Enables a 65B model (which is ~130GB in FP16) to be ~33GB in 4-bit. Fits on 48GB GPU with room for other data.



$$s = \frac{2^{b-1}-1}{\alpha} \quad \text{(scale factor)}$$

$$x_{quantized} = \text{round}\left(s \cdot x\right) \quad \text{(quantization)}$$

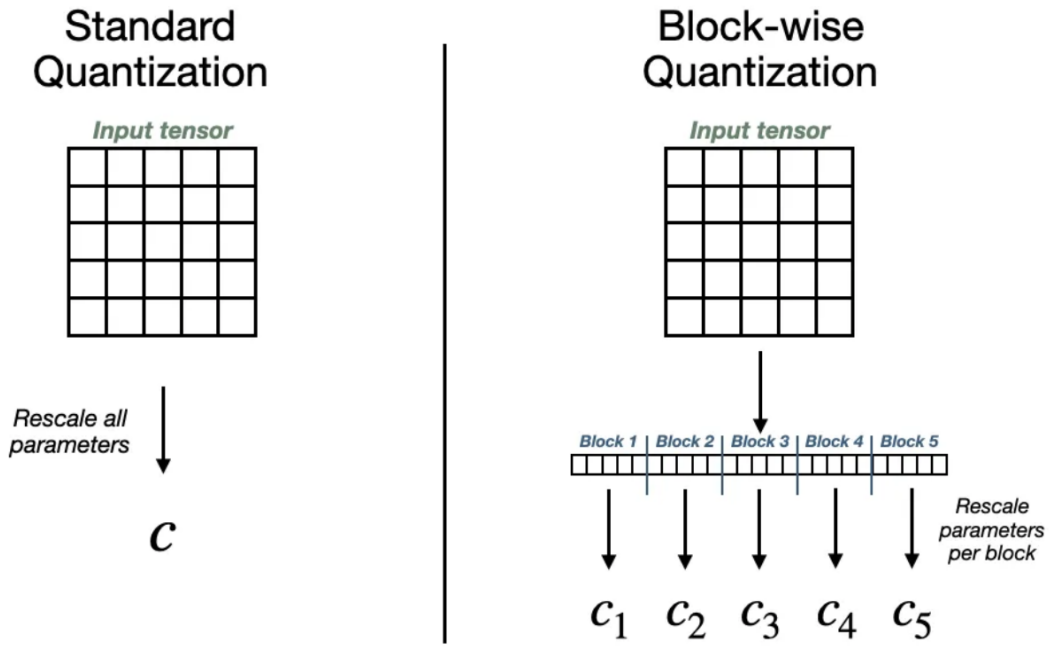
$$s = \frac{127}{10.8} = 11.76 \quad \text{(scale factor)}$$

$$x_{quantized} = \text{round}\left(11.76 \cdot \blacksquare\right) \quad \text{(quantization)}$$

$$x_{dequantized} = \frac{\blacksquare}{s} \quad \text{(dequantize)}$$

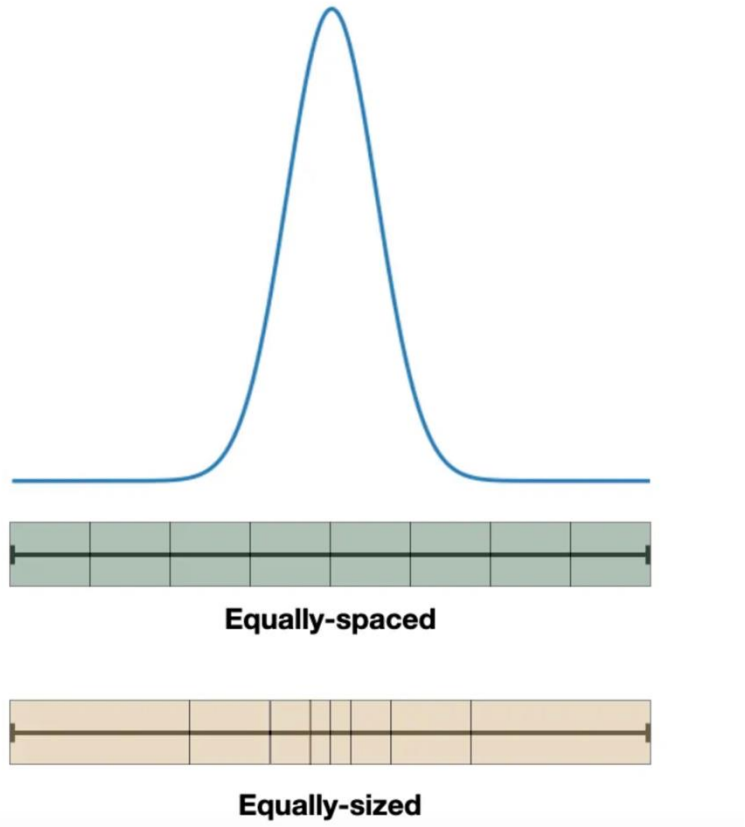
Quantization Recap – Why & How (2)

- **Block-wise quantization:**
  Quantize weights in small blocks. This addresses outliers by giving each block its own dynamic range. (Prevents one extreme weight from blowing precision for others in a large layer.)



NormalFloat4 (NF4) – 4-bit Optimal Quantization (1)

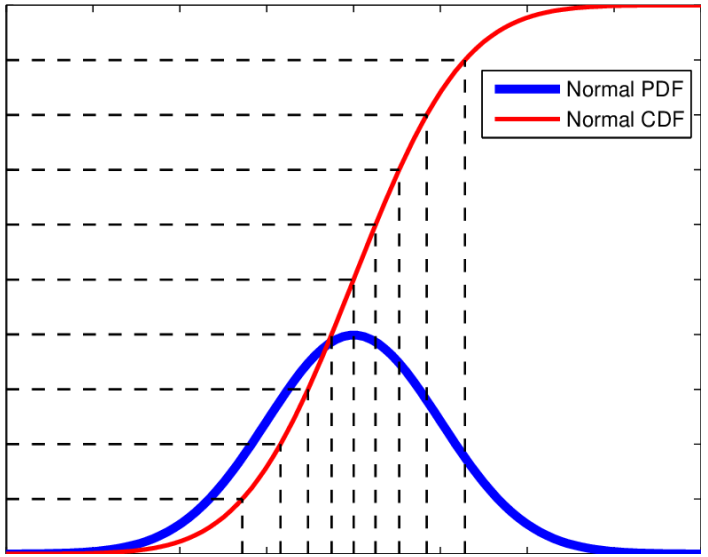
- **Straight int4 vs. improved:**
  Naive int4 (16 evenly spaced levels) can hurt accuracy significantly for LLMs. We need a smarter 4-bit scheme to maintain quality. That's where **NF4** comes in.
- **Observation:** Pre-trained weight distributions are roughly normal (Gaussian) around 0. Many values near 0, few at extremes. But a uniform 4-bit quantizer assumes equal probability per level – wasteful!
- **NF4 = NormalFloat 4-bit:** quantization levels are *density-aware*. Each of the 16 possible values represents an equal slice of the normal distribution.
  - Bins are **narrow near 0** (finer precision where weights cluster) and **wider outwards** (covering tails)
  - Since model parameters tend to clump around 0, this is an effective strategy for representing LLM parameters.



NormalFloat4 (NF4) – 4-bit Optimal Quantization (2)

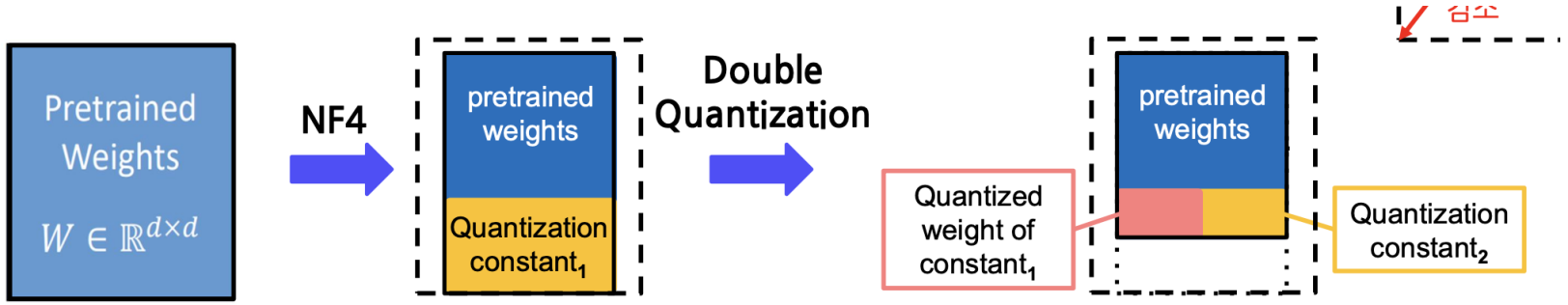
- **How it's derived:** Compute thresholds from Normal(0,1) CDF so that $P(weight\ in\ bin_i) = 1/16$. a nonlinear mapping that "compresses" the normal curve into 16 values.

- **Benefit:** NF4 greatly reduces quantization error for weights. It's **information-theoretically optimal** for normally distributed data.
- NF4 improves performance significantly over FP4 and Int4



Double Quantization – Compressing the Scales (1)

- After 4-bit quantization, each weight block requires a scale factor (and possibly bias) for dequantization, **typically stored in 16-bit or 32-bit**, adding memory overhead (~0.5 bits per weight for 32-bit scale per 64 weights). For large models, this overhead can reach gigabytes.
- **DQ (Dequantization Quantization)** reduces this overhead by quantizing the scale values (e.g., to 8-bit), optimizing memory usage while keeping core weights at 4-bit precision. This enables fitting larger models into limited GPU memory (e.g., 33B on 24GB, 65B on 48GB).
- **The accuracy impact** is negligible since 8-bit is sufficient for scales. Experiments showed no measurable performance drop.



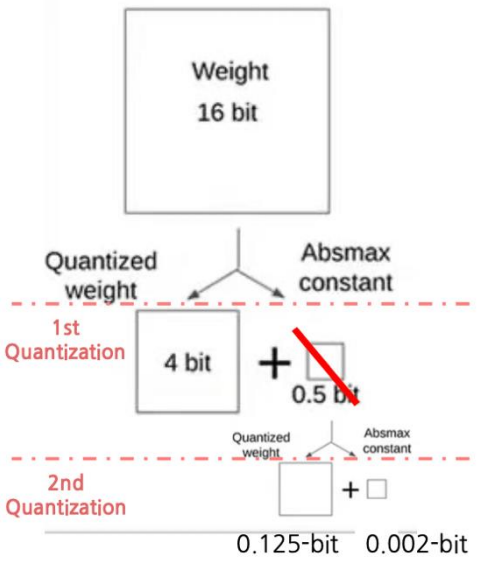
Double Quantization – Compressing the Scales (2)

- **Double Quantization (DQ):** *Quantize the quantization constants.* Apply a second layer of quantization to the scales themselves.
  - In QLoRA: First-level scales for each 64-weight block are collected. Then groups of these scales (block of 256 scales) are quantized into 8-bit floats ($c_{\{1,FP8\}}$), with one 32-bit "scale of scales" (c_2) per group.
  - Essentially: **4-bit weights + 8-bit scale per block + 32-bit super-scale per 256 blocks.**

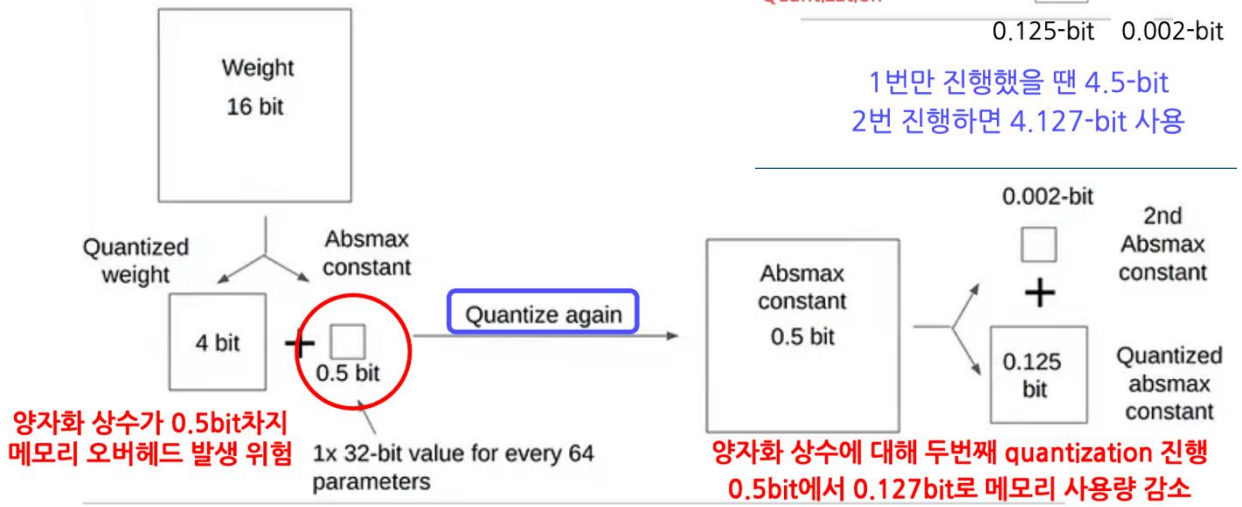
- **Memory math:**

  Without DQ: 4-bit weight + 32-bit scale/64 = 4 + 0.5 = 4.5 bits per weight.
  With DQ: 4-bit weight + 8-bit scale/64 + 32-bit per 256 blocks = 4 + 0.125 + 0.002 = 4.127 bits.

  Saves ~0.37 bits/weight (about 8% relative reduction, ~3GB for 65B model).



1번만 진행했을 땐 4.5-bit
2번 진행하면 4.127-bit 사용



Paged Optimizers – Avoiding Out-of-Memory (OOM)

- **Memory Spikes:** Fine-tuning large models can cause memory spikes due to long sequences or gradient checkpointing.
- **Paged Optimizers:** They use GPU unified memory to automatically move optimizer state tensors (e.g., Adam's moments) from GPU to CPU RAM when GPU memory is nearly full.
- This mechanism, enabled via the is_paged=True setting in bitsandbytes optimizer, maximizes GPU usage and handles variable input lengths without manual adjustments.

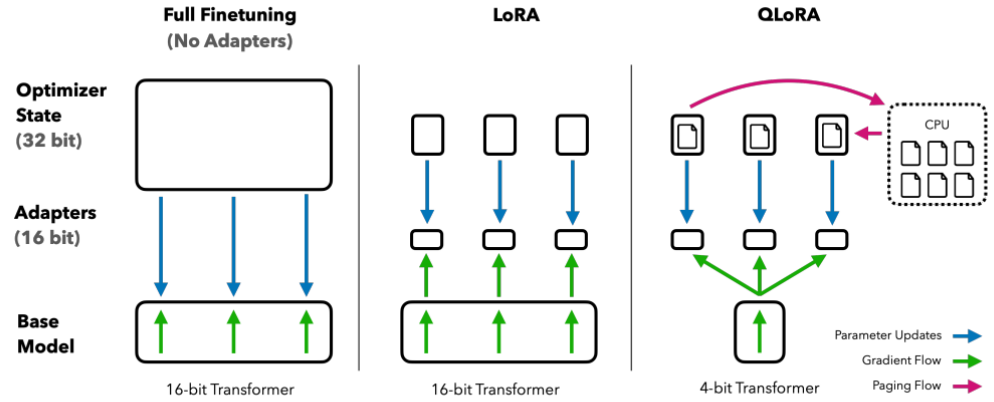
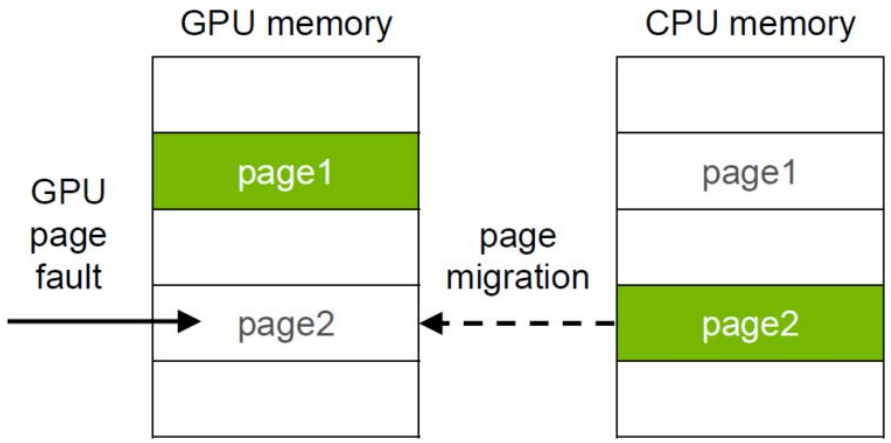
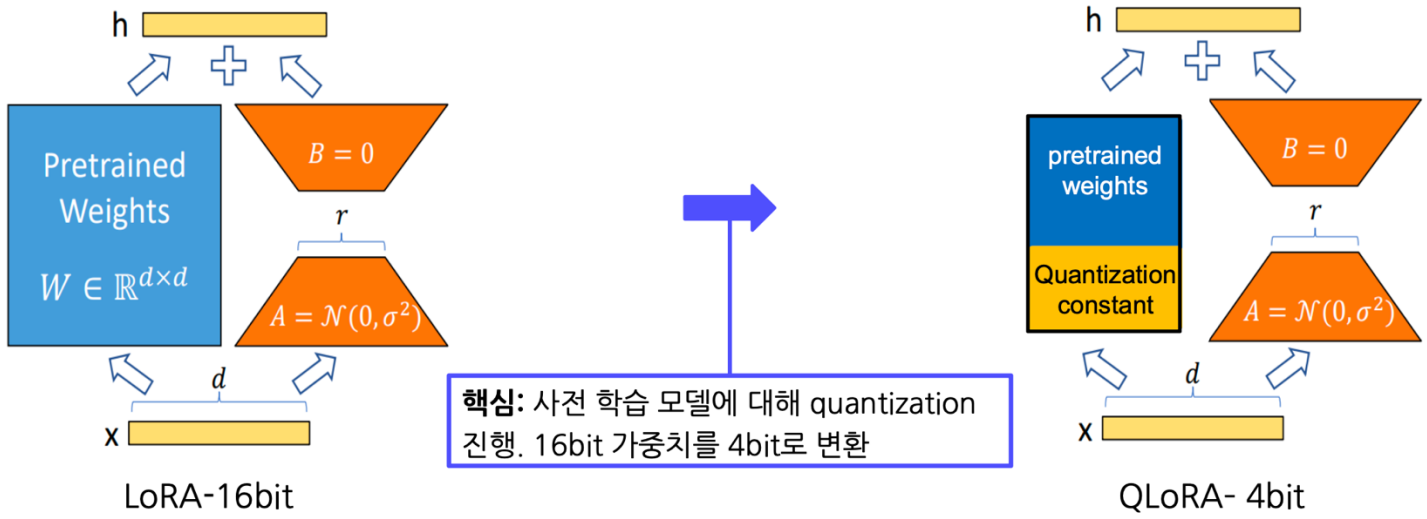


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.



QLoRA Training Pipeline

- **Storage vs Compute Dtypes:** QLoRA uses *4-bit NF4 for storage* of base model, and *16-bit (BF16) for computation*. During forward, weights are **dequantized on the fly** to BF16 for matrix multiplies. Backward computes grads only for LoRA (BF16) – no grad needed for 4-bit weights.
- **LoRA integration:** Each layer: output = (Quantized $W * x$) + $(B \times A * x)$ (LoRA). Only A, B get updated. After training, final weight = $W_0 + BA$ (which one can merge for inference).
- **Memory layout:** All 65B weights in 4-bit (≈33GB). LoRA params ~0.2% of that (tens of MBs). Optimizer states only for LoRA (2x or 3x LoRA params). Activations in BF16 (some GBs depending on sequence). Paged optimizer can offload some activations/moments if needed. The total stays within 48GB.



Performance Benchmarks

- **Quality vs Baselines:** QLoRA (4-bit) = **Full 16-bit finetune ≈ 16-bit LoRA finetune** on multiple tasks. No significant drop in accuracy observed up to 65B scale.
  - *GLUE (RoBERTa-large):* QLoRA achieved 88.8 avg score vs 88.6 full FT(essentially identical).
  - *NI tasks (T5-XL):* QLoRA Rouge-L 55.4 vs 54.3 full FT (within noise).
  - *MMLU (LLaMA models):* NF4+DQ QLoRA fully recovered 16-bit LoRA performance at 7B,13B,33B,65B. FP4 QLoRA was ~1% lower. For 65B, ~63% accuracy across Alpaca/Flan tuning, matching baseline. (Quote: "NF4 + DQ matches BF16 performance, FP4 consistently one point behind".)
- **NF4 vs FP4 Impact:** On zero-shot tasks (HellaSwag, Winogrande, etc.), 4-bit NF4 significantly outperformed plain 4-bit float/int. *Chart:* Mean zero-shot accuracy (Y) vs data type (X) – NF4 almost as high as 16-bit, FP4 clearly lower. Double quantization didn't boost accuracy but didn't hurt either, it mainly reduced memory.
- **State-of-the-art chat model:** Guanaco-65B (QLoRA on LLaMA65B, OASST1 data) scored 99.3 vs ChatGPT 100 on Vicuna benchmark. It surpassed all other open models at the time. This shows QLoRA can unlock top-tier performance *using smaller models with good data*. Guanaco-33B was ~97% of ChatGPT – even a 33B model fine-tuned with QLoRA can nearly match a 175B dialogue model given high-quality finetune data.
- **Efficiency metrics:** QLoRA reduced finetune memory by ~**>10×**. E.g., fine-tuning LLaMA-65B with LoRA in 16-bit required ~780GB (!), but QLoRA fit in 48GB. That's an extreme case: 16× reduction. More generally, 4-bit is 4× smaller than 16-bit, plus DQ ~8% more, plus LoRA not updating all weights – overall huge savings.
  - For 7B and 13B models, QLoRA means you can fine-tune on a 8GB or 12GB GPU (which was impossible with full FT). For 30B+, you need ~24GB (instead of multi-GPU). This greatly broadens who can fine-tune LLMs.
- **Hyperparameters:** They found to match full FT on larger models, LoRA needed to be applied to more layers (not just attention). Once they did LoRA on "All" linear layers, performance matched 16-bit. LoRA rank could remain small (e.g. 8 or 16). No special learning rate or trick was needed for QLoRA beyond what's used for LoRA normally.

Conclusions

- **QLoRA = Finetuning democratized:** It enables fine-tuning huge models on accessible hardware with *no* performance penalty. This is a big step for open-source AI – researchers can experiment with 30B+ models without needing a GPU farm.
- **Technique synergy:** The success comes from combining: LoRA (efficient training), NF4 (minimal info loss quantization), Double Quant (memory tuning), and paging (memory safety). Each component addresses a different challenge (compute, accuracy, memory overhead, stability). Together they make 4-bit training not only possible but effective.
- **Evaluation & Data:** An insight from QLoRA experiments – smaller models can punch above their weight if fine-tuned on high-quality data (Guanaco 33B ~ ChatGPT level on some benchmarks). This suggests a strategy: use QLoRA to cheaply try many fine-tuning datasets on a base model, find which yields best outcome, etc. QLoRA makes that kind of extensive experimentation feasible (they fine-tuned 1000+ models as ablations). It's a tool for both research and production (e.g., customizing a model on a domain-specific dataset on one GPU).

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization
- https://www.researchgate.net/figure/Partition-of-the-normal-probability-distribution-function-normal-PDF-thick-blue-line_fig1_276900209
- https://www.boostcourse.org/boostcampaitech7/joinLectures/391802?null
- https://medium.com/towards-data-science/qlora-how-to-fine-tune-an-llm-on-a-single-gpu-4e44d6b5be32