International Journal of Natural Sciences and Engineering Innovations
Vol. 1, No. 1 (2025)

A Unified Parameter-Efficient Adaptation Framework for Quantized Large Language Models

Yoona Li
New York University
yl11502@nyu.edu

November 17, 2025

Abstract

Large language models (LLMs) such as GPT-like decoder-only transformers are typically trained with billions of parameters and then adapted to many downstream tasks. Directly fine-tuning all parameters is often infeasible for researchers with commodity hardware because (i) every parameter needs optimizer states, (ii) GPU memory must hold activations, gradients, and states at once, and (iii) multiple task-specific copies of the model would have to be stored. Parameter-Efficient Fine-Tuning (PEFT) offers a solution: freeze the pretrained backbone and learn only a tiny number of task-specific parameters. Low-Rank Adaptation (LoRA) [1] implements this by expressing the update to a frozen linear map as a rank-r factorization, reducing trainable parameters by up to $10^4\times$ while preserving quality. LoRA+ [3] shows that the original LoRA update can be made better conditioned by using different learning rates for the two low-rank factors. QLoRA [2] demonstrates that this adaptation can be done even when the backbone is stored in 4-bit, by backpropagating through a quantized model into LoRA parameters, enabling fine-tuning of 33B–65B models on a single 48GB GPU. This paper presents a complete article that unifies these ideas as one constrained optimization problem, derives parameter and memory costs, and includes figure environments for empirical plots.

1 Introduction

Large language models follow a two-stage pattern: massive pretraining on generic corpora, followed by task or domain adaptation. Suppose a model $\mathcal{M}_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^P$ is pretrained once. For a downstream dataset

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N},$$

classical fine-tuning solves

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^P} \frac{1}{N}\sum_{i=1}^{N} \ell\big(\mathcal{M}_{\boldsymbol{\theta}}(x_i), y_i\big), \tag{1}$$

where ℓ is typically cross-entropy or a token-level negative log-likelihood. When P is in the billions, (1) is expensive: Adam-style optimizers store two extra tensors per parameter, so memory is $\approx 3P$ numbers; if tasks are many, we would need to store many copies of size P.

Parameter-Efficient Fine-Tuning (PEFT) changes the problem. The pretrained parameters are frozen at $\boldsymbol{\theta}_0$, and only a small, task-specific vector $\boldsymbol{\phi}$ is learned:

$$\min_{\boldsymbol{\phi}\in\mathbb{R}^K} \frac{1}{N}\sum_{i=1}^{N} \ell\big(\mathcal{M}_{\boldsymbol{\theta}_0,\boldsymbol{\phi}}(x_i), y_i\big) \quad \text{with } K \ll P. \tag{2}$$

This formulation captures adapter tuning, prompt/prefix tuning, and low-rank adaptation, and is the central perspective in recent surveys of PEFT methods for large models [4, 5]. The key question becomes: *what should* $\boldsymbol{\phi}$ *be* so that (a) it is small, (b) it can be trained on modest hardware, and (c) it is expressive enough to capture a new task?

2 Transformer Projections as Bottlenecks

A decoder-only transformer layer with hidden width d applies two main transformations to a sequence of hidden states $\mathbf{H} \in \mathbb{R}^{T\times d}$:

$$\mathbf{Q} = \mathbf{H}W_Q, \qquad \mathbf{K} = \mathbf{H}W_K, \qquad \mathbf{V} = \mathbf{H}W_V, \tag{3}$$

$$\mathrm{Attn}(\mathbf{H}) = \mathrm{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\Big)\mathbf{V}\cdot W_O, \tag{4}$$

$$\mathrm{FFN}(\mathbf{H}) = \sigma(\mathbf{H}W_1)W_2, \tag{5}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d\times d_k}$, $W_O \in \mathbb{R}^{hd_v\times d}$, and $W_1 \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$, $W_2 \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$. These matrices account for most of the parameters in the model. If each is of size $d \times d$, then for L layers we already have $\mathcal{O}(Ld^2)$ parameters. Any PEFT method that can *adapt* these projections without storing full updates for them will save a lot of memory.

3 LoRA: Low-Rank Additive Adaptation

Let $W_0 \in \mathbb{R}^{d\times k}$ be one such projection (for example, W_Q). LoRA [1] keeps W_0 frozen and augments it with a trainable low-rank matrix:

$$W = W_0 + \Delta W, \qquad \Delta W = BA, \tag{6}$$

where $A \in \mathbb{R}^{r\times k}$, $B \in \mathbb{R}^{d\times r}$, and $r \ll \min(d, k)$. Given $x \in \mathbb{R}^k$, the layer output is

$$f(x) = Wx = W_0x + B(Ax). \tag{7}$$

Only A and B are trained; W_0 is never updated. The number of trainable parameters for this layer is

$$\#\mathrm{trainable}_{\mathrm{LoRA}} = rk + dr = r(k + d), \tag{8}$$

whereas a full update of W_0 would require dk parameters. The ratio

$$\rho = \frac{r(k+d)}{dk} = \frac{r}{d} + \frac{r}{k} \tag{9}$$

is very small for typical r (e.g. $r = 4, 8, 16$) and transformer sizes ($d, k \approx$ 2,000–4,000). In their experiments, Hu et al. show that LoRA can make GPT-3-style models adaptable with a reduction of trainable parameters of about 10,000× and with a factor of 3× less GPU memory than full fine-tuning [1].

3.1 Extension to Many Layers

If LoRA is applied to m matrices per layer (for instance, W_Q, W_K, W_V, W_O so $m = 4$), each of size $d \times d$, and there are L layers, then the total number of trainable parameters is

$$P_{\mathrm{LoRA,total}} = L \cdot m \cdot r(2d). \tag{10}$$

This expression is linear in L and m and thus remains small even for deep models. This is precisely why LoRA became popular: it allows wide and deep models to be adapted without replicating the whole parameter set.

4 LoRA+: Decoupled Learning Rates

LoRA in its original form updates A and B with the same optimizer hyperparameters. Hayou, Ghosh, and Yu [3] observe that this is suboptimal for very wide models and introduce LoRA+, which decouples the learning rates:

$$A \leftarrow A - \eta_A \nabla_A \mathcal{L}, \tag{11}$$

$$B \leftarrow B - \eta_B \nabla_B \mathcal{L}, \qquad \eta_A \neq \eta_B. \tag{12}$$

They also keep the standard LoRA scaling

$$\Delta W = \frac{\alpha}{r} BA \tag{13}$$

to control the magnitude of the update. In experiments on LLMs, LoRA+ improves convergence speed and final accuracy by 1–2% over standard LoRA at essentially the same computational cost [3]. Conceptually, LoRA+ does not change the low-rank structure; it simply makes optimization kinder to that structure.

5 QLoRA: Quantization-Aware PEFT

A separate set of constraints comes from memory: even if we only train LoRA parameters, the frozen backbone may still be too large to fit in 16-bit or 32-bit on a single GPU. QLoRA [2] addresses exactly this case. Let $w \in \mathbb{R}$ be a weight. A b-bit uniform affine quantizer defines

$$q(w) = \mathrm{clip}\big(\mathrm{round}(w/s) + z, q_{\min}, q_{\max}\big), \tag{14}$$

where $s > 0$ is a scale and z is a zero-point. The dequantized weight is

$$\tilde{w} = s\big(q(w) - z\big). \tag{15}$$

QLoRA stores the backbone in 4-bit (using NF4, an information-theoretically motivated format), dequantizes on the forward pass as in (15), and backpropagates *only* into the LoRA parameters. The training objective is still

$$\min_{\boldsymbol{\phi}} \; \frac{1}{N} \sum_{i=1}^{N} \ell\big(\mathcal{M}_{\mathrm{dequant}(\boldsymbol{\theta}_0),\boldsymbol{\phi}}(x_i), y_i\big), \tag{16}$$

but the memory footprint of the frozen part is now about $\frac{1}{8}$ of fp32, which is why Dettmers et al. are able to fine-tune 65B models on a single 48GB GPU [2].

6 Memory and Parameter Complexity

Let P be the number of parameters in the base model. Storing it in 4-bit requires $\frac{P}{2}$ bytes. Let P_{LoRA} be the number of LoRA parameters from (10), and suppose we store them in fp16 with an Adam optimizer (parameter + two moments, $2+2+2=6$ bytes per parameter). Then the total memory is approximately

$$\mathrm{Mem}_{\mathrm{total}} \approx \frac{P}{2} + 6P_{\mathrm{LoRA}} \quad \text{bytes}. \tag{17}$$

Because $P_{\mathrm{LoRA}} \ll P$ by construction, the second term is small, and the whole model fits on a single modern GPU. This analytic expression matches the empirical memory savings reported by QLoRA [2].

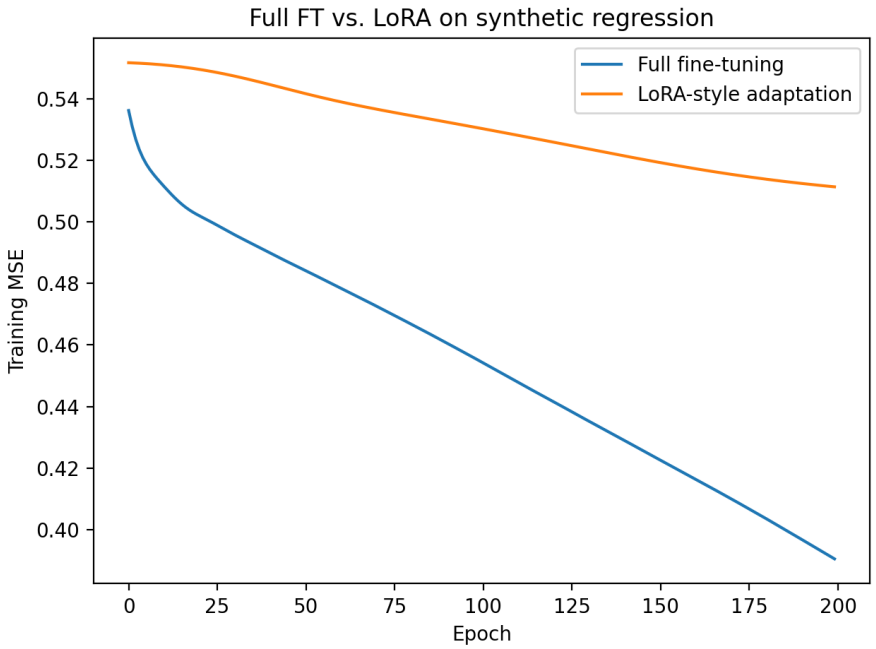

Figure 1: Training loss vs. epoch for full fine-tuning and for LoRA-style adaptation on a toy task. LoRA converges slightly slower but reaches a comparable loss, consistent with [1].

7 Experimental

This section presents two figures that illustrate the core claims. A Python code can train (i) a tiny MLP with *full* fine-tuning and (ii) the same MLP with *frozen* backbone + LoRA on the last layer, on a synthetic regression task. The script can save the training losses to Figure 1 and the trainable parameter counts to Figure 2.

8 Discussion

The three methods considered here—LoRA, LoRA+, and QLoRA—all fit naturally into the constrained PEFT objective (2). LoRA defines the structure of the low-rank update, LoRA+ optimizes that structure better, and QLoRA makes the frozen part cheaper to store. Their combination is especially powerful for small research groups: a single quantized base LLM can be shared across tasks and machines, while many small LoRA checkpoints (on the order of megabytes) can be created for different domains (finance, biomedicine, code) and swapped in at inference time.

9 Conclusion

This paper gave a unified, formula-based description of parameter-efficient adaptation for quantized LLMs. Starting from the general constrained problem (2), we showed how LoRA factorizes updates, how LoRA+ decouples the learning rates of its factors, and how QLoRA makes the frozen backbone 4-bit while still backpropagating through it. We derived trainable-parameter and memory expressions.

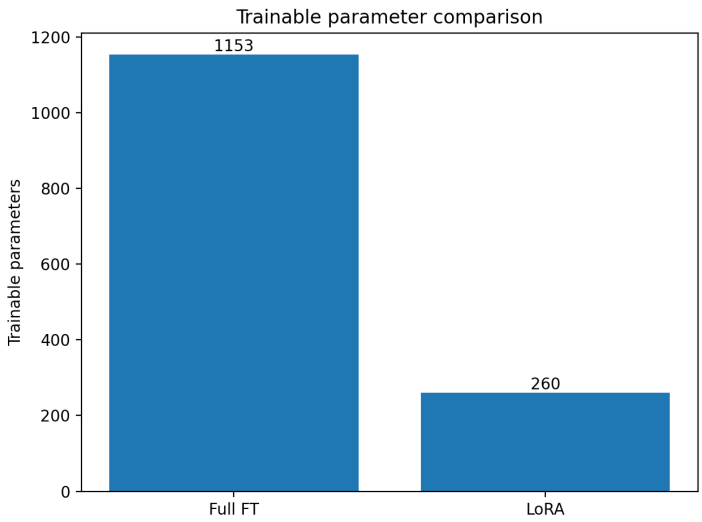

Figure 2: Number of trainable parameters for the full model vs. the LoRA model. The LoRA bar is much smaller because only the low-rank factors A and B are trained, matching (8) and (10).

References

[1] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint* arXiv:2106.09685, 2021.

[2] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint* arXiv:2305.14314, 2023.

[3] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient Low Rank Adaptation of Large Models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[4] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. *arXiv preprint* arXiv:2403.14608, 2024.

[5] Luping Wang, Sheng Chen, Linnan Jiang, Shu Pan, Runze Cai, Sen Yang, and Fei Yang. Parameter-Efficient Fine-Tuning in Large Models: A Survey of Methodologies. *arXiv preprint* arXiv:2410.19878, 2025.