Training Efficiency Comparisons for LLM Fine-Tuning

Songchen Xue, Sanchit Sahay, Sruthi Raghavan
New York University
{sx2618, ss19723, sr7892}@nyu.edu

***Abstract*—Fine-tuning large language models demands massive GPU memory and compute, making it out of reach for most teams without expensive infrastructure. We built a reproducible benchmarking framework on GSM8K using Qwen2.5-1.5B-Instruct, comparing LoRA, QLoRA, GoRA, and Gradient Checkpointing. We also use the NVIDIA Nsight Systems suite of tools to compare DDP and FSDP across full fine-tuning and PEFT methods. Our results show that GoRA achieves roughly 15% higher throughput than LoRA with comparable accuracy, QLoRA reduces GPU memory by 7.3% with a slight accuracy drop, and gradient checkpointing cuts GPU memory usage by approximately 57% at the cost of training speed. For distributed training, FSDP uses significantly less peak memory and NCCL communication time than DDP in full fine-tuning, but this difference is less apparent and even reverses for PEFT workflows. Practitioners get a concrete, data-backed tradeoff map of memory vs throughput vs accuracy to pick the right training method for their hardware constraints.**

***Index Terms*—large language models, parameter-efficient fine-tuning, LoRA, QLoRA, GoRA, gradient checkpointing, distributed training, DDP, FSDP**

I. Introduction

A. Background and Motivation

As language models keep getting larger, fine-tuning them has turned into a serious resource problem. Full fine-tuning of a model with ϕ parameters using mixed-precision and Adam requires at minimum 16ϕ bytes of memory just for parameters, gradients, and optimizer states, not counting activations [3]. For a lot of teams without multi-GPU clusters, this is not feasible.

This has led to a growing interest in parameter-efficient fine-tuning (PEFT) methods. LoRA [1] is the most widely used approach. It freezes pretrained weights and adds small trainable low-rank matrices to the attention layers. QLoRA [2] goes further by quantizing the frozen weights to 4 bits. GoRA [3], a recent method published at NeurIPS 2025, introduces gradient-driven adaptive rank allocation. Separately, gradient checkpointing [4] saves activation memory by recomputing activations instead of storing them during the backward pass.

Each of these methods has been benchmarked in its own paper on different models, datasets, and hardware. So if you are trying to decide what to use in practice, it is hard to get a direct comparison. On top of that, the interaction between PEFT and distributed strategies like DDP [5] and FSDP [6] has not been explored much for smaller models that people actually fine-tune under limited budgets.

B. Problem Statement

We wanted to answer two questions:

Q1. Given a specific large language model and limited GPU resources, which PEFT and Quantization methods should be picked.

Q2. With access to multiple GPUs, which distributed training strategy is more efficient between DDP and FSDP.

C. Objectives and Scope

Our goals were: (1) Build a config-driven, reproducible experiment framework with versioned outputs and structured logging, (2) Benchmark LoRA, QLoRA, GoRA, and gradient checkpointing on the same model and dataset, and (3) Compare DDP vs FSDP under weak scaling across 1, 2, and 4 GPUs with profiler-guided analysis using NVidia's Nsight Systems.

We used Qwen2.5-1.5B-Instruct [9] as our base model. Under limited resources, we excluded full fine-tuning as a standalone baseline because it was substantially more expensive in terms of time and memory at our target scale. Using LoRA as the baseline still kept our initial goals intact while making the study feasible.

II. Literature Review

A. Review of Relevant Literature

LoRA. Hu et al. [1] proposed Low-Rank Adaptation, which decomposes weight updates into two small matrices $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$ where r is much smaller than the original dimensions. The core idea is that weight changes during fine-tuning tend to live on a low-dimensional subspace, so constraining updates this way does not lose much. They report up to 10,000$\times$ fewer trainable parameters compared to full fine-tuning while matching quality on tasks like RoBERTa, DeBERTa, and GPT-3 [1]. We used LoRA as our main baseline.

QLoRA. Dettmers et al. [2] extended LoRA by quantizing the frozen pretrained weights to 4 bits using NormalFloat (NF4). They also introduced double quantization to reduce the overhead of quantization constants, and paged optimizers to handle memory spikes. Their main result was fine-tuning a 65B model on a single 48GB GPU while matching 16-bit fine-tuning quality. We wanted to see if the memory savings would still be meaningful on a smaller 1.5B model.

GoRA. He et al. [3] proposed GoRA at NeurIPS 2025, which tackles both rank selection and weight initialization in a

single framework. It uses gradient information during training to dynamically assign ranks and initialize adapter weights. On Llama3.1-8B they reported a 5.13-point improvement over LoRA on GSM8K. We were interested in whether the throughput and utilization gains would hold on a smaller model.

Gradient Checkpointing. Chen et al. [4] introduced the idea of selectively storing activations at checkpoint boundaries and recomputing the rest during backpropagation. This reduces activation memory from $O(n)$ to $O(\sqrt{n})$ at the cost of extra compute.

DDP and FSDP. PyTorch DDP [5] replicates the full model on each GPU and synchronizes gradients with all-reduce after each backward pass. FSDP [6] shards parameters, gradients, and optimizer states across devices, similar to ZeRO [7]. FSDP uses less per-device memory but requires more communication in the form of all-gather and reduce-scatter operations.

Profiling. We used NVIDIA Nsight Systems [10] for timeline-based profiling of GPU workloads, including exporting to SQLite databases for programmatic analysis of NCCL kernels and collective operations.

B. Identification of Gaps

Most papers benchmark their method against one or two baselines on their own hardware. We could not find a study that puts LoRA, QLoRA, GoRA, and gradient checkpointing side by side in a controlled setup. The DDP vs FSDP comparison also tends to focus on full fine-tuning of large models, and how it works with PEFT has not been explored in detail.

III. METHODOLOGY

A. Data Collection and Preprocessing

We used GSM8K [8], which has about 8,500 grade-school math word problems with step-by-step solutions. We chose this because math reasoning is non-trivial for a 1.5B model, so differences between methods show up clearly. We built a deterministic data pipeline with instruction formatting and tokenization for both training and evaluation splits, using a max token length of 256. We used the standard train/test split and formatted prompts consistently across methods. Performance turned out to be quite sensitive to prompt and training configurations, which required careful design and tuning.

B. Model Selection

We used Qwen2.5-1.5B-Instruct [9]. At 1.5B parameters it is large enough to be realistic but small enough that we could run all our experiments. We originally planned to use full fine-tuning as our primary baseline, but dropped it because full fine-tuning at our target scale was substantially more expensive in terms of time. Using LoRA as the baseline kept our goals intact while being more feasible to train.

C. Optimization Procedures

We tested four approaches:

LoRA (Baseline). Parameter-efficient fine-tuning via low-rank adapters on the attention layers. In our initial baseline run on 1x L4 GPU with batch size 8 and max token length 256, LoRA had 2,179,072 trainable parameters, used 3139.99 MB peak GPU memory, achieved 931.9 tokens/sec throughput, 0.446 s/step training time, and reached 52.67% eval accuracy with a training loss of 0.3646.

QLoRA. Extends LoRA with 4-bit quantization for reduced memory usage. The pretrained weights are stored in NF4 format and gradients flow through the quantized model into the LoRA adapters.

Gradient Checkpointing. Trades additional compute for lower memory by recomputing activations during the backward pass instead of storing them. This is mathematically equivalent to normal training so it does not affect accuracy.

GoRA. Gradient-based low-rank adaptation that dynamically updates important weight subspaces [3]. It uses gradient information to adaptively assign ranks and initialize adapter weights during training.

For distributed training, we compared DDP and FSDP under weak scaling across 1, 2, and 4 GPUs with matched per-GPU batch size, including memory-stress settings to measure runtime, per-device memory, and scaling behavior.

D. Profiling Tools and Methods

We used the NVIDIA Nsight Systems suite for all profiling. We collected Nsight timeline traces to see how training steps break down between compute, communication, and idle time. We exported traces to SQLite databases to query NCCL kernel share, collective breakdowns, and device-to-device data movement programmatically.

Timeline tools show wall-clock regions while kernel summaries show actual GPU work. We had to be careful to separate compute, communication, and synchronization overhead, since a training step that looks slow in the timeline might actually have high GPU utilization with the wall-clock time dominated by a synchronization barrier rather than wasted compute.

E. Evaluation Metrics

We tracked per-step timing, tokens/sec, samples/sec, and peak GPU memory consistently across all experiments for fair comparison. For memory we recorded per-device peak memory (worst-case footprint) and end-of-run allocated memory (steady-state footprint). For distributed runs we measured weak-scaling throughput across 1, 2, and 4 GPUs, along with NCCL kernel share, collective breakdown, and device-to-device data movement from Nsight Systems analysis.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

We built a shared benchmarking core to make sure experiments are directly comparable and reproducible. This includes config-driven runs with versioned output directories, structured logging, a deterministic GSM8K data pipeline, per-step efficiency instrumentation, and smoke-test validation scripts to verify end-to-end execution before committing to full runs. We ran the initial LoRA baseline on 1x L4 GPU

to validate the pipeline, then moved to A100 GPUs for the full comparison runs. Our code is publicly available at https://github.com/brian-xue/HPML-proj. Experiments were fully conducted on NVIDIA A100 40GB GPUs. The learning rate and LoRA rank were set to 1e-4 and 16, respectively.

B. PEFT Method Comparison

Table I summarizes the main results.

TABLE I
PEFT METHOD COMPARISON ON QWEN2.5-1.5B-INSTRUCT / GSM8K

Method	Key Advantage	Tradeoff
LoRA	Fastest loss reduction	Baseline
GoRA	∼15% throughput ↑	Comparable accuracy
QLoRA	7.3% GPU memory ↓	Slightly lower accuracy
Grad. Ckpt	∼57% GPU memory ↓	Lower efficiency

LoRA had the fastest loss reduction among all methods and served as our baseline. In our initial pipeline validation, LoRA on Qwen2.5-1.5B with 18464768 trainable parameters achieved 10965.1 tokens/sec, a peak GPU memory of 24988.9 MB, and 52.67% eval accuracy on GSM8K with 0.3646 training loss.

GoRA vs LoRA. As shown in Table II, GoRA achieves comparable accuracy to LoRA while delivering approximately 15% higher throughput. The kernel-level Nsight Compute profiling results suggest that this improvement is primarily associated with more effective GPU utilization. In particular, GoRA increases compute utilization from 4.2% to 70.3% and average memory utilization from 3.5% to 49.1%, while maintaining a nearly unchanged peak memory footprint. One possible explanation is that GoRA's gradient-driven rank allocation assigns larger ranks to more important layers, making the training workload better aligned with efficient GPU kernel execution. Consistent with this interpretation, the NCU profiles show that GEMM- and attention-related kernels are more prominently exercised in GoRA, whereas LoRA is dominated by low-utilization elementwise operations. As a result, GoRA can concentrate model capacity on the most influential weight subspaces without increasing the overall parameter budget. These results highlight that meaningful training speedups can be achieved not only through algorithmic changes, but also through improved hardware utilization under a similar memory footprint.

LoRA vs QLoRA. QLoRA reduced peak GPU memory by 7.3% compared to LoRA by quantizing the frozen pretrained weights to 4 bits. The accuracy was slightly lower since quantization introduces some information loss that the low-rank adapters cannot fully make up for. Throughput was roughly the same between the two because the dequantization overhead partially offsets the memory bandwidth savings. On a 1.5B model the 7.3% savings is more modest than what Dettmers et al. reported on larger models [2], which makes sense because the adapter weights and optimizer states take up a bigger fraction of total memory at this scale.

TABLE II
PERFORMANCE AND KERNEL-LEVEL PROFILING COMPARISON BETWEEN LORA AND GORA

Metric	LoRA	GoRA
End-to-end Training Performance		
Step Time (s)	0.373 ± 0.0017	$\mathbf{0.322 \pm 0.0007}$
Step Time CV (%)	0.45	**0.23**
Throughput (tokens/s)	10965.1 ± 23.3	$\mathbf{12697.9 \pm 28.6}$
Throughput CV (%)	**0.21**	0.23
Peak Memory (MB)	24988.9	**24859.9**
Kernel-Level Profiling (NCU)		
Compute Utilization (%)	4.2	**70.3**
Memory Utilization (%)	3.5	**49.1**
Achieved Occupancy (%)	∼6	10–25
Kernel Type	Elementwise	GEMM / Attention

Gradient Checkpointing. This gave the biggest memory reduction at roughly 57% lower GPU memory usage. Training is slower because activations have to be recomputed during the backward pass, but accuracy is not affected since checkpointing does not change what gets computed, just when.

The key observation is that QLoRA and checkpointing target different bottlenecks. QLoRA helps in the parameter-dominated regime by quantizing frozen weights. Checkpointing helps in the activation-dominated regime by avoiding activation storage. So QLoRA trades accuracy for memory, while checkpointing trades efficiency for memory. Due to the significant memory savings brought by gradient checkpointing, activations become the dominant source of GPU memory usage during training in our experiments.

C. Distributed Training: DDP vs FSDP

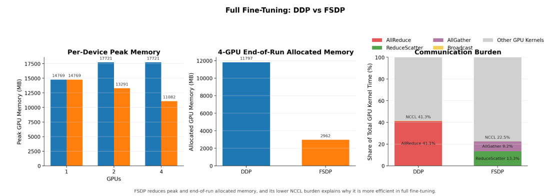

Fig. 1. Full fine-tuning comparison between DDP and FSDP. Left: per-device peak memory across 1, 2, and 4 GPUs. Middle: 4-GPU end-of-run allocated memory. Right: communication burden as a share of total GPU kernel time.

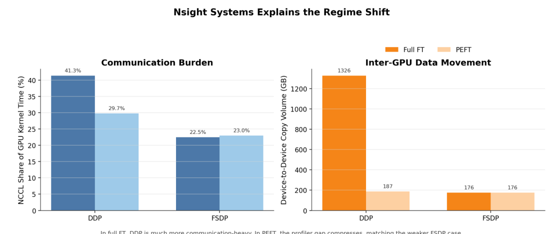

Fig. 2. Profiler summary explaining the regime shift.

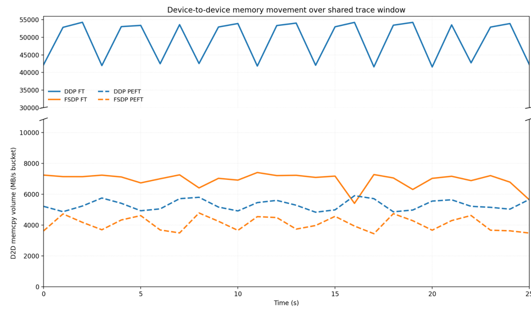

Fig. 3. Device-to-device memory movement over time for DDP and FSDP in full fine-tuning and PEFT.

We compared DDP and FSDP under weak scaling across 1, 2, and 4 GPUs. Table III shows the key observations.

We compared DDP and FSDP under weak scaling across 1, 2, and 4 GPUs, and then examined how the comparison changes between full fine-tuning and PEFT. The results show that the best distributed strategy depends strongly on the training regime.

For full fine-tuning (FT), FSDP was clearly more efficient than DDP. As shown in Figure 1, both methods achieve similar trraining behaviour but FSDP reduces per-device peak memory as GPU count increases. The difference in peak memory is around 4GB on 2 GPUs, and around 6 GB on 4 GPUs. The profiler view in Figure 1 also explains this difference: DDP spends 41.3% of total GPu kernel time in NCCL communication, while FSDP spends 22.5%, with DDP dominated by all-reduce.

However, this advantage does not carry over to PEFT. Figure 3 shows that in PEFT (LoRA) the memory gap becomes much smaller and can even reverse. At 4 GPUs, DDP PEFT uses about 8.6GB of peak memory, compared with about 10.1GB for FSDP PEFT. The same pattern is visible in 2 GPUs as well.

TABLE III
OBSERVED DDP VS FSDP RESULTS ON QWEN.

Metric	**DDP**	**FSDP**
Full FT peak memory, 2 GPUs (MB)	17720.8	13290.7
Full FT peak memory, 4 GPUs (MB)	17720.8	11082.2
Full FT step time, 4 GPUs (s)	0.202	0.182
Full FT train loss, 4 GPUs	0.3574	0.3575
Full FT NCCL share, 4 GPUs (%)	41.3	22.5
PEFT peak memory, 4 GPUs (MB)	8641.2	10060.2
PEFT step time, 4 GPUs (s)	0.239	0.299
PEFT throughput, 4 GPUs (tok/s)	13857.9	11254.4
PEFT train loss, 4 GPUs	0.3848	0.3847

The reason for this is that LoRA-style PEFT changes the communication requirements. When only a small adapter parameter set is trainable, DDP's replicated structure is no longer heavily penalized by synchronization since the state being shared is small. By contrast, FSDP still incurs sharding-related communication and coordination overhead. Figure 2 shows this directly. Under PEFT, NCCL communication burden falls sharply from 41.3% to 29.7% for DDP, while FSDP remains roughly flat.

Figure 3 provides a timeline view to this effect. In FT, DDP exhibits much larger sustained D2D memory movement. Under PEFT this falls to the FT FSDP range, while PEFT FSDP does not provide any gains.

D. *Analysis of Results*

System efficiency is largely determined by GPU utilization rather than raw model performance. GoRA's 15% throughput gain did not come from fewer parameters or less memory. It came from the GPU spending more of its time doing useful work thanks to fused kernels and better scheduling. Improving utilization, as GoRA demonstrates, can significantly boost throughput without increasing memory usage.

Memory optimization depends on the bottleneck. QLoRA addresses the parameter-dominated regime. Gradient checkpointing addresses the activation-dominated regime. GoRA does not really target memory at all but instead improves utilization, which is a different optimization axis entirely.

V. DISCUSSION

A. *Interpretation of Results*

One thing this project made clear to us is how much of training time goes to things besides useful computation: kernel launch overhead, synchronization barriers, communication between devices. Just looking at parameter counts or FLOP estimates does not capture this. The Nsight profiling was essential for understanding why GoRA is faster. It does not do less math, it does the same math more efficiently at the kernel level.

The DDP vs FSDP result also changed our thinking. We initially assumed FSDP would be better across the board since it is the newer approach. But for PEFT workloads on Qwen with a small number of trainable parameters, FSDP's communication overhead was not worth it. This is something the PyTorch documentation does not really emphasize.

B. *Comparison with Previous Studies*

Our LoRA results are consistent with Hu et al. [1]. The QLoRA memory savings (7.3%) are smaller than what Dettmers et al. [2] showed on 65B models, which is expected at the 1.5B scale. GoRA's throughput improvement aligns with He et al. [3], though their paper focuses more on accuracy and less on system-level analysis. Our DDP vs FSDP comparison extends prior work by Zhao et al. [6] by looking at how PEFT changes the picture.

C. *Challenges and Limitations*

Resource constraints. Under limited GPU resources, we used a 1.5B model and excluded full fine-tuning as a baseline. We carefully balanced training configurations to keep comparisons fair. On larger models, QLoRA's savings would likely be much more pronounced.

Efficiency tradeoff. Frequent evaluation improves tracking but increases runtime overhead. We wanted detailed loss curves but had to weigh that against inflating wall-clock time.

Configuration sensitivity. Performance depends heavily on prompt and training configurations. We spent time early on debugging results that turned out to be prompt formatting issues rather than method issues.

Profiling complexity. Timeline tools show wall-clock regions while kernel summaries show GPU work. Careful interpretation is needed to separate compute, communication, and synchronization overhead. This took more effort than we initially expected.

VI. Conclusion

A. Summary of Findings

We built a benchmarking framework and compared LoRA, QLoRA, GoRA, and gradient checkpointing on Qwen2.5-1.5B-Instruct with GSM8K, alongside a distributed training study of DDP vs FSDP.

LoRA provided the fastest loss reduction and served as a strong baseline. GoRA achieved comparable performance with approximately 15% higher throughput through better GPU utilization and fused kernels. QLoRA reduced GPU memory by 7.3% with a slight accuracy cost, targeting the parameter-dominated regime. Gradient checkpointing cut GPU memory by roughly 57% at the cost of efficiency, targeting the activation-dominated regime.

FSDP suits full fine-tuning with significantly lower peak memory and NCCL communication time. But for PEFT, DDP is more practical. The choice depends on the actual workload.

Lightweight PEFT methods enable practical fine-tuning on limited hardware. System efficiency is largely determined by GPU utilization rather than raw model performance, and improving utilization can boost throughput without increasing memory usage.

B. Contributions

We provide (1) a config-driven benchmarking framework with versioned outputs and structured logging for reproducibility, (2) a controlled comparison of LoRA, QLoRA, GoRA, and gradient checkpointing on the same model and dataset, (3) a distributed training study showing that FSDP vs DDP depends on whether you are doing full fine-tuning or PEFT, and (4) profiler-guided analysis using Nsight Systems with timeline and SQLite-based kernel-level analysis.

C. Recommendations for Future Work

Scaling to larger models (7B, 13B) would show how the tradeoffs shift with model size. Combining QLoRA with gradient checkpointing could yield compound memory savings. Testing on tasks beyond math reasoning would check if the rankings hold. Exploring FSDP with hybrid sharding might find a better fit for PEFT workloads. Adding power consumption and cost metrics would make the comparison more practical.

References

[1] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv preprint arXiv:2106.09685*, 2021.

[2] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[3] H. He, P. Ye, Y. Ren, Y. Yuan, L. Zhou, S. Ju, and L. Chen, "GoRA: Gradient-driven Adaptive Low Rank Adaptation," *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[4] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training Deep Nets with Sublinear Memory Cost," *arXiv preprint arXiv:1604.06174*, 2016.

[5] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "PyTorch Distributed: Experiences on Accelerating Data Parallel Training," *Proc. VLDB Endowment*, vol. 13, no. 12, 2020.

[6] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel," *Proc. VLDB Endowment*, vol. 16, no. 12, 2023.

[7] S. Rajbhandari, J. Rasley, O. Rber, and Y. He, "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," *Proc. SC*, 2020.

[8] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training Verifiers to Solve Math Word Problems," *arXiv preprint arXiv:2110.14168*, 2021.

[9] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, et al., "Qwen Technical Report," *arXiv preprint arXiv:2309.16609*, 2023.

[10] NVIDIA Corporation, "NVIDIA Nsight Systems," https://developer.nvidia.com/nsight-systems, 2024.

[11] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods," https://github.com/huggingface/peft, 2022.