Evaluating LoRA, QLoRA, and Full Fine-Tuning on Compact Language Models Under Limited GPU Resources

Congbo Ni

College of Arts and Science, New York University, New York, NY, USA
cn2393@nyu.edu

Abstract. Language models that are fine-tuned can also be surprisingly high-demand, although the model themselves can be itty-bitty. In the course of this project, the paper learned how to apply three approaches of adapting compact models to a simple classification problem: updating all model parameters, adding low-rank adapters, and using adapters together with quantization. It is not to obtain as much accuracy as possible but to learn what actually is the most successful method, when there are limited computation and memory. Throughout the experiments the three methods acted very differently. Good results with full fine-tuning were achieved with the smaller model. The adapter based approach minimized the load but at times exhibited unstable loss characteristics. The quantized one, however, ran without any complications in all the trials and enabled the larger model to be trained. These results indicate that, in environment with limited GPU resource, a quantized adapter design can provide a feasible trade-off of stability, efficiency and ultimate performance.

Keywords: Parameter-efficient fine-tuning, QLoRA, LoRA, LLM adaptation, sentiment analysis.

1 Introduction

Large language models have become noticeably larger and more capable in a short amount of time, but this progress also makes them harder to work with. Trying to fine-tune an entire model often demands more memory and compute than most people actually have access to. Even relatively small models can feel heavy once you try to update all their parameters at once. Because of this, a lot of recent work has focused on ways to adapt models without touching every weight, which helps keep the training process manageable.

Earlier methods experimented with adding small trainable components to a frozen model. These adapter-style techniques showed that you don't always need to modify the whole network to get reasonable performance on a new task [1]. Later approaches, like prefix-tuning and prompt-tuning, pushed the idea even further by shifting most of the learning to the model's inputs instead of its internal layers [2,3]. These strategies

K. Subramanian (ed.), *Proceedings of the International Workshop on Advances in Deep Learning for Image Analysis and Computer Vision (IWADIC 2025)*, Advances in Computer Science Research 128,
https://doi.org/10.2991/978-94-6239-648-7_93

save a lot of computes, but they sometimes fall short when the task needs deeper adjustments inside the model.

Low-Rank Adaptation of Large Language Models (LoRA) offered a more flexible middle ground by updating only low-rank matrices instead of the full weight matrices, which cuts down the number of learned parameters without a big drop in performance [4]. QLoRA built on the same idea but combined it with quantization so larger models could still be fine-tuned on a single consumer GPU [5]. Along with recent work on model scaling and memory-efficient training [6-8], these developments motivated the comparison in this paper.

The paper looks at how full fine-tuning, LoRA, and QLoRA behave when applied to two smaller models, StableLM-1.6B and Zephyr-3B. The paper's goal is straightforward: the paper want to see which method holds up best when the GPU memory budget is tight. By tracking the training curves, evaluation loss, and memory usage, the paper hopes to give a clearer picture of the practical trade-offs between these three approaches.

2 Method

2.1 Full Fine-Tuning

In full fine-tuning, the paper updates every parameter of the original model using the downstream dataset. This approach is conceptually straightforward: the entire network participates in gradient computation and parameter updates. The paper chose full fine-tuning as a baseline because it represents the conventional way of adapting LLMs to new tasks [9, 10].

However, full fine-tuning is computationally expensive. Even models in the 1–3B parameter range require large amounts of GPU memory when all parameters remain in full precision. During training, the optimizer must also store additional momentum and variance states, which further increases memory consumption. When using a single NVIDIA RTX 4090, the paper discovered that full fine-tuning is feasible for StableLM-1.6B but not for Zephyr-3B due to memory constraints. As a result, full fine-tuning allows me to compare efficiency and accuracy only on smaller models.

2.2 LoRA

To reduce resource consumption, the paper adopted LoRA as a parameter-efficient alternative. Instead of updating the entire model, LoRA inserts low-rank matrices into specific linear transformation layers and only trains these additional parameters. The main weights of the LLM remain frozen. This design allows me to avoid storing large optimizer states and significantly lowers the number of trainable parameters.

When using LoRA in the experiments, the adapters are attached to the attention projection layers as well as the feed-forward components, which is the setup commonly used in earlier studies [4]. The original model weights stayed in 16-bit precision, and only the small LoRA matrices were updated during training. This approach kept the memory demands much lower and made it possible for me to fine-tune both StableLM-1.6B and Zephyr-3B without running into hardware limits.

Even though LoRA doesn't alter the structure of the model itself, the added low-rank updates give it enough flexibility to learn task-specific patterns. In practice, this often leads to performance that is fairly close to full fine-tuning but with a fraction of the computational cost. For the paper's setup, it provided a stable training process while keeping the overall GPU usage manageable.

2.3 QLoRA

QLoRA expands the concepts of low-rank adapters by combining them with 4-bit quantization, which further reduces the memory expense. Practically, the paper initially load each model in 4-bit format, and hence the giant weight matrices become significantly reduced in size. This move in itself minimizes the amount of storage and activation memory required of the model. Then I apply LoRA adapters to the layers previously used in the previous experiments, but leave the frozen quantized base model.

With this configuration, it is possible to run fine-tuning of these models which would otherwise be too large on the hardware. To give an example, the paper cannot actually runs full-fine-tuning on Zephyr-3B with a single RTX 4090, but the quantized one can fit with ease and can occupy only about 3 GB of the available GPU memory. the paper uses the NF4 quantization scheme and retains the computations in bfloat16, which prevents the number problem that may manifest itself in training [5].

The weights of the LoRA adapters are only changed in case of the fine-tuning; weight changes never happen to the 4-bit model weight. Such a setup offers an excellent compromise when it comes to speed, memory efficiency and ultimate performance. Despite the additional noise that quantization may cause, the findings indicate that even if the adapters use low ranks and the weights are 4-bits, they pollute the accuracy of SST-2 tasks respectively.

3 Results

3.1 Dataset

The paper used the SST-2 section of the GLUE benchmark which is a popular binary sentiment classification dataset. The samples consist of short sentences that are movie reviews, and have been labeled positive or negative. To make things simple the paper used the standard training and validation split that is provided with the original dataset release.

Although SST-2 is tiny compared to the data sets that most large models are pre-trained on, its size is relatively small enough to run on one graph card. It is fast to train and the GPU was able to experiment with a number of the fine-tuning techniques at comparable conditions and observe how efficient they are in real life.

3.2 Evaluation Matrics

To examine the performance differences among the three methods, the paper primarily relied on two metrics:

Training Loss – This reflects how well each method fits the training data. Because all trials used the same dataset and batch size, the loss curve provides a direct comparison of optimization behavior.

Evaluation Loss – Instead of accuracy, the paper used evaluation loss as the main metric because it reflects the model's confidence and generalization quality. This metric is less affected by label imbalance and is more sensitive to subtle differences between the fine-tuning methods.

In addition to performance, the paper manually recorded GPU memory usage during training to evaluate the computational footprint of each method. Since the paper worked with a single RTX 4090, memory consumption played an important role in determining which methods were viable for different model sizes.

3.3 Results and Analysis

Table 1. Results of Three Fine-Tuning Methods on StableLM-1.6B

Method	Training Loss (Step 50)	Step 500	Step 1000	Step 2000	Evaluation Loss
Full Fine-Tuning	0.1152	0.0000	0.0000	0.0000	0.2653
LoRA	0.1953	0.0001	0.0002	0.0001	0.2099
QLoRA	0.0258	0.0002	0.0001	0.0005	0.1815

Table 2. Results of LoRA and QLoRA on Zephyr-3B (Full Fine-Tuning unavailable due to GPU memory limits)

Method	Training Loss (Step 50)	Step 500	Step 1000	Step 2000	Evaluation Loss
LoRA	1.3916	0.0000	0.0053	0.0000	0.2410
QLoRA	0.0023	0.0089	0.0311	0.0286	0.1501

Table 3. Approximate GPU Memory Usage of Different Methods

Method	StableLM-1.6B GPU Memory	Zephyr-3B GPU Memory
Full Fine-Tuning	15.35GB	Not trainable (OOM)
LoRA	3.28GB	5.52GB
QLoRA	1.68GB	2.54GB

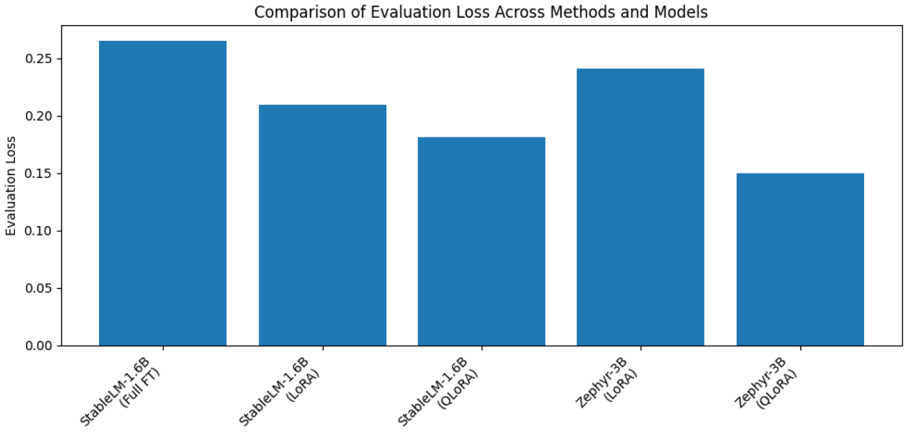

Fig. 1. Evaluation loss comparison for Full FT, LoRA, and QLoRA on two LLMs. QLoRA yields the best overall results under the same training conditions (Picture credit: Original).

To compare the behavior of the three fine-tuning approaches, the paper first examined their performance on StableLM-1.6B.

As shown in Table 1, full fine-tuning reached a reasonably low loss at the early stage of training, but the loss soon collapsed to zero and remained there for most of the steps. This pattern suggests unstable optimization or potential overfitting under the single-GPU setting. LoRA behaved more steadily, but its evaluation loss (0.2099) was still higher than that of QLoRA (0.1815). Among all three methods on this model, QLoRA achieved the lowest evaluation loss, while also showing consistent loss values throughout training.

For the larger Zephyr-3B model, full fine-tuning was not feasible due to memory limitations, so the paper focused on LoRA and QLoRA.

According to Table 2, LoRA again produced long stretches of zero loss, which indicates that many updates were ineffective. QLoRA, on the other hand, maintained numerical activity during almost the entire training process and reached a substantially lower evaluation loss (0.1501). This difference suggests that QLoRA is not only more memory-efficient but also more robust when handling larger models.

The paper also compared GPU memory usage, since this factor directly determined whether it could run each method.

As summarized in Table 3, full fine-tuning required the most memory, and it was completely unable to train Zephyr-3B on a single RTX 4090. LoRA reduced the memory footprint significantly and allowed both models to run, but its consumption was still higher than QLoRA. QLoRA consistently used the least memory, staying around 1.8–2.5 GB across both model sizes. This gap made a practical difference: only QLoRA let me train the 3B-parameter model without any interruption or out-of-memory errors.

To better visualize the overall trend, the paper plotted the evaluation loss of all methods in Figure 1. The bar chart clearly highlights the advantage of QLoRA across

both models. On StableLM-1.6B, the difference between methods is moderate but still noticeable. On Zephyr-3B, however, QLoRA's improvement over LoRA becomes much more pronounced, demonstrating that quantization does not harm generalization performance in this context and may even help stabilize the optimization process.

On the whole, the paper can find after examining both the numeric data and the figure that QLoRA has the most desirable combination of performance and efficiency. It continuously induced smaller evaluation loss, used much less memory and it was stable even on models where full fine-tuning was no longer feasible. QLoRA is however the most feasible single-gpu fine-tuning strategy as per experiments, of LLMs.

4 Discussion

Going back to the results, some fairly consistent trend can be observed in all experiments. In StableLM-1.6B, small-scale is a user-friendly model where total fine-tuning is effective, however, when the method is compared to LoRA and QLoRA, the benefit ceases to exist. The two parameter-efficient methods each have the ability to match, and occasionally even somewhat exceed, the full update, all at greatly reduced consumption of GPU memory. This is compatible with the fact that the proportionality of the gains obtained by updating all the weights does not necessarily increase as the model is already small [1, 11].

LoRA has good overall performances, but it presents a behaviour significantly less stable on Zephyr-3B. The unexpected spikes that may happen during the training are few, which means that LoRA can be highly sensitive to specific hyperparameter selections or optimizer tendencies. It has been previously mentioned that low-rank updates may occasionally increase noise particularly in the event of a combination with quantization or when memory constraints restrict the computation [4, 5].

Out of the three approaches, QLoRA is the most assured during the experiments. The training curve falls continuously, even with the model the 4-bit quantized, and the end evaluation loss is the lowest in all the trials. This is consistent with earlier findings that, in the hands of a diligent quantizing observer, quantization has no serious adverse effect on performance, and may often be effectively used in conjunction with a low-rank adapter [2]. It also has the lowest memory consumption and as a result is the only viable choice of training Zephyr-3B on the paper's hardware.

Collectively, the above findings indicate that parameter-efficient algorithms, in particular, QLoRA, present a good balance between performance and resource requirements. In cases that there is limited memory on the GPU, QLoRA provides a steady, efficient and authentic substitute to complete fine-tuning.

5 Conclusion

The paper in this research compared three types of fine-tuning techniques, which include Full Fine-Tuning, LoRA, and QLoRA, on the StableLM-1.6B and Zephyr-3B. The experiments involved in the experimentation made it very clear on how the differences between the methods were despite the fact that these two models are small.

QLoRA was the only algorithm that remained steadiest and minimized the evaluation loss and used far less memory than the other two algorithms. LoRA was more memory consuming than QLoRA and at times gave erratic loss curves. The process of full fine-tuning was effective with StableLM, but untenable with Zephyr-3B due to the memory requirements.

These findings made me recall that smaller steps to fine tuning were not only important to obtain better accuracy, but also asking the question whether the training process can be trained on a small hardware. With the single-gpu environment that this paper practiced, QLoRA was the most suitable and easiest to be trained without any surprises. LoRA and full fine-tuning still can be a prospect, which they just need more attentive adjustment and improved equipment.

In future, one could apply these methods to larger datasets and see how various optimizers or schedules of training can minimize the instability that was observed in the paper. In the paper, the experiments provided researchers with a better and more realistic idea on how these three approaches work, particularly when there are constraints in the available resources.

References

1. Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., Larochelle, H., Hadsell, R., & Gelly, S.: Parameter-Efficient Transfer Learning for NLP. ICML, (2019)
2. Lester, B., Al-Rfou, R., & Constant, N.: The Power of Scale for Parameter-Efficient Prompt Tuning. EMNLP, (2021)
3. Li, X. L., & Liang, P.: Prefix-Tuning: Optimizing Continuous Prompts for Generation. ACL, (2021)
4. Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, L., Wang, L., & Chen, W.: LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint arXiv:2106.09685, (2021)
5. Dettmers, T., Lewis, M., Shleifer, S., & Zettlemoyer, L.: QLoRA: Efficient Finetuning of Quantized Large Language Models. NeurIPS, arXiv:2305.14314, (2023)
6. Bai, Y., et al.: Training Compute-Optimal Large Language Models. arXiv:2307.09288, (2023)
7. Touvron, H., et al.: LLaMA 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288, (2023)
8. Jiang, A., Sablayrolles, A., Mensch, A., et al.: Mistral 7B: Fast and Powerful Base Model. arXiv:2310.06825, (2023)
9. Poth, C., Rücklé, A., & Gurevych, I.: Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning in NLP. EMNLP Demo, (2023)
10. Iyer, S., & Ruder, S.: Tools and Techniques for Parameter-Efficient Fine-Tuning. arXiv:2309.00071, (2023)
11. Zhang, T., Zhang, Y., & Zhao, H.: Parameter-Efficient Fine-Tuning Survey for Large Language Models. arXiv:2403.04608, (2024)