Resource-Efficient Large Language Model on Log Anomaly Detection

Christy Zhang, Mohammad Hossein Shokouhi, Vincent W.S. Wong

Abstract

Large language models (LLMs) achieve strong results for log anomaly detection (LAD) but are costly to fine-tune to a downstream task and generate inference. This work presents a two-stage pipeline that separates training cost from deployment cost. In the first stage, Quantized Low-Rank Adaptation (QLoRA) is used to adapt a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model while significantly reducing GPU memory usage during fine-tuning, with minimal impact on accuracy. On the HDFS dataset, peak training memory is reduced by 85%. In the second stage, iterative structured pruning is applied to the multi-layer perceptron (MLP) layers on top of the QLoRA-adapted model, targeting deployment efficiency by shrinking the model and reducing latency. At the 90% pruning level, the model achieves F1 score at 0.944 that is higher than full fine-tuning, while reducing model size by 46% and inference latency by 45% on GPU. We further validate these gains by running inference on a Raspberry Pi device, where pruning reduces latency by over 40%. These improvements come at the cost of increased training time. Overall, the proposed pipeline enables resource-efficient adaptation and deployment of large language models without sacrificing performance.

I. Introduction

LLMs such as BERT have achieved impressive results in LAD tasks, making them valuable tools for improving reliability and security in large-scale computing systems. However, their high computational and memory demands often make it challenging to deploy and fine-tune these models in resource-limited environments, such as edge servers or embedded devices.

To address these limitations, this project introduces a two stage pipeline for reducing both the memory requirements of fine-tuning and the computational overhead of inference for BERT-based LAD models. In the first stage, We first employ Low-Rank Adaptation (LoRA) [1] and QLoRA [2] methods to significantly reduce GPU memory usage during fine-tuning, allowing fine-tuning to be performed on less powerful hardware without sacrificing detection accuracy.

In the second stage, we apply an iterative structured pruning method [3] to the MLP layers on top of the QLoRA approach. By progressively removing less important neurons in several steps, we further reduce model size and speed up inference, supporting near real-time applications and deployment on small devices.

This pipeline improves both deployment efficiency and detection performance, though at the cost of longer fine-tuning time. Experiments on the HDFS dataset show that the first stage lowers peak memory during fine-tuning without sacrificing accuracy, and the second stage reduces final model size and shortens inference latency. These gains come with longer training due to the prune and tune cycles. The two stage design gives practitioners explicit control over training and deployment budgets, making LLM based LAD practical under strict resource constraints.

II. Related Work

Efficient LAD is crucial for maintaining the reliability and security of large-scale computer systems. Over recent years, researchers have concentrated on developing robust preprocessing methods for raw log data to enhance anomaly detection performance. A variety of specialized log parsers and data pipelines have emerged, significantly improving the way logs are represented and utilized in machine learning models. Examples include LogBERT [4], which applies a log parser to transform log messages into sequences of log keys, and embeds them with both log key embeddings and positional embeddings to

capture structural patterns. Another approach, LLMeLog [5], leverages LLMs to extract enriched semantic vectors from log messages, facilitating the capture of deeper contextual information essential for anomaly detection. These approaches have already demonstrated strong performance in detecting anomalies.

Furthermore, recent studies have begun investigating parameter-efficient fine-tuning (PEFT) strategies, such as LoRA [6] and Representation Fine-Tuning (ReFT) [7], aiming to reduce resource demands without sacrificing model performance. These studies have shown that PEFT strategies can substantially reduce the number of trainable parameters and epoch time.

In addition, prior work shows that pruning can substantially reduce model size with no loss in accuracy [3], yielding smaller storage footprints and lower inference latency. With an appropriate pruning level, it may also improve generalization and thus overall model performance [8].

Existing LAD methods demonstrate the effectiveness of transformer models, while PEFT and pruning offer complementary levers for efficiency. Prior work shows that LoRA and QLoRA reduce the memory needed for adaptation, and that structured pruning reduces the computational cost of inference. Our work offers a systematic, pipeline-based evaluation of LoRA, quantization, and iterative structured pruning and reports accuracy, training memory, model size, and latency to show where each technique provides practical value under tight resource constraints.

III. Methodology

A. Dataset and Preprocessing

This project uses the HDFS_v1 dataset [9], [10], which consists of 558,233 sequences labeled as normal and 16,838 sequences labeled as abnormal. The dataset provides a reliable and realistic benchmark for LAD in distributed systems.

Given the significant class imbalance, down-sampling is applied to the majority class to avoid avoid introducing bias toward it. The training set is composed of 5,582 normal traces and 2,791 abnormal traces, which is approximately 0.01 of the dataset while maintaining a 2:1 class ratio. Prior studies have shown that a 0.01 training ratio is sufficient for fine-tuning BERT on LAD tasks, and increasing the ratio yields minimal performance gains[6]. This strategy exposes the model to a representative distribution while mitigating overfitting to the dominant class.

To retain semantic information, each input sequence is represented by its log event templates. These human-readable event tokens preserve contextual structure, enabling transformer-based models such as BERT to exploit their natural language understanding capabilities. All sequences are tokenized before being fed into the BERT encoder. The tokenized representations include attention masks and positional embeddings, which allow the model to capture both temporal dependencies and structural patterns in the event sequences.

B. Model Selection

The BERT model[11] is selected as the base architecture for this project due to its encoder-only transformer structure, which offers computational efficiency and scalability. BERT has also been widely adopted in prior LAD research, consistently demonstrating its effectiveness in representing log sequences. Additionally, its modular design makes it well-suited for QLoRA and structured pruning.

C. Beseline Methods

The baseline models used for comparison are the zero-shot BERT model and the fully fine-tuned BERT model.

Zero-Shot Model refers to the original pre-trained BERT model applied directly to the LAD task without any fine-tuning. This serves as a performance lower bound and helps assess how well general-purpose language representations transfer to LAD.

Fully Fine-Tuned Model involves updating all parameters of the pre-trained BERT model on the LAD dataset. This approach allows the model to fully adapt to the target task and typically yields strong task-specific performance. However, it is computationally intensive and memory-demanding, making it less suitable for deployment on edge devices with constrained resources.

D. Efficient Fine-Tuning and Compression Methods

This project adopts a two stage pipeline and centers on three techniques: LoRA, quantization, and iterative structured pruning. Stage one applies LoRA and quantization to reduce GPU memory during fine-tuning while preserving accuracy. Stage two compresses the adapted model with iterative structured pruning of the MLP layers to cut parameters, shrink on disk size, and lower inference latency. These methods enable the model to retain the competitive performance of full fine-tuning while significantly lowering fine-tuning memory usage and inference overhead, making them more suitable for deployment in resource-constrained environments.

1) ***LoRA****:* LoRA is a PEFT technique that reduces the number of trainable parameters required to adapt large pre-trained models to downstream tasks. Instead of updating all model weights during fine-tuning, LoRA freezes the original model parameters and injects trainable rank-decomposition matrices into each layer of the transformer architecture. In this project, LoRA adapters are inserted into the query and value projections of each attention layer. This approach reduces the number of trainable parameters from over 100 million in full fine-tuning to 75,266, which saves significant GPU memory and reduces the training cost during fine-tuning.

2) ***Quantization****:* Quantization is a model compression technique that reduces the precision of weights and activations in neural networks, thereby lowering both memory footprint and computational requirements. The base BERT model is stored in 32-bit floating point (float32) by default, which consumes substantial GPU memory during fine-tuning. In this project, encoder layers from the base model are quantized to 4-bit precision using the NF4 quantization scheme. This conversion further lowers GPU memory usage during fine-tuning and enables large pre-trained models to be adapted on resource-constrained servers.

3) ***Pruning****:* Pruning is a model compression technique that reduces the size and complexity of neural networks by removing less important weights or structural components. In this project, structured pruning is applied to the MLP layers within each transformer encoder block of the BERT model. An iterative pruning approach is used: in each iteration, 10% of the original neurons in all MLP layers are pruned based on their magnitude or importance, and the model is then fine-tuned on the LAD task to adapt to the reduced architecture and improve the F1 score. This prune-and-fine-tune process is repeated for several iterations, gradually reducing the model size while aiming to maintain classification accuracy. By carefully controlling the pruning schedule and training after each step, this method reduces the total number of model parameters. The resulting model is smaller, has lower inference latency, and is therefore more suitable for deployment in near real-time applications on small devices.

E. Evaluation Metrics

To assess both the anomaly detection performance and the computational efficiency of different fine-tuning strategies, this project adopts a combination of metrics that capture classification quality such as accuracy-related measures and resource usage such as memory, time, and storage.

Classification Metrics assess each model's ability to detect anomalies in the test dataset, where abnormal samples form the minority class. The F1 score is used as the primary metric for final performance comparison, as it provides a balanced evaluation of both precision and recall, making it particularly suitable under class imbalance.

- F1 Score (Primary Metric): The harmonic mean of precision and recall, offering a single measure that balances false positives and false negatives.
- Precision (Reference Metric): The proportion of correctly identified anomalies among all instances predicted as anomalous.

- Recall (Reference Metric): The proportion of actual anomalies correctly identified by the model.

Efficiency Metrics To evaluate the resource demands of each approach, the following efficiency metrics are recorded:

- Peak GPU Memory Usage: The maximum GPU memory consumed during training, measured using PyTorch's built-in memory tracking utilities.
- Total Training Time: The total duration required to complete the fine-tuning process.
- Inference Latency: The average time required to perform prediction on a single data sample.
- Model Storage Size: Total on-disk size of deployable files.

These metrics together enable a comprehensive comparison between baseline and PEFT and model compression strategies in terms of both performance and deployability on resource-constrained platforms.

IV. Experiment Setup

A. Experimental Environment

All fine-tuning and evaluation experiments were conducted on a local workstation running Windows 11 Home (version 24H2), equipped with an Intel Core i7-14700F processor (20 cores, 2.10 GHz), 32 GB of system memory, and an NVIDIA GeForce RTX 4070 SUPER GPU with 12 GB of dedicated VRAM. The experimental environment was based on Python 3.10.13.

To assess deployment performance in constrained environments, inference was also evaluated on a Raspberry Pi 5, equipped with a quad-core ARM Cortex-A76 CPU and 8 GB of RAM.

B. Model Configuration Details

The base model used in this project is bert-base-uncased, obtained from the Hugging Face platform. It is loaded using the AutoModelForSequenceClassification class with the number of output labels set to 2 for binary classification. Internally, this loads a BERT model with 12 transformer encoder layers, each with a hidden size of 768, 12 self-attention heads, and a feedforward intermediate size of 3072 in the MLP layers. The model supports input sequences of up to 512 tokens in length, and contains approximately 110 million parameters in total. The corresponding AutoTokenizer is used to preprocess the text data by tokenizing each log sequence, converting text to lowercase, and applying truncation and padding to a fixed length of 512 tokens.

C. Fine-tuning Procedure and Hyperparameters

All models are fine-tuned with batch size of 4. The primary evaluation metric for selecting the best model is the F1 score. Weight decay is set to 0.01, and early stopping is employed with a patience of 1 epoch. The learning rate is set to 5×10^{-5} by default and reduced to 2×10^{-5} when quantization is applied.

For LoRA-based fine-tuning, the configuration includes a rank of 2, a scaling factor (α) of 4, and a dropout rate of 0.005. LoRA is applied to the query and value projection matrices within the attention modules. As described in the methodology section, the model is quantized to 4-bit using the NF4 data type in QLoRA experiments.

In the iterative pruning setup, 10% of the original neurons in the MLP (intermediate) layers are removed in each pruning iteration, followed by fine-tuning for 2 epochs. In the final iteration, the pruned model is fine-tuned until convergence to fully adapt to the LAD task. Experiments are conducted across cumulative pruning levels ranging from 50% to 90%, to assess the effect of progressive pruning on model performance and efficiency.

D. Evaluation Settings

All evaluations on the local workstation were performed with a batch size of 8 over the full test dataset to measure classification performance (e.g., F1 score) and inference latency.

To assess real-world deployability, inference was also evaluated on a Raspberry Pi 5 device. Due to limited hardware resources, evaluation on the Raspberry Pi was conducted with a batch size of 1 using 100 randomly sampled test sequences.

E. Reproducibility

To support reproducibility, a fixed random seed of 42 was applied across all experiments to ensure consistent model initialization, data shuffling, and training behavior. All code, configuration files, and data preprocessing steps are available in the project's GitHub repository to facilitate replication and further experimentation.

V. Results

A. Baseline Models

Without task-specific adaptation, the zero-shot BERT model collapses to predicting all samples as abnormal, yielding an F1 of 0.048 and making this baseline unusable for LAD. After full fine-tuning for 2 epochs over 424.63 seconds, performance rises to an F1 of 0.919. The per-sample inference latency is 0.00760 seconds, final model size is 418 MB, and peak GPU memory during fine-tuning is 2,528.87 MB. We use this configuration as the accuracy and latency reference for the remainder of the section. For the complete performance data, see Table I.

TABLE I: Baseline Performance and Efficiency

Model	F1-Score	Precision	Recall	Peak GPU Memory Usage (MB)	Training Time (s)	Training Epochs	Model Size (MB)	Inference Latency (s)
Zero-shot Baseline	0.048	0.025	1.000	—	—	—	417	0.00765
Full Fine-Tuning	0.919	0.884	0.957	2528.87	424.63	2	418	0.00760

B. LoRA and QLoRA Performance and Efficiency

Table II shows that LoRA attains an F1 of 0.886 with peak training memory of 1,447.76 MB, about 43% lower than full fine-tuning at 2,528.87 MB. QLoRA without pruning reaches an F1 of 0.916, nearly matching full fine-tuning, while lowering peak training memory by about 85% to 378.34 MB. Inference latency and final model size remain essentially unchanged relative to full fine-tuning. However, with LoRA and quantization, fine-tuning requires more epochs and longer time to converge.

TABLE II: Beseline Models Performance

Model	F1-Score	Precision	Recall	Peak GPU Memory Usage (MB)	Training Time (s)	Training Epochs	Model Size (MB)	Inference Latency (s)
LoRA Fine-Tuning	0.886	0.836	0.944	1447.76	1224.16	8	418	0.00790
QLoRA Fine-Tuning (No Pruning)	0.916	0.884	0.952	378.34	853.13	4	418	0.00793

C. Iterative Pruning

When iterative pruning is applied on top of QLoRA, increasing the cumulative pruning level reduces model size and speeds up inference. The F1 score dips slightly at low pruning levels, then recovers and ultimately surpasses both the full fine-tuning and unpruned QLoRA baselines. Peak training memory remains close to the QLoRA baseline, while training time and the total number of epochs increase due to the prune-then-fine-tune schedule. For complete results across pruning levels, see Table III.

At the 90% cumulative MLP pruning level, the model achieves an F1 of 0.944, exceeding full fine-tuning. Inference latency drops by 45% to 0.00418 seconds, and the final model size decreases by 46% to 224 MB compared with full fine-tuning. The trade-off is longer fine-tuning time: 3601.71 seconds over 20 epochs versus 424.63 seconds for full fine-tuning.

TABLE III: QLoRA with Iterative Pruning Performance and Efficiency by Cumulative Pruning Level

Cumulative MLP Pruned (%)	F1-Score	Precision	Recall	Peak GPU Memory Usage (MB)	Training Time (s)	Training Epochs	Model Size (MB)	Inference Latency (s)
50	0.899	0.882	0.917	370.57	2305.94	12	310	0.00611
60	0.903	0.873	0.935	370.57	2678.65	14	289	0.00571
70	0.922	0.918	0.926	370.57	3021.59	16	267	0.00536
80	0.928	0.915	0.941	370.57	3638.97	20	246	0.00468
90	0.944	0.946	0.941	370.57	3601.71	20	224	0.00418

D. Performance Trade-offs and Overall Improvement

Figure 1 compares accuracy and efficiency across model configurations on HDFS. The zero-shot baseline is omitted because it failed to produce meaningful predictions.

Figure 1(a) reports F1 on the evaluation set. Task-specific fine-tuning raises F1 to 0.90. LoRA causes a small decrease, and quantization largely recovers it. Iterative pruning then improves generalization, with the highest pruning level reaching an F1 of 0.944. Figure 1(b) reports peak GPU memory during fine-tuning. Both LoRA and Quantization dramatically reduce memory, which together reduce 85% of memory usage to 378.34 MB, while additional pruning has little effect on peak memory. Figures 1(c) and 1(d) summarize inference metrics. Pruning delivers the largest gains, reducing model size by 46% to 224 MB and cutting per-sample latency by 45% to 0.00418 seconds at the 90% pruning level. LoRA or QLoRA does not speed up inference as its adapter layers introduce slight overhead that increases latency.

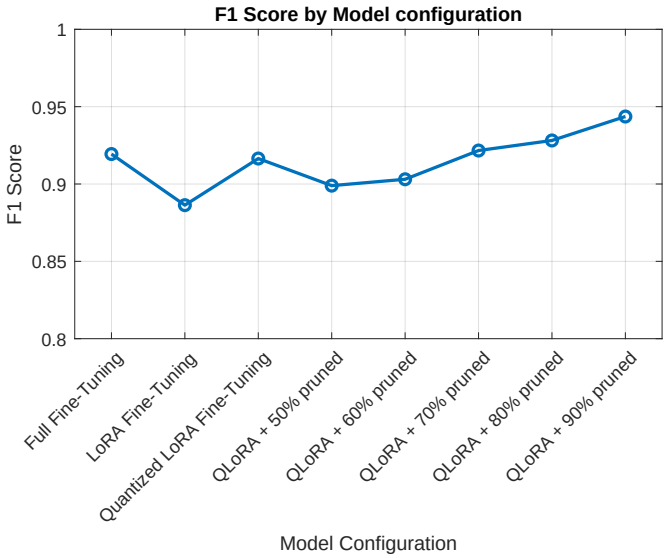
These gains come with a cost: fine-tuning time rises from 424.63 seconds in full fine-tuning to 1224.16 seconds with LoRA (+188.3%), 853.13 seconds with QLoRA (+100.9%), and 3601.71 seconds at QLoRA with 90% pruning (+748.2%).

E. Raspberry PI Deployment Results

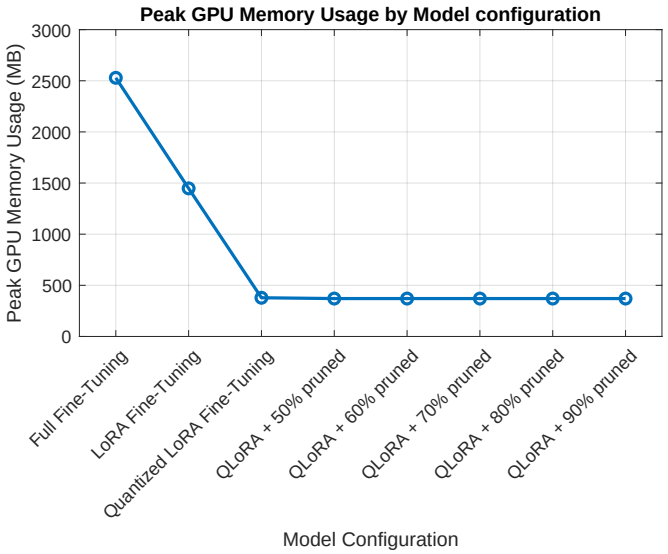
In addition to GPU-based measurements, we also evaluated inference latency on a Raspberry Pi 5 to assess real-world performance on resource-limited edge devices. The Raspberry Pi is considered a worst-case deployment scenario because it lacks a dedicated GPU and has very limited computational resources, making it a strong test of efficiency gains.

As shown in Table IV and Figure 2, iterative pruning significantly improves latency even in low-power settings. QLoRA with 90% MLP pruning reduces per-sample latency from 1.344 seconds in full fine-tuned model to 0.795 seconds, which gives a 40.9% reduction.

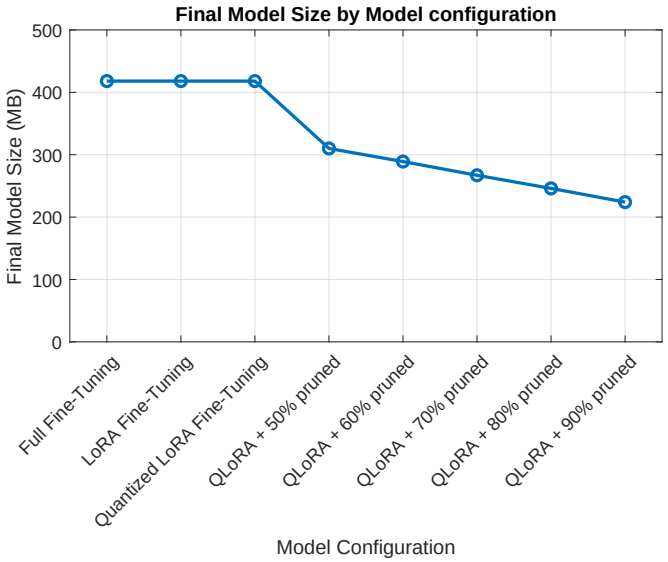
These results observed on the Raspberry Pi are consistent with those obtained on the local workstation. In both environments, pruning reduced inference latency, confirming that the efficiency improvements generalize across different hardware platforms.



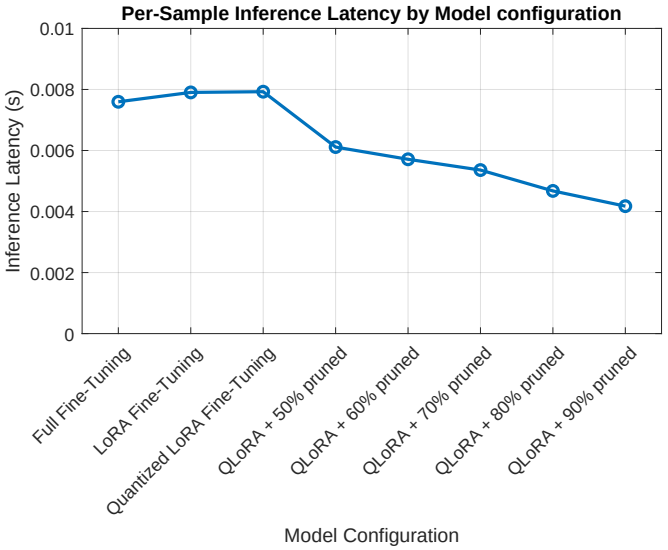
(a) F1 score across model configurations



(b) Peak GPU memory during fine-tuning



(c) Final model size after fine-tuning



(d) Per-sample inference latency

Fig. 1: Performance and efficiency across model configurations on HDFS (lower is better for memory, model size, and latency)

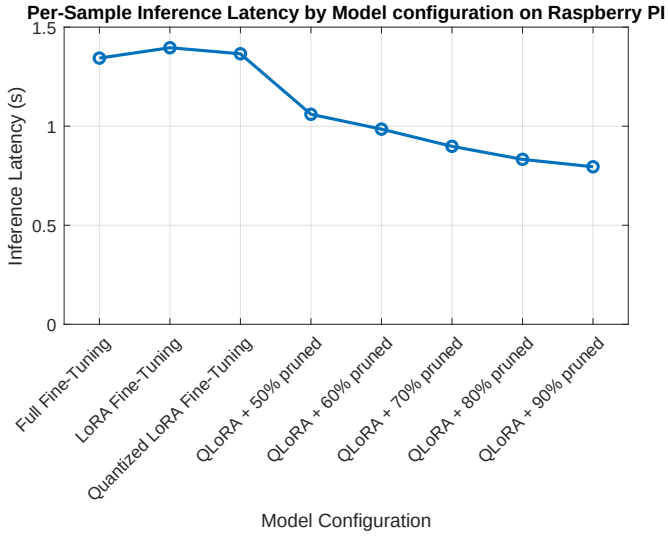


Fig. 2: Inference latency on Raspberry Pi

TABLE IV: Inference Latency on Raspberry Pi

Model	Inference Latency (s)
Full Fine-Tuning	1.344
LoRA Fine-Tuning	1.396
Quantized LoRA Fine-Tuning	1.366
QLoRA + 50% pruned	1.060
QLoRA + 60% pruned	0.985
QLoRA + 70% pruned	0.899
QLoRA + 80% pruned	0.833
QLoRA + 90% pruned	0.795

VI. Conclusion

This work evaluated full fine-tuning, LoRA, QLoRA, and QLoRA with iterative MLP pruning for log anomaly detection on the HDFS dataset. QLoRA maintains detection accuracy while significantly reducing GPU memory usage during adaptation. Building on this, iterative structured pruning further enhances the deployment profile by reducing model size and inference latency, while also improving F1 performance beyond both full fine-tuning and unpruned QLoRA. Evaluation on a Raspberry Pi confirms that the latency improvements from pruning translate to real-world gains on resource-constrained edge devices. These benefits come at the cost of increased fine-tuning time, as QLoRA extends training duration and iterative pruning adds further overhead.

In practice, select QLoRA when GPU memory is the binding constraint and longer training is acceptable. Apply iterative MLP pruning on top of QLoRA when deployment efficiency such as lower latency and smaller models is the priority. Choose full fine-tuning when rapid convergence is required and computational resources are abundant.

References

[1] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2022.

[2] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Proc. Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, New Orleans, LA, USA, Dec. 2023.

[3] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Montreal Canada, Dec. 2015.

[4] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Shenzhen, China, Jul. 2021.

[5] M. He, T. Jia, C. Duan, H. Cai, Y. Li, and G. Huang, "LLMeLog: An approach for anomaly detection based on LLM-enriched log events," in *Proc. IEEE Int. Symp. Softw. Reliab. Eng. (ISSRE)*, Tsukuba, Japan, Oct. 2024.

[6] S. He, Y. Lei, Y. Zhang, K. Xie, and P. K. Sharma, "Parameter-efficient log anomaly detection based on pre-training model and LoRA," in *Proc. IEEE Int. Symp. Softw. Reliab. Eng. (ISSRE)*, Florence, Italy, Oct. 2023.

[7] Y. F. Lim, J. Zhu, and G. Pang, "Adapting large language models for parameter-efficient log anomaly detection," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining (PAKDD)*, Sydney, NSW, Australia, Jun. 2025.

[8] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, New Orleans, Louisiana, USA, May 2019.

[9] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Haifa, Israel, Jun. 2010.

[10] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for AI-driven log analytics," in *Proc. Int. Symp. Softw. Reliab. Eng. (ISSRE)*, Florence, Italy, Oct. 2023.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol. (NAACL-HLT)*, Minneapolis, Minnesota, Jun. 2019.