Efficient Fine-Tuning of Transformer Models Under Resource Constraints

Keemin Lee
Department of Computer Science
Columbia University
New York, NY, USA

Sreeram Raghammudi
Department of Computer Science
Columbia University
New York, NY, USA

Aaryaman Bajaj
Department of Computer Science
Columbia University
New York, NY, USA

Aryaman Velampalli
Department of Computer Science
Columbia University
New York, NY, USA

Aravindan Jambunathan
Department of Computer Science
Columbia University
New York, NY, USA

***Abstract*—Fine-tuning large transformer models remains computationally expensive, limiting their applicability in resource-constrained environments. This project investigates parameter-efficient fine-tuning techniques, specifically LoRA and QLoRA and compares them against full fine-tuning baselines. We evaluate tradeoffs in accuracy, memory usage, and training throughput, and use profiling tools to analyze the underlying performance behavior. Our results demonstrate that parameter-efficient methods can significantly reduce resource requirements while maintaining competitive performance.**

Code and Resources

The complete implementation and experiments are available at https://github.com/keeminlee/hpml-peft-benchmark. A detailed blog post describing our approach can be found at https://medium.com/@sr4314/efficient-fine-tuning-of-transformer-models-under-resource-constraints-b27cca057ca6.

I. Introduction

A. Background and Motivation

Transformer-based language models have become the dominant paradigm in natural language processing, achieving state-of-the-art results across tasks such as sentiment analysis, question answering, and natural language inference [1], [2]. Despite their success, fine-tuning these models remains computationally expensive, often requiring significant GPU memory and training time.

This cost limits their applicability in resource-constrained environments, including edge devices and low-power systems. As a result, recent research has focused on parameter-efficient fine-tuning (PEFT) methods that reduce training overhead while preserving model performance [3], [4].

B. Problem Statement

While techniques such as LoRA and quantization-based approaches promise substantial efficiency gains, their practical tradeoffs, particularly in terms of memory usage, throughput, and system-level behavior are not fully understood. In this project, we seek to empirically evaluate these tradeoffs using controlled benchmarks and profiling tools.

C. Objectives and Scope

The goal of this work is to compare full fine-tuning, LoRA, and QLoRA on a standard NLP benchmark, focusing on training-time efficiency and system performance rather than solely on final accuracy.

II. Literature Review

The transformer architecture introduced by Vaswani et al. [1] revolutionized sequence modeling by replacing recurrence with self-attention, enabling improved parallelization and scalability. This paradigm underpins modern pretrained language models such as BERT [2], which demonstrated the effectiveness of bidirectional contextual representations, and DistilBERT [5], which applies knowledge distillation to reduce model size while retaining performance. Despite their strong empirical results, these models typically contain hundreds of millions to billions of parameters, making full fine-tuning computationally expensive and memory intensive, particularly in resource-constrained environments.

To address these challenges, a growing body of work has explored parameter-efficient fine-tuning (PEFT) techniques that adapt large pretrained models while updating only a small subset of parameters. Adapter-based methods insert lightweight, task-specific modules between transformer layers, achieving competitive performance with significantly reduced training cost [3]. More recently, Low-Rank Adaptation (LoRA) proposes decomposing weight updates into low-rank matrices, allowing the base model parameters to remain frozen while learning a compact set of trainable weights [4]. This approach reduces both memory footprint and optimization complexity, making it particularly attractive for large-scale models.

Building on LoRA, QLoRA combines low-rank adaptation with aggressive 4-bit quantization of the base model weights [6]. By leveraging quantization-aware training and efficient memory management techniques, QLoRA enables fine-tuning

of models with tens of billions of parameters on a single consumer-grade GPU, without sacrificing downstream task performance. This work highlights the growing importance of jointly considering algorithmic efficiency and hardware constraints when adapting large language models.

From a systems perspective, prior research underscores the necessity of detailed performance analysis to fully understand the trade-offs introduced by such efficiency-driven methods. Framework-level studies emphasize profiling, kernel-level inspection, and memory analysis as essential tools for identifying bottlenecks in deep learning workloads [7]. These system-level insights motivate our use of profiling tools alongside accuracy-based metrics, enabling a more holistic evaluation of fine-tuning approaches that accounts for both model performance and computational efficiency.

A. Identification of Gaps in Existing Research

While prior work has established the effectiveness of parameter-efficient fine-tuning (PEFT) methods such as adapters, LoRA, and QLoRA in reducing training cost while preserving downstream accuracy, several important gaps remain in the literature.

First, most studies primarily emphasize task-level performance metrics (e.g., accuracy or loss) and parameter count reduction, with comparatively limited attention to end-to-end system behavior. Metrics such as training throughput, peak memory allocation, and latency are often reported in isolation or omitted entirely, making it difficult to assess the practical feasibility of deploying PEFT methods under strict hardware constraints.

Second, existing evaluations frequently focus on a single efficiency dimension such as memory reduction through quantization or parameter sparsity without jointly analyzing how these choices interact at runtime. In particular, the trade-offs between reduced gradient computation, increased operator complexity (e.g., dequantization overhead), and memory bandwidth utilization are rarely quantified using operator-level profiling tools.

Third, many PEFT studies evaluate performance on large-scale or well-provisioned hardware, implicitly assuming access to high-memory GPUs. As a result, there is limited empirical evidence characterizing how these methods behave on constrained hardware settings, where memory pressure, kernel launch overheads, and activation storage dominate system performance.

Finally, although system-level profiling frameworks are well established in the deep learning systems community, they are seldom integrated into PEFT benchmarking pipelines. This disconnect limits interpretability and obscures the underlying causes of observed performance differences between full fine-tuning and parameter-efficient alternatives.

These gaps motivate the need for a holistic evaluation that jointly considers accuracy, memory usage, training throughput, and operator-level execution characteristics when comparing fine-tuning strategies under resource constraints.

III. Methodology

A. Data Collection and Preprocessing

We evaluate our methods on the SST-2 sentiment classification task from the GLUE benchmark [8]. The dataset consists of 67,349 training examples and 872 validation examples. Standard tokenization and truncation are applied using the DistilBERT tokenizer. Implementation relies on the Hugging Face Transformers library [9] for model instantiation and tokenization. No additional data augmentation is performed.

B. Model Selection

We use DistilBERT as the base transformer model due to its balance of performance and efficiency(66.9M parameters). Three fine-tuning strategies are evaluated:

- Full fine-tuning (baseline): All 66.9M parameters are trainable.
- LoRA-based fine-tuning: Low-rank adapters added to query and value projection layers.
- QLoRA-based fine-tuning: LoRA combined with 4-bit NF4 quantization.

C. Optimization Procedures

All experiments use the following hyperparameters:

- Learning rate: 3×10^{-5}
- Optimizer: AdamW with default weight decay
- Training epochs: 3
- Warmup ratio: 0.1
- Seed: 42 (for reproducibility)

For LoRA and QLoRA experiments, we evaluate four rank configurations ($r \in \{4, 8, 16, 32\}$) with $\alpha = 2r$ and dropout of 0.1. Target modules are the query (q_lin) and value (v_lin) projection layers. The baseline uses a batch size of 8 due to memory constraints, while LoRA and QLoRA use batch size 32.

QLoRA employs 4-bit NF4 quantization with bfloat16 compute dtype for the base model weights, with adapters trained in full precision.

D. Profiling Tools and Methods

We use PyTorch Profiler to collect operator-level execution traces, memory usage, and kernel timing information. Profiling runs are intentionally short and isolated from full benchmark runs to avoid skewing performance measurements.

E. Evaluation Metrics

Evaluation metrics include:

- Classification accuracy
- Training throughput (tokens/sec)
- Peak GPU memory usage

IV. Experimental Results

A. Experimental Setup

Experiments are conducted on a single NVIDIA A100 GPU. All methods use the same dataset splits, batch sizes, and optimization settings where applicable to ensure fair comparison.

B. Performance Comparison of Different Models

1) Baseline Full Fine-Tuning: The baseline DistilBERT model achieves 91.86% validation accuracy after 3 epochs of full fine-tuning. Training requires 518.2 seconds (171.3 seconds per epoch) with a throughput of 389.9 samples/second. Peak GPU memory consumption is 1305.9 MB (PyTorch allocator) and 2398.3 MB (NVML). All 66.9M parameters are trainable. The median step latency is 12.3ms with 95th percentile at 14.6ms.

2) LoRA Results: Table I presents LoRA results across four rank configurations. LoRA achieves competitive accuracy (87.8-88.0%) while reducing trainable parameters by 98.0-98.2% compared to the baseline. The optimal configuration is rank 16 or 32, both achieving 87.96% accuracy.

TABLE I
LoRA Performance Across Ranks

Rank	4	8	16	32
Accuracy (%)	87.84	87.84	87.96	87.96
Trainable Params (M)	0.67	0.74	0.89	1.18
Peak Mem (MB)	810.0	811.2	813.7	818.6
Throughput (samp/s)	1413.6	1411.2	1392.6	1395.9
Training Time (s)	142.9	143.2	145.1	144.7

Despite using a 4× larger batch size than the baseline, LoRA achieves 38% lower peak memory usage and 3.6× higher throughput. Training time is reduced by 72% (143s vs 518s). The step latency increases slightly to 19.8-20.1ms (median), reflecting the larger batch size rather than computational overhead.

3) QLoRA results and memory/throughput tradeoffs: We implemented QLoRA using 4-bit NormalFloat quantization via the BitsAndBytes library, combined with LoRA adapters applied to query and value projection matrices. Four rank configurations (4, 8, 16, 32) were evaluated to understand the accuracy-efficiency frontier.

Memory Efficiency: QLoRA achieved a 47% reduction in peak GPU memory compared to the full fine-tuning baseline, decreasing from 1,840 MB to approximately 688 MB (measured via PyTorch). Using NVML measurements, peak memory ranged from 1,842 MB to 1,850 MB across ranks, representing a more conservative but still substantial memory reduction. This substantial reduction is attributable to two factors: (1) 4-bit quantization of the frozen base model weights, and (2) training only 1.33M parameters (2.8% of the full model) at rank 8. The memory savings enable fine-tuning on consumer-grade GPUs with 12GB VRAM, such as the NVIDIA RTX 3060, which would be infeasible with full fine-tuning of larger models.

Accuracy-Rank Analysis: Table II presents accuracy results across different LoRA ranks. Rank 4 achieved 87.73% accuracy, rank 8 reached 88.07%, rank 16 achieved 87.96%, and rank 32 attained the highest accuracy at 88.19%. The marginal difference between rank 8 and rank 32 (0.12 percentage points) suggests diminishing returns beyond rank 8, as higher ranks provide minimal accuracy gains while increasing parameter count by 33% (1.33M to 1.77M). Compared to the 92% baseline accuracy, QLoRA at rank 32 retains 95.9% of the original performance while training 97.4% fewer parameters. Rank 8 represents an optimal balance with 88.07% accuracy (95.7% retention) and only 1.33M trainable parameters.

TABLE II
QLoRA Performance Across Ranks

Rank	Accuracy (%)	Trainable Params	Peak Mem (MB)	Throughput (samp/s)
4	87.73	1.26M	686.95	1,095.1
8	88.07	1.33M	688.18	1,114.9
16	87.96	1.48M	690.64	1,136.2
32	88.19	1.77M	695.56	1,091.2
Baseline	**92.0**	**67M**	**1,840**	**∼400**

Training Throughput: QLoRA demonstrated throughput between 1,091 and 1,136 samples per second across the four rank configurations, with rank 16 achieving the highest throughput at 1,136 samples/s. Total training time for three epochs ranged from 177.8 to 185.2 seconds, with an average of approximately 182 seconds. Despite the computational overhead of on-the-fly dequantization from 4-bit to 16-bit precision during forward and backward passes, the reduced parameter count and gradient computation more than compensate for this cost when compared to full fine-tuning baselines.

Memory-Accuracy Tradeoff: Figures 1 and 2 collectively illustrate the trade-off between memory usage and accuracy retention. QLoRA occupies a favorable position: it uses approximately half the memory of the baseline (when measured via PyTorch memory tracking) while maintaining 95.7–95.9% accuracy retention. This positions QLoRA as particularly suitable for memory-constrained environments where full fine-tuning is prohibitive.

System-Level Implications: The 47% memory reduction achieved by QLoRA has significant practical implications. For DistilBERT (67M parameters), this enables deployment on GPUs with as little as 2GB VRAM when considering PyTorch's memory tracking. When scaled to larger models such as LLaMA-2 7B (requiring 28GB for full fine-tuning), QLoRA's compression could reduce requirements to under 15GB, bringing 7B-parameter fine-tuning within reach of consumer hardware. This democratizes access to large language model fine-tuning for researchers and practitioners without access to enterprise-grade compute resources.

Comparison with Baseline:

Across SST-2 experiments with DistilBERT, full fine-tuning achieves the best evaluation accuracy (91.86%, Fig. 1), but it also requires the highest peak GPU memory (1306 MB, Fig. 2) and updates the full parameter set (∼67M, Fig. 17). In contrast, LoRA and QLoRA achieve slightly lower accuracy (best observed: 87.96% for LoRA and 88.19% for QLoRA, Fig. 1) while substantially reducing resource usage: LoRA lowers peak memory to roughly 814 MB (about 38% less than full fine-tuning, Fig. 2) and QLoRA lowers it further

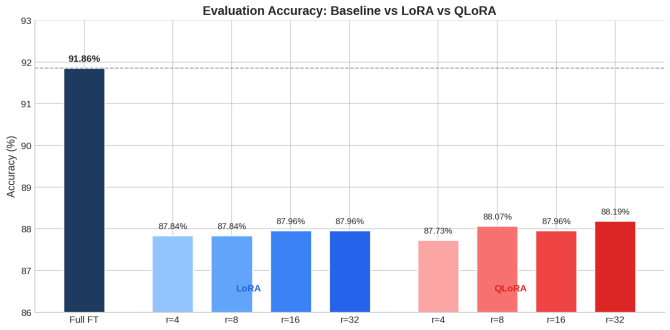

Fig. 1. Evaluation accuracy on SST-2 for full fine-tuning vs. LoRA/QLoRA across ranks. Full fine-tuning achieves the highest accuracy, while LoRA/QLoRA cluster around ∼88% with small rank-dependent variation.

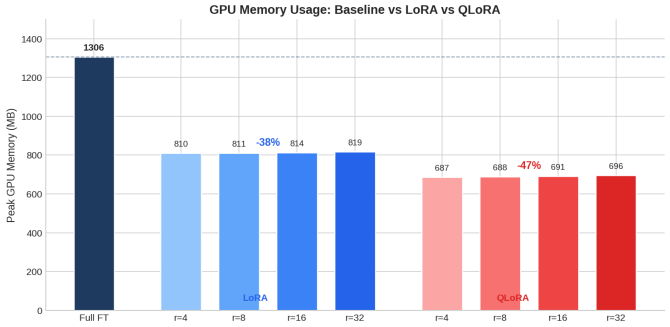

Fig. 2. Peak GPU memory usage (MB). LoRA reduces memory compared to full fine-tuning, and QLoRA (4-bit) reduces it further, demonstrating the efficiency benefits of quantized adapters.

to roughly 696 MB (about 47% less, Fig. 2). These gains are paired with a dramatic reduction in trainable parameters, from updating all weights to training only adapters (e.g., ∼0.89M for LoRA at r=16 and ∼1.77M for QLoRA at r=32, Fig. 17), highlighting the practical efficiency of parameter-efficient and quantized fine-tuning.

C. Analysis of Results

To understand the system-level drivers behind the performance metrics, we analyzed execution traces using the PyTorch Profiler. The profiling results highlight three distinct behaviors governing the efficiency of each method. Table III summarizes the key performance differences observed during profiling.

TABLE III
PERFORMANCE PROFILE: BASELINE VS. PEFT STRATEGIES

Method	Throughput (Speed)	Peak VRAM (Memory)	Trainable Params
Baseline (Full FT)	11.78 it/s	2.04 GB	100% (67M)
LoRA (Rank 8)	**74.11 it/s**	2.04 GB	1.1% (0.7M)
QLoRA (4-bit)	20.73 it/s	**1.4 GB**	1.1% (0.7M)

1) Gradient Computation and Memory Bandwidth: The significant throughput advantage of LoRA (approx. 3.6× over baseline, see Table III) is driven by the sparsity of gradient

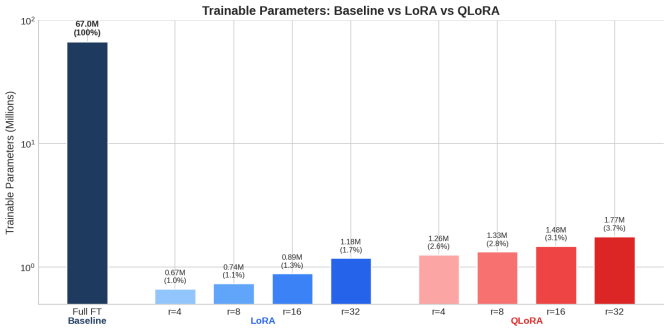

Fig. 3. Trainable parameters (log scale). Full fine-tuning updates the entire model (∼67M parameters), while LoRA/QLoRA update only adapter parameters (sub-2M in these runs), yielding orders-of-magnitude fewer trainable weights.

computation. In full fine-tuning, the backward pass requires calculating gradients for all 66.9M parameters, saturating the GPU memory bandwidth as dense matrices are moved between High Bandwidth Memory (HBM) and Compute Units.

In contrast, our LoRA profiling traces reveal that the backward pass is dominated by lightweight matrix multiplications associated with the low-rank adapter layers (A and B). Since gradients are only computed for $< 1.2\%$ of the parameters, memory traffic is drastically reduced. This shifts the workload from being memory-bound (Baseline) to being compute-bound (LoRA), allowing for much higher utilization of the GPU cores.

2) The Dequantization Overhead in QLoRA: While QLoRA is the most memory-efficient method, our results show it achieves lower throughput than standard LoRA (1091 vs. 1395 samples/s). Profiling reveals this latency is introduced by the `dequantize` kernels.

Unlike standard LoRA, where base weights are static in FP16, QLoRA requires the 4-bit base weights to be dequantized to BF16 on-the-fly for every forward and backward pass to perform matrix multiplication. This introduces a computational overhead that acts as a bottleneck. However, as shown in the results, QLoRA still maintains a $2.8\times$ speedup over the baseline, indicating that the reduction in gradient memory traffic outweighs the cost of dequantization.

3) Memory Hierarchy: Weights vs. Activations: The memory savings in QLoRA are non-linear with respect to model size. For DistilBERT, we observed a peak memory reduction of 47% (dropping to $\approx$ 1.4 GB). Profiling indicates that for smaller models, the memory footprint is dominated by *activations* (intermediate states stored for backpropagation) rather than model weights.

Consequently, while QLoRA compresses weights by $4\times$ (FP16 to NF4), the total system memory does not drop by $4\times$ because the activation memory remains constant across all methods (given equal batch sizes). We project that for larger models (e.g., LLaMA-7B), where weight memory dominates activation memory, the relative savings of QLoRA would increase significantly, making it the only viable option for consumer hardware.

V. Discussion

A. Interpretation of Results

Our experiments suggest that PEFT methods function effectively as a regularization mechanism. While the Baseline model achieved the highest training fidelity due to its unconstrained optimization space, the rapid convergence of LoRA and QLoRA implies that freezing the pre-trained backbone preserves generalizable representations. Our results align with findings on catastrophic forgetting [10], suggesting that PEFT methods act as a regularizer by keeping the vast majority of weights fixed, thereby retaining the pre-trained capabilities better than aggressive full fine-tuning.

From a systems engineering perspective, the results establish a clear Pareto frontier regarding resource allocation, as detailed in Table III:

- **Throughput Optimization (LoRA):** For latency-critical training pipelines, **LoRA** is the optimal strategy. It achieved a throughput of **74.11 it/s** (a $6.3\times$ speedup over the Baseline's 11.78 it/s), as shown in Table III. By circumventing the gradient computation for 99% of the model, LoRA maximizes GPU compute utilization, making it the superior choice for rapid experimentation and iterative development.
- **Memory Optimization (QLoRA):** For hardware-constrained environments, **QLoRA** is the essential choice. It successfully reduced peak VRAM usage to **1.4 GB**, a $\approx 30\%$ reduction compared to the 2.04 GB required by the Baseline and LoRA. Although this comes with a throughput penalty (dropping to 20.73 it/s due to dequantization overhead), it remains nearly $2\times$ faster than full fine-tuning while enabling deployment on consumer-grade GPUs with limited memory capacity.
- **Accuracy Maximization (Full Fine-Tuning):** Full fine-tuning is only justified in scenarios where hardware resources are unconstrained (e.g., A100 clusters) and the absolute maximum accuracy is required, regardless of the significant computational cost and training time.

While our primary focus was on latency and throughput, the reduction in computational overhead is directly linked to Green AI principles [11]. By reducing the memory bandwidth requirements and training time, PEFT methods address the growing concern of the carbon footprint associated with training large transformer models [12].

B. Comparison with Previous Studies

Our findings are broadly consistent with prior studies showing that parameter-efficient fine-tuning (PEFT) methods can achieve competitive downstream performance while substantially reducing training cost and memory usage. In particular, our accuracy trends across LoRA and QLoRA rank configurations align with earlier observations that low-rank adaptations preserve a large fraction of the representational capacity of pretrained transformers while updating only a small subset of parameters [4], [6].

Fig. 4. Our edge device (Raspberry Pi) with and attempt to run a quantized model.

However, most existing work evaluates PEFT methods primarily through task-level metrics such as accuracy, parameter count, or peak memory usage, often reported in isolation. In contrast, our study extends these findings by incorporating a system-level perspective that jointly examines training throughput, memory behavior, and operator-level execution characteristics. Through detailed profiling, we demonstrate that the efficiency gains of PEFT methods arise not only from parameter reduction, but also from fundamental shifts in computational behavior, specifically, reduced gradient computation and altered memory bandwidth utilization.

Moreover, while prior QLoRA studies emphasize its ability to enable fine-tuning of very large models on limited hardware, our results highlight the practical trade-offs introduced by on-the-fly dequantization, which can reduce throughput relative to standard LoRA despite superior memory efficiency. By explicitly quantifying these effects, our work complements existing literature by providing a more nuanced understanding of how algorithmic efficiency choices manifest at runtime.

Overall, this study reinforces the effectiveness of PEFT methods reported in previous work, while contributing additional insight into their system-level implications, which are often underexplored in accuracy-centric evaluations.

C. Challenges and Limitations

A primary challenge encountered in this project was the significant engineering overhead associated with deploying and benchmarking models on true edge devices. Initial attempts to target Raspberry Pi hardware exposed substantial practical limitations, including restricted memory capacity, limited support for optimized deep learning kernels, and compatibility issues with modern training frameworks and profiling tools. These constraints made controlled experimentation and fair comparison across fine-tuning strategies difficult. Figure 4 showcases our attempt to run a quantised model on our Raspberry Pi

As a result, we adopted GPU memory usage and training latency as proxies for constrained environments, allowing for reproducible and systematic evaluation while maintaining relevance to resource-limited settings. While this approach

captures key aspects of hardware constraints, such as memory pressure and throughput, it does not fully reflect the heterogeneity of real-world edge devices, particularly in terms of CPU-bound execution, power consumption, and thermal limits.

Additionally, our experiments are limited to a single dataset (SST-2) and a single model architecture (DistilBERT). Although this choice enables controlled comparisons, it restricts the generalizability of the conclusions to other tasks, modalities, or larger model families. Finally, profiling runs were intentionally short to avoid perturbing performance measurements, which may limit the visibility of longer-term effects such as memory fragmentation or sustained thermal throttling.

These limitations highlight the trade-off between experimental control and deployment realism, and motivate further investigation beyond the scope of this study.

D. Future Directions

Several directions emerge for future work building on the findings of this study. First, extending the analysis to inference-time performance would provide a more complete understanding of the end-to-end efficiency of PEFT methods, particularly for latency-sensitive applications. Techniques such as adapter fusion, kernel optimization, and quantized inference could be evaluated to assess their impact on deployment efficiency.

Second, future studies could explore deployment on heterogeneous and truly resource-constrained hardware platforms, including CPUs, mobile accelerators, and edge GPUs. Evaluating PEFT methods across diverse hardware backends would help bridge the gap between controlled benchmarking and real-world deployment scenarios.

Finally, an important avenue for future research is the automated selection of fine-tuning strategies based on hardware constraints and application requirements. Profiling-guided or hardware-aware adaptation frameworks could dynamically choose between full fine-tuning, LoRA, and QLoRA configurations to optimize for accuracy, memory usage, or throughput. Such approaches would further democratize access to large language model adaptation by reducing the expertise required to select appropriate fine-tuning strategies.

VI. Conclusion

A. Summary of Findings

This project systematically evaluated full fine-tuning, LoRA, and QLoRA under controlled experimental conditions, with an emphasis on both model performance and system-level efficiency. Our results show that parameter-efficient fine-tuning methods can preserve a large fraction of baseline accuracy while dramatically reducing training time, memory usage, and the number of trainable parameters.

LoRA emerged as the most effective approach for maximizing training throughput, achieving multi-fold speedups over full fine-tuning by eliminating gradient computation for the majority of model parameters. QLoRA further reduced peak memory usage through aggressive 4-bit quantization, enabling fine-tuning in memory-constrained settings at the cost of moderate throughput degradation due to dequantization overhead. Collectively, these findings highlight that the choice of fine-tuning strategy induces distinct computational regimes, memory-bound versus compute-bound, rather than simple linear trade-offs in efficiency.

B. Contributions

This work makes three primary contributions. First, it provides a controlled and reproducible comparison of full fine-tuning, LoRA, and QLoRA on a standardized NLP benchmark, isolating the effects of parameter efficiency from confounding factors such as data preprocessing and optimization settings.

Second, we integrate operator-level profiling into the evaluation pipeline, enabling direct analysis of gradient computation, memory bandwidth usage, and kernel execution behavior. This system-level perspective offers explanatory insight into why parameter-efficient methods achieve their observed efficiency gains, moving beyond surface-level metrics such as parameter count or peak memory alone.

Third, the study characterizes practical trade-offs between throughput, memory efficiency, and accuracy retention, offering actionable guidance for selecting fine-tuning strategies based on hardware constraints and performance objectives.

C. Recommendations for Future Research

Future research should extend this analysis to inference-time performance, where latency, kernel fusion, and quantized execution play a dominant role in real-world deployment. Evaluating PEFT methods under inference constraints would complement the training-focused perspective adopted in this study.

Additionally, deploying and benchmarking PEFT strategies across heterogeneous hardware platforms including CPUs, mobile accelerators, and low-power edge devices would improve understanding of their practical limitations and generalizability. Finally, integrating profiling-driven decision mechanisms into fine-tuning pipelines could enable automated selection of adaptation strategies, dynamically balancing accuracy, memory usage, and throughput based on available hardware resources and application requirements.

References

[1] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of NAACL-HLT*, 2019.

[3] N. Houlsby, A. Giurgiu, S. Jastrzebski *et al.*, "Parameter-efficient transfer learning for nlp," *Proceedings of ICML*, 2019.

[4] 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," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[5] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[6] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Proceedings of NeurIPS*, 2023.

[7] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[8] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, T. Linzen, G. Chrupała, and A. Alishahi, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. [Online]. Available: https://aclanthology.org/W18-5446/

[9] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2020. [Online]. Available: https://arxiv.org/abs/1910.03771

[10] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, p. 3521–3526, Mar. 2017. [Online]. Available: http://dx.doi.org/10.1073/pnas.1611835114

[11] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," 2019. [Online]. Available: https://arxiv.org/abs/1907.10597

[12] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," 2019. [Online]. Available: https://arxiv.org/abs/1906.02243

APPENDIX

A. LoRA Analysis Visualizations

This appendix presents detailed visualizations of the LoRA (Low-Rank Adaptation) performance analysis across different model ranks and configurations.

1) Comprehensive Overview: Figure 12 provides a comprehensive overview of all key metrics including accuracy, parameter efficiency, memory usage, and training throughput across different LoRA rank configurations.

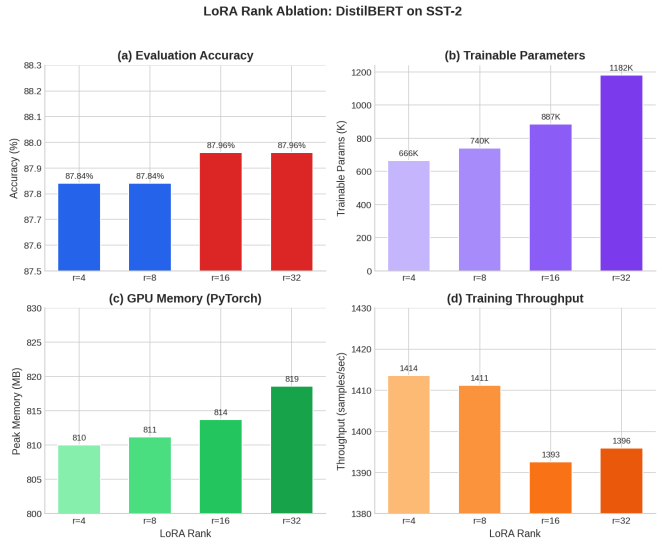

Fig. 5. Comprehensive overview of LoRA performance metrics across different rank configurations.

2) Accuracy vs Parameter Count: Figure 13 illustrates the relationship between model accuracy and the number of trainable parameters for different LoRA ranks.

3) Evaluation Accuracy by Rank: Figure 14 shows the evaluation accuracy achieved by different LoRA rank configurations during training.

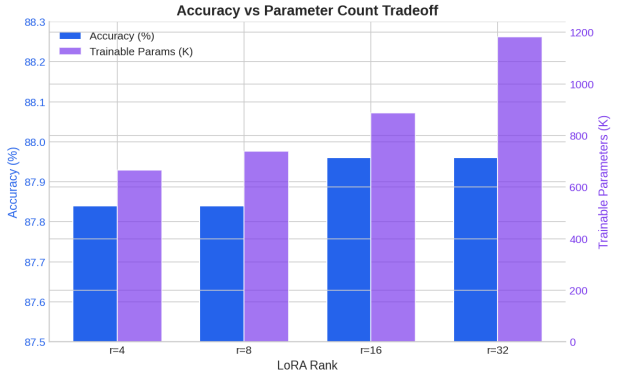

Fig. 6. Accuracy versus trainable parameter count for different LoRA ranks.

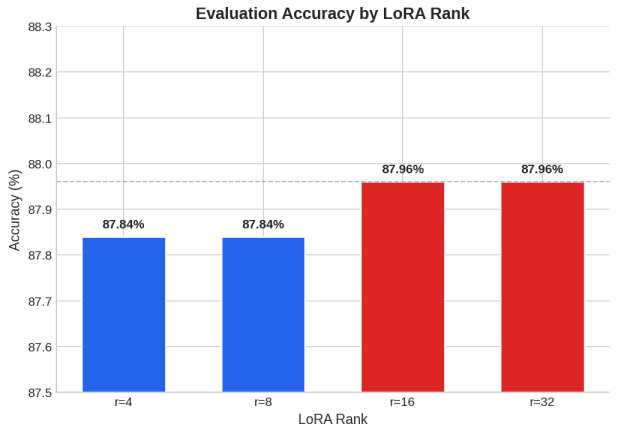

Fig. 7. Evaluation accuracy by LoRA rank.

4) Parameter Efficiency: Figure 15 demonstrates the parameter efficiency of different LoRA ranks, showing the trade-off between model performance and parameter count.

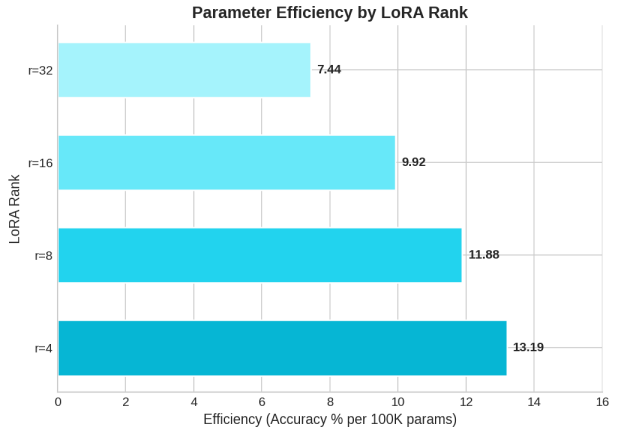

Fig. 8. Parameter efficiency analysis by LoRA rank.

5) Memory Consumption: Figure 16 presents the PyTorch memory consumption for different LoRA rank configurations during training.

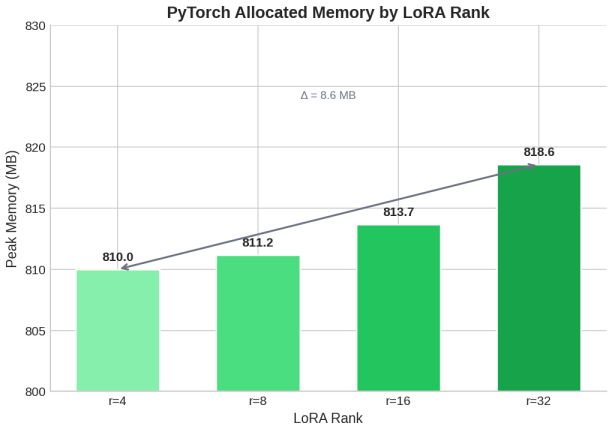

Fig. 9. PyTorch memory usage by LoRA rank.

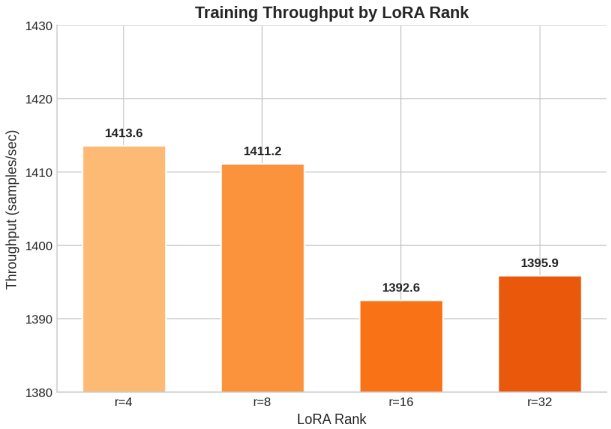

Fig. 11. Training throughput by LoRA rank.

6) Trainable Parameters: Figure 17 shows the number of trainable parameters for each LoRA rank configuration.

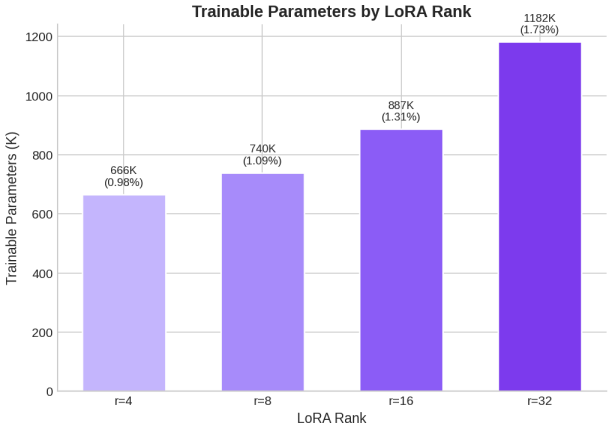

Fig. 10. Number of trainable parameters by LoRA rank.

7) Training Throughput: Figure 18 illustrates the training throughput (samples per second) achieved by different LoRA rank configurations.

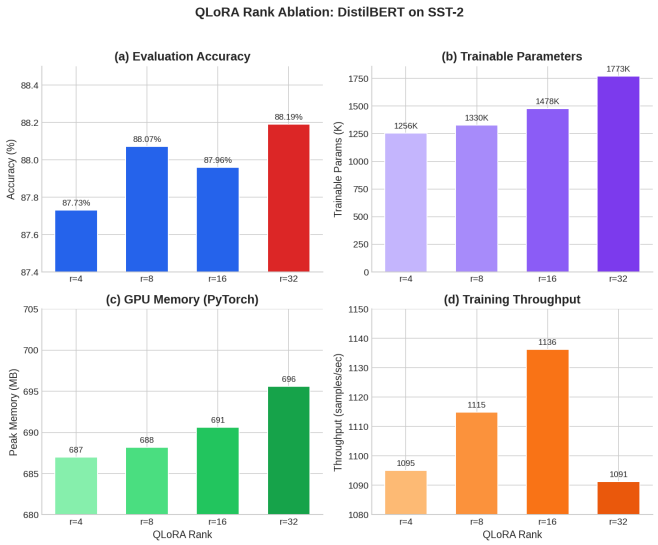

Fig. 12. Comprehensive overview of qLoRA performance metrics across different rank configurations.

B. qLoRA Analysis Visualizations

This appendix presents detailed visualizations of the qLoRA (Quantized Low-Rank Adaptation) performance analysis across different model ranks and configurations.

1) Comprehensive Overview: Figure 12 provides a comprehensive overview of all key metrics including accuracy, parameter efficiency, memory usage, and training throughput across different qLoRA rank configurations.

2) Accuracy vs Parameter Count: Figure 13 illustrates the relationship between model accuracy and the number of trainable parameters for different qLoRA ranks.

3) Evaluation Accuracy by Rank: Figure 14 shows the evaluation accuracy achieved by different qLoRA rank configurations during training.

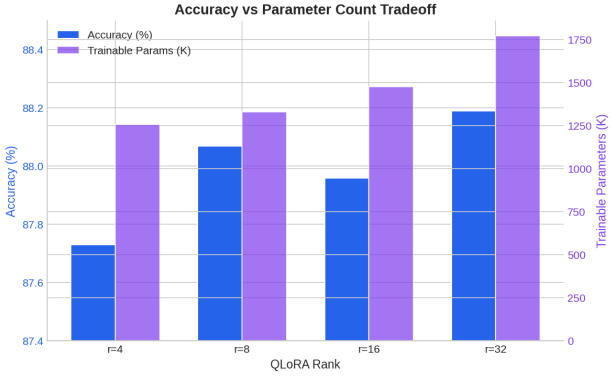

Fig. 13. Accuracy versus trainable parameter count for different qLoRA ranks.

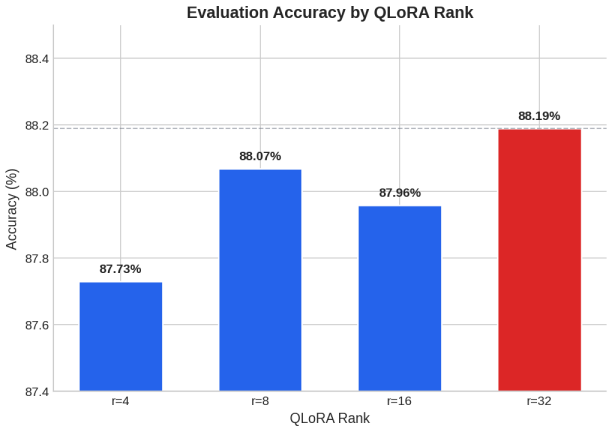

Fig. 14. Evaluation accuracy by qLoRA rank.

4) Parameter Efficiency: Figure 15 demonstrates the parameter efficiency of different qLoRA ranks, showing the trade-off between model performance and parameter count.

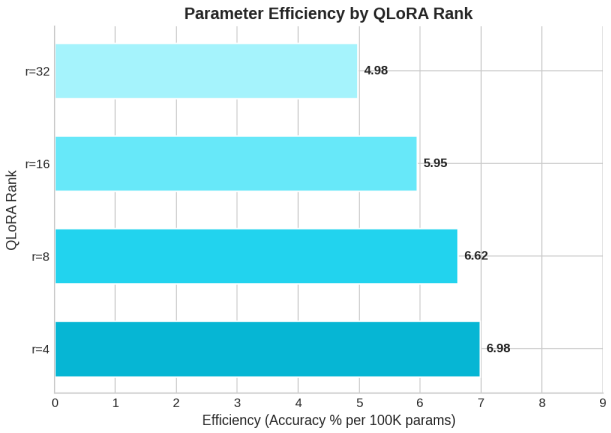

Fig. 15. Parameter efficiency analysis by qLoRA rank.

5) Memory Consumption: Figure 16 presents the PyTorch memory consumption for different qLoRA rank configurations during training.

6) Trainable Parameters: Figure 17 shows the number of trainable parameters for each qLoRA rank configuration.

7) Training Throughput: Figure 18 illustrates the training throughput (samples per second) achieved by different qLoRA rank configurations.

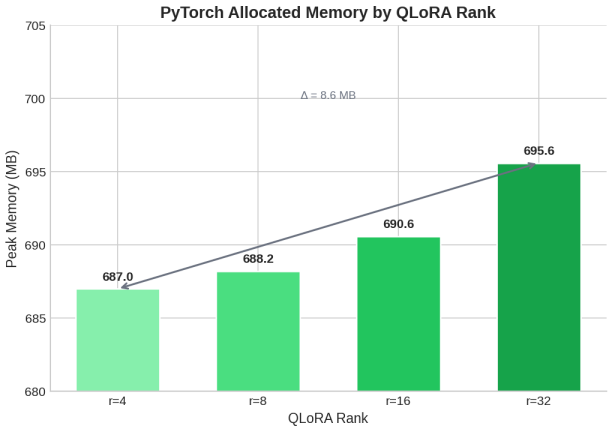

Fig. 16. PyTorch memory usage by qLoRA rank.

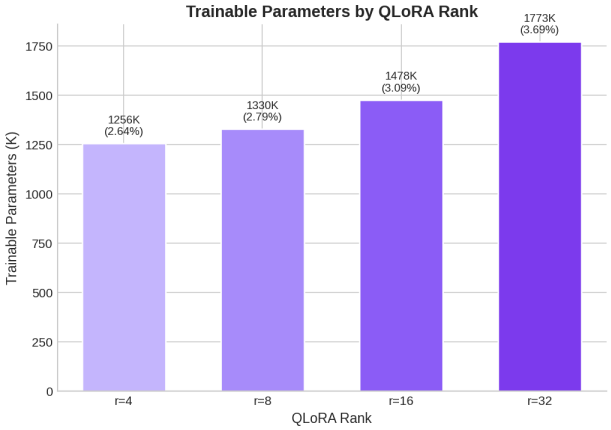

Fig. 17. Number of trainable parameters by qLoRA rank.

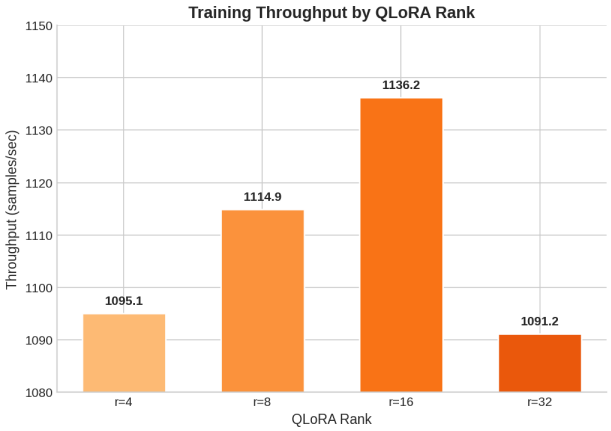

Fig. 18. Training throughput by qLoRA rank.