Optimizing the Fine-tuning Process of Large Language Models

Mahbub Islam Mahim and Jugal Krishna Das
Department of Computer Science and Engineering, Jahangirnagar University, Savar, Dhaka-1342
Correspondence E-mail: mahim.stu20171@juniv.edu

Abstract—We present an optimized fine-tuning process for large language models (LLMs) that combines Low-Rank Adaptation (LoRA) and Quantization. Traditional full fine-tuning methods are computationally expensive, requiring significant GPU memory, which limits their accessibility. In our approach, we first quantize the LLaMA-2 7B model and then apply LoRA fine-tuning to that quantized model. We demonstrate that the combination of quantization and LoRA significantly reduces GPU memory requirements while maintaining model performance. Through rigorous experiments of different combinations of quantization and LoRA, we successfully fine-tuned the 7B LLaMA-2 model on CodeAlpaca-20k dataset with only 10.8 GB of GPU memory using the best combination of quantization (4-bit) and LoRA (rank 16), compared to the 112 GB required by traditional methods. We further developed an inference system using this optimized fine-tuned model for practical deployment.

***Index Terms*— Generative Artificial Intelligence, Low-Rank Adaptation, Quantization, Optimization, LLaMA-2, Code Alpaca.**

I. Introduction

The rapid evolution of large language models (LLMs) has revolutionized natural language processing, allowing significant advancements across a myriad of applications. Refining LLMs by finetuning is well acknowledged for greatly boosting their performance [1-4] on specific knowledge and for adjusting behaviors to accomplish desired results [4-6]. Despite their impressive capabilities, the substantial computational and memory demands associated with fine-tuning these models present a formidable challenge [7]. For instance, conventional 16-bit fine-tuning of a LLaMA 65B parameter model [8] takes more than 780 GB of GPU memory and a 7B parameter model takes 112 GB. So, optimizing the fine-tuning process is crucial for leveraging LLMs in resource-constrained environments.

This paper explores two key methodologies-model quantization and Low-Rank Adaptation (LoRA)-that address this challenge by reducing the computational burden while maintaining model performance [9-10]. Model quantization reduces the precision of model weights, thus significantly decreasing memory usage and increasing inference speed, while LoRA achieves efficient fine-tuning by approximating weight updates with low-rank matrices. Through a detailed examination of these techniques and their integration, this research aims to enhance the efficiency of fine-tuning large language models, paving the way for broader accessibility and practical deployment in various domains.

II. Literature Review

Recent advancements in large language models (LLMs) have led to significant improvements in natural language processing tasks. However, the computational and memory requirements for fine-tuning these models remain substantial. This review examines current methodologies for optimizing the fine-tuning process, focusing on model quantization and Low-Rank Adaptation. Model quantization reduces the precision of the weights and activations, typically from 32-bit floating-point to lower bit-width representations such as 16-bit, 8-bit, or NF4-bits. Quantization can significantly decrease the model size and increase inference speed with minimal loss in accuracy. Some research [9] demonstrates that post-training quantization can yield up to a 4x reduction in model size with negligible performance degradation. Mixed-precision training [11] combines high and low-precision operations to balance speed and accuracy effectively.

LoRA is a widely used PEFT [12] technique for optimizing LLM because of its wide range of applications and strong results as compared to other approaches. It focuses on efficient fine-tuning by approximating the weight updates using low-rank matrices. LoRA showed that it could drastically reduce the number of trainable parameters while maintaining model performance [10]. It inserts low-rank decomposition matrices into each layer of the transformer architecture, enabling the adaptation of large pre-trained models with a fraction of the original parameters, thus making the fine-tuning process more efficient and cost-effective. Further research has been conducted to optimize LoRA. For instance, DoRA [13] decomposes the original weight into magnitude and direction components, updating the direction component using LoRA. LoRA+ [14] employs different learning rates for the two low-rank matrices to enhance learning efficiency. Additionally, ReLoRA [15] incorporates LoRA into the LLM during training to raise the rank of the final ΔW. QLoRA [16] extends LoRA by integrating 4-bit quantization, further reducing memory usage and enabling fine-tuning of larger models on resource-constrained hardware.

These studies indicate that integrating these techniques can maintain acceptable accuracy while significantly reducing resource requirements. Research by [17] on Q8BERT, which integrates quantization with other optimization techniques, supports the potential for such combined approaches. The integration of model quantization and LoRA represents a promising direction for optimizing the fine-tuning of LLMs. Future research should focus on developing robust techniques to combine these methods effectively, ensuring minimal trade-offs between efficiency and model performance.

III. Methodology

Dataset Engineering begins the process. This phase is essential for preparing the data used to fine-tune the model. It involves dataset collection, data cleaning, and data formatting. Dataset collection entails gathering a comprehensive and representative set of textual data relevant to the task. Depending on the target

application, this data might be sourced from various domains. Since we aim to fine-tune by coding dataset, we collect "CodeAlpaca-20k" [18] from Hugging Face. Following collection, data cleaning processes are implemented to remove noise, correct errors, and filter out irrelevant content, ensuring the quality and relevance of the dataset. Data formatting then converts this cleaned data into a format compatible with the LLaMA2 model, typically involving tokenization and the creation of input-output pairs suitable for the training process. Once the dataset is prepared, the next step involves loading the pre-trained LLaMA2 model. LLaMA2, like other large language models, is trained on extensive corpora to capture diverse linguistic patterns and knowledge. However, for tasks like sentiment analysis, coding answers, or machine translation, fine-tuning is necessary to adapt its generalized knowledge.

LoRA (Low-Rank Adaptation) efficiently facilitates this fine-tuning process. Before applying LoRA, the LLaMA2 model undergoes Quantization to reduce its computational complexity and memory footprint. Quantization involves converting the high precision floating-point weights of the model into lower precision representations.

We experimented with different quantization techniques, such as 32-bit floating point to 16- bit, 8-bit, and NF4-bit. These techniques help in significantly reducing the memory requirements while retaining the model's performance. Let W represents the original weight matrix and Q(W) the quantized weights. The quantization can be represented as:

$$Q_{16\text{-bit}}(W) = \text{Quantize}_{32\text{-to-}16\text{-bit}}(W)$$
$$Q_{8\text{-bit}}(W) = \text{Quantize}_{32\text{-to-}8\text{-bit}}(W)$$
$$Q_{4\text{-bit}}(W) = \text{Quantize}_{32\text{-to-NF4-bit}}(W)$$

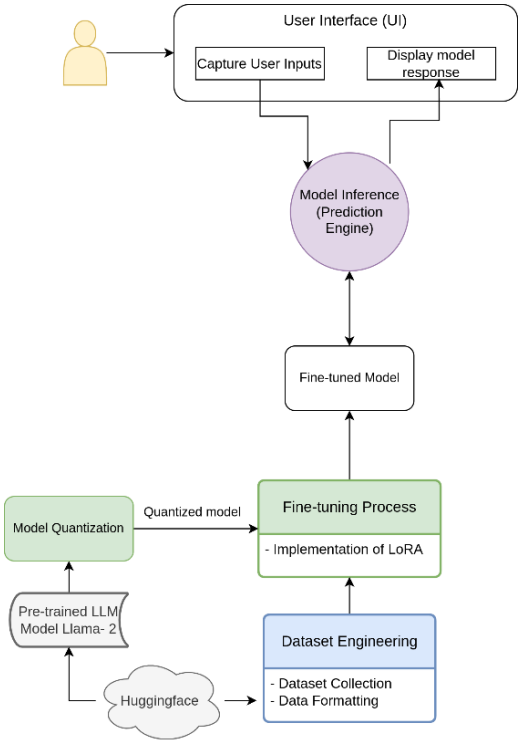

Fig.1. Overview of our method

where 'Quantize' is the function that maps the high-precision weights to lower precision. After quantization, the weights are more memory-efficient, enabling faster computations and reduced hardware requirements. With the model quantized, LoRA introduces trainable low-rank matrices into each layer of the transformer architecture (decoder only) of the model. Instead of updating all the parameters of the already large and complex model, It focuses on a smaller set of additional parameters. This approach is based on the observation that changes required for task-specific fine-tuning can often be represented as low-rank updates to the pre-trained weights. Let W_q represents the quantized weight matrix and A and B be the low-rank matrices. The fine-tuning update can be expressed as-

$$W_{new} = W_q + \Delta W$$

$$\text{Or, } W_{new} = W_q + AB^T$$

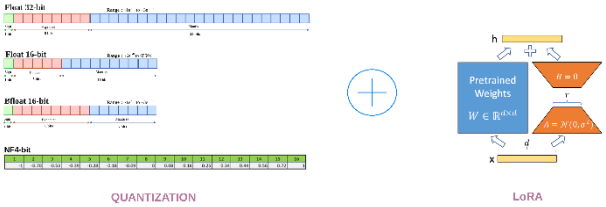

Fig. 2. Combination of Quantization and LoRA

By only updating these low-rank matrices, we significantly reduce the number of parameters that need to be trained, thus saving computational resources and reducing memory usage. During the fine-tuning process of the LLaMA2 model, we keep the original pre-trained weights frozen. We experiment with different ranks, such as 1, 2, 4, 8, 16, and 64, for the low-rank matrices to balance the trade-off between model adaptation and computational efficiency. The input data is passed through the model, and gradients are calculated only for the parameters within the LoRA matrices. This selective updating is highly efficient and ensures that the pre-trained knowledge encoded in the original model is preserved while adapting the model to new tasks. The training typically involves optimizing a loss function, such as cross-entropy, that quantifies the gap between the projected outputs and the actual target values. The cross-entropy loss L for a set of predictions $\hat{y}$ and true labels y is given by-

$$L = -\sum_i y_i \log(\hat{y}_i)$$

Gradient descent algorithms, such as Adam, are used to minimize this loss function-

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L$$

where θ represents the model parameters, η is the learning rate, and $\nabla_\theta L$ is the gradient of the loss function with respect results to the parameters.

Regularization strategies, such as dropout or weight decomposition, are typically applied during training to minimize overfitting. Overfitting happens when the model learns to perform very well on the training data but fails to generalize to new, unknown data. Dropout, in particular, is a strategy where random neurons are momentarily deleted from the network during training, driving the model to acquire more robust and generalizable characteristics. Hyperparameters, such as learning rate, batch size, and the rank of the adaptation matrices, can considerably affect the training efficiency and the final model performance. Each combination of hyperparameters is assessed on a validation set, and the best-performing configuration is picked for the final training run.

After the training phase, the fine-tuned model undergoes rigorous evaluation using a validation set that was not seen during training. This evaluation helps in assessing the model's performance on the specific task. If the performance is unsatisfactory, iterative adjustments are made, including re-tuning hyperparameters, modifying the dataset, or even altering the model architecture by adjusting the size or number of LoRA matrices. Finally, the model is tested on a separate test set to ensure its robustness and generalizability. The performance on the test set provides a final assessment of the model's capability to handle the specific task effectively. Following fine-tuning, the resultant fine-tuned LLM can process user inputs, generate relevant responses, and be ready for inference tasks. This fine-tuned model incorporates the specific knowledge and patterns required for the target application (Coding Assistant), ensuring it can handle the specific queries and tasks effectively.

The final phase involves integrating the fine-tuned LLM into a user-facing application. In this case, we utilize a Gradio web interface as the user interface (UI). The UI is designed to capture user inputs and display the model's responses. When a user interacts with the UI, their input is sent to the model inference (prediction engine), which processes the input using the fine-tuned LLM. The prediction engine utilizes the fine-tuned model to generate and return responses based on the user's input, ensuring an interactive and responsive user experience. Additionally, we have implemented prompt engineering and session holding to maintain context and continuity during user interactions, further enhancing the usability and effectiveness of the application.

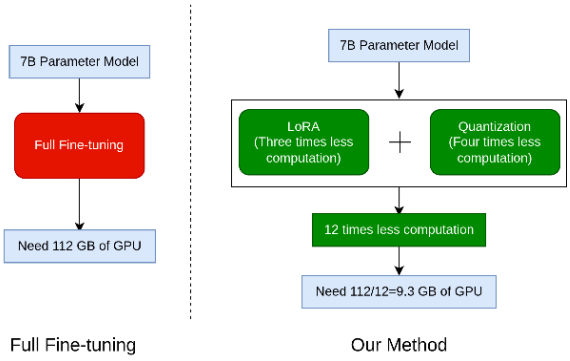

Fig.3. Comparison of GPU memory requirements

Figure 3 illustrates the GPU memory requirements for traditional full fine-tuning of a 7B parameter model compared to our optimized method using LoRA and quantization. The optimized method significantly reduces computational needs, making it 12 times more efficient in terms of GPU memory requirements.

IV. Results and Evaluation

The evaluation examines key performance metrics, including GPU usage, learning rate dynamics, training loss, and overall model performance, to assess the efficiency of fine-tuning. We conducted experiments across various configurations of quantization and LoRA to analyze trade-offs in memory efficiency, training time, and model accuracy. The study explores three setups: (i) Quantization Only, applying 4-bit and 8-bit post-training quantization without fine-tuning; (ii) LoRA Only, evaluating different LoRA ranks (1, 4, 8, 16, 64) without quantization; and (iii) Combined Quantization + LoRA. We integrated both quantization (4-bit and 8-bit) with LoRA fine-tuning at different ranks, aiming to achieve optimal trade-offs between memory reduction and performance retention.

GPU memory consumption varied significantly across the tested configurations. Quantization alone reduced memory requirements considerably, with 8-bit quantization lowering the demand to ~28 GB and 4-bit quantization further reducing it to ~15 GB. LoRA-only configurations exhibited varying memory footprints depending on the rank used. Lower-rank LoRA settings such as rank 1, 4 demonstrated minimal overhead compared to the base model, while higher-rank LoRA such as rank 64 incurred additional memory usages. The most significant reduction in memory consumption was observed in the Combined Quantization + LoRA setting, where **4-bit** quantization with LoRA **rank 16** required only **10.8 GB** of GPU memory. This configuration provided the best balance between efficiency and fine-tuning effectiveness. Figure 4 illustrates the memory consumption of the best configuration.

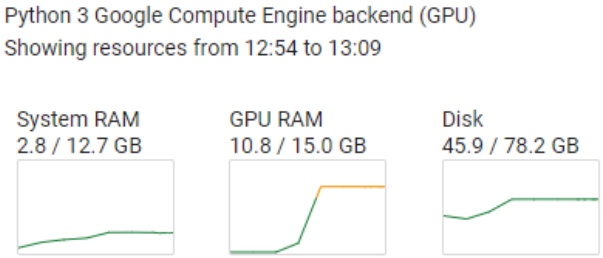

Fig.4. GPU usage during fine-tuning

Additionally, the fine-tuning process of this best combination of LoRA and quantization required nearly 3 hours to complete.

Training stability and convergence rates were monitored across all configurations. Across different setups, lower-rank LoRA settings (ranks 1, 4) converged faster but exhibited limited adaptation, while higher-rank LoRA (ranks 16, 64)

achieved better fine-tuning but at the cost of increased resource usage. Quantization alone (without LoRA) showed a slight degradation in model performance due to reduced numerical precision. However, integrating LoRA effectively compensated for this loss, particularly in the **4-bit** quantization + LoRA **rank 16** configurations, which achieved near-baseline performance while maintaining a lightweight footprint. Figure 5 presents the training loss progression over 5000 steps for the best-performing configuration.

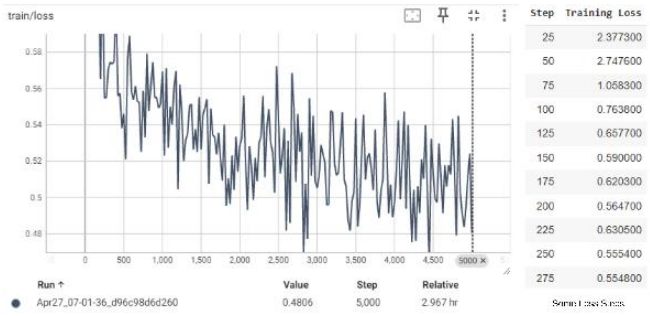

Fig.5. Training loss over steps

Figure 6 shows the learning rate during the fine-tuning process of the best combination **4-bit** quantization + LoRA **rank 16**. It follows a cosine learning rate schedule, which starts high and smoothly decreases over the 5000 training steps. The smooth decay pattern indicates effective learning rate management, leading to faster convergence and better model generalization.

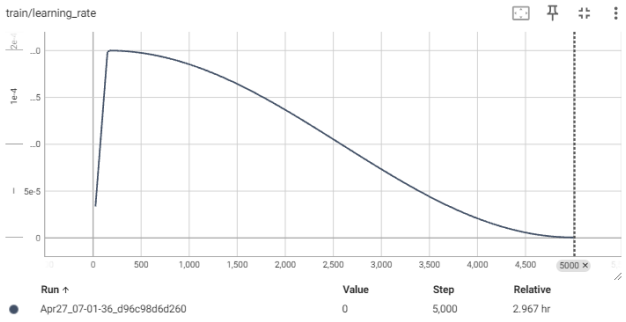

Fig. 6. Learning rate curve

Finally, loading a 7 billion parameter model of LLaMA-2 in FP32 requires approximately 28 GB of GPU memory, and fine-tuning needs around 4 x 28 = 112 GB of GPU memory [8]. In contrast, best configuration of our method needs only **10.8 GB** of GPU memory during fine-tuning. Theoretically, Figure 3 estimates that 9.3 GB of GPU memory is needed to fine-tune the 7B LLaMA-2 model using our method. Practically speaking, from Figure 4, the actual requirement is 10.8 GB, demonstrating the successful implementation of our approach.

V. Conclusion

In conclusion, our study demonstrates that combining quantization and Low-Rank Adaptation significantly reduces GPU memory requirements while maintaining or improving the performance of large language models. We fine-tuned the 7B LLaMA-2 model on code alpaca 20k dataset using only 10.8 GB of GPU memory, compared to the 112 GB needed for traditional methods. This approach enables advanced LLM capabilities in resource-constrained environments, democratizing access to powerful on-device AI tools. The developed user interface for real-time model interaction underscores the practical deployment potential of these techniques, making sophisticated language models accessible for everyday tasks.

References

[1] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi, “Metaicl: Learning to learn in context,” arXiv preprint arXiv:2110.15943, 2021.

[2]] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, “Finetuned language models are zero-shot learners,” arXiv preprint arXiv:2109.01652, 2021

[3] Y. Wang, S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Arunkumar, A. Ashok, A. Selvan Dhanasekaran, A. Naik, D. Stap, and others, “Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks,” arXiv preprint arXiv:2204.07705, 2022.

[4] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, and others, “Training language models to follow instructions with human feedback,” Advances in neural information processing systems, vol. 35, pp. 27730–27744, 2022

[5] A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, and others, “A general language assistant as a laboratory for alignment,” arXiv preprint arXiv:2112.00861, 2021.

[6] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, and others, “Training a helpful and harmless assistant with reinforcement learning from human feedback,” arXiv preprint arXiv:2204.05862, 2022.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv preprint arXiv:1810.04805, 2018.

[8] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, and others, “Llama 2: Open foundation and fine-tuned chat models,” arXiv preprint arXiv:2307.09288, 2023.

[9] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko, “Quantization and training of neural networks for efficient integerarithmetic-only inference,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.

[10] 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.

[11] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and others, “Mixed precision training,” arXiv preprint arXiv:1710.03740, 2017.

[12] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, “PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods,” GitHub repository, 2022. [Online]. Available: https://github.com/ huggingface/peft.

[13] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.- T. Cheng, and M.-H. Chen, “DoRA: Weight-Decomposed Low-Rank Adaptation,” arXiv preprint arXiv:2402.09353, 2024.

[14] S. Hayou, N. Ghosh, and B. Yu, “LoRA+: Efficient Low Rank Adaptation of Large Models,” arXiv preprint arXiv:2402.12354, 2024

[15] V. Lialin, S. Muckatira, N. Shivagunde, and A. Rumshisky, "ReLoRA: High-Rank Training Through Low-Rank Updates," in Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023), 2023.

[16] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," in Advances in Neural Information Processing Systems, vol. 36, 2024.

[17] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," in 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS), pp. 36–39, IEEE, 2019.

[18] S. Chaudhary, "Code Alpaca: An Instruction-following LLaMA model for code generation," GitHub repository, 2023. [Online]. Available: https: //github.com/sahil280114/codealpaca.

Biography

Mahbub Islam Mahim has completed his B.Sc. and M.Sc. in Computer Science and Engineering from Jahangirnagar University, Dhaka, Bangladesh in 2021 and 2023 respectively. He was awarded the Bronze Medal in the International University Physics Competition 2021. During his undergraduate studies, he was actively involved in competitive programming. Currently, he works as a Software Engineer at the Samsung R&D Institute Bangladesh, where he is also engaged in developing patents and other innovation works. His research interests include Advanced Machine Learning, Natural Language Processing, Natural Language Generation, Computer Vision and Image Processing, Large Language Models, Trustworthy and Efficient AI, and Software Engineering.

Dr. Jugal Krishna Das has completed his B.Sc., M.Sc., and Ph.D. degrees in Computer Engineering from Donetsk Technical University, Ukraine. He is currently a Professor in the Department of Computer Science and Engineering at Jahangirnagar University, Dhaka, Bangladesh. Dr. Das's research interests include Computer Networks, Machine Learning, Natural Language Processing, and Software Engineering. His key fields of interest encompass System Design and Analysis, E-commerce, Management Information Systems, Database Management Systems, Computer Architecture, Data Structures and Algorithms, Microprocessors, Computer Networking, Discrete Mathematics, Digital Logic Design, Operating Systems, Parallel and Distributed Systems, Internet Engineering, and IP Technology. He has 48 publications in Computer Science and Engineering in national and international journals and conference proceedings.