Memory Efficient Low-Rank Systems for Large Foundation Models

Ivan Jaen-Marquez*, Laik Ruetten*, Sadman Sakib*, Zheyang Xiong*

Abstract

Deep learning models are becoming bigger, with parameter count rapidly increasing, making it more and more infeasible to train all at once due to hardware constraints on most machines. In this work, we focused on exploring memory efficient systems for pre-training large foundation models through the use of low rank structures. LoRA is first of this kind [1]. More specifically, we primarily focus our analysis on recent strategies such as LTE [2] and GaLore [11] which can achieve parameter and memory efficient pre-training while maintaining model performance. We confirmed that these tools can alleviate the pre-training memory problems using a small number of training nodes. We aimed to understand how these strategies function as claimed by the authors, evaluate their performance ourselves, and eventually combine their complementary advantages for further memory efficiency. Our code is publicly available at - https://github.com/sadmankiba/Lowrank-Training.

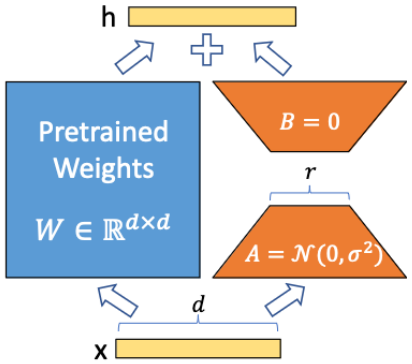
1 Introduction

Training modern neural network-based models is becoming increasingly challenging as there is a tendency to massively increase its number of parameters in order to achieve state-of-the-art performance on multiple tasks.

Parameter efficient techniques allow users to train larger models with small memory loads, and adjusting these models to their needs without requiring to upgrade to expensive hardware. Ideally we can reduce the amount of trainable parameters enough to run on conventional consumer grade GPUs, or even laptop CPUs. This will, hopefully, help further democratize the ability to train large models to a larger portion of the human population by alleviating hardware constraints.

Existing approaches such as the popular Low-Rank Adaptation (LoRA) [1] were originally proposed to fine-tune by training only lower-dimensional matrices. LoRA performs fine-tuning by reparametrizing the weight matrices W as a factorization of two low-rank matrices A and B (Figure 1). LoRA trains less parameters and so needs less memory in training and less space for saving fine-tuned weights.

*Equal contribution



$$h = W_0 x + \Delta W x = W_0 x + BAx$$

Figure 1: LoRA as it was originally proposed. Weights W are frozen while A and B are trained.

Recent studies described a rank diminishing phenomenon happening across different matrix structures (Weights, Gradients, Embeddings) [3], [6], [5] used during the training process. These observations suggest the possibility of performing pre-training efficiently in lower dimensional spaces. ReLoRA [7] periodically absorbs LoRA weights to main weight to gradually achieve higher ranks. LTE [2] trains multiple LoRA heads in parallel with micro-batches and periodically merges weights. GaLore [11] exploits the low-rank property of weight gradients by performing the optimization phase in a compact space.

Our first and simplest idea was to take the original LoRA architecture and use methods that we have learned in class to implement our own version of implementing LoRA training in parallel. However, from literature survery we found that LTE already suggested a parallel training method for LoRA and using distrubuted data parallel is also trivial in Pytorch. We changed direction to compare the different related works of LTE and GaLore to evaluate performance and their different use cases.

Our contribution is as follows. We applied the projected gradient update method from GaLore to vision and language tasks, and parallel LoRA heads training method from LTE on language tasks. We experimented with the variations suggested in the e two papers and different hyperparameter settings. We measured the test loss, itera-

tion time and memory efficiency after using the methods from the papers.

Our results show that training low-rank matrices takes less memory, takes less or same training time, but can perform similar or better than training all parameters. A vision transformer (ViTs) with LoRA performs much better with only 0.18 - 2.7% trainable parameters compared to full parameter fine-tuning. In ViT+LoRA fine-tuning, the model also achieved higher accuracy in less number of iterations. When training with LTE, we see a GPT-2 model with 4 LoRA heads and rank 4 reduces trainable parameters by 24%, and can perform close to or slightly better than full-parameter training. Replacing attention layers in transformers with LoRA was found to be most effective. We also saw that using higher rank, such as rank 64, can degrade performance.

In future work, we hope developers explore use cases where this work is valuable in practical applications, such as medical image analysis. For example, developers could make a proof of concept for simple medical image analysis, or FedLoRA that combines LoRA with Federated Learning, which could be very useful for hospitals to preserve data privacy.

2 Related Work

The original proposed LoRA [1] is in the field of Parameter-Efficient Fine-Tuning (PEFT) for the purposes of an alternative to full fine-tuning. Lora-The-Explorer (LTE) [2], uses parallel LoRA heads for training from scratch. GaLore [11] utilizes the low-rank updates to gradients and updates the weights in a smaller subspace. We discuss the methods of LTE and GaLore in details in section 3.

Federated LoRA utilizes the distributed nature of Federated Learning to provide a LoRA at each individual worker rather than a copy of the full model [10] [9]. This allows individual workers to not need high end training clusters, and also allows them to keep their data private. The only thing they need to provide to the server is the gradient updates, not the data itself. This helps improve privacy and would especially be useful to the healthcare industry.

LoraRetriever [12] is similar to the INFaaS paper, where individual users have different desires for what they want to use the foundation model for, so different LoRAs are trained and provided depending on the user prompts.

[8] is an example of a use case for LoRA in finance models. [13] is an example of a use case for LoRA in medical image analysis models.

3 Methods

3.1 LTE (low rank weights)

LTE [2] showed that during pre-training the effective-rank of the model parameters keeps increasing. They also show that choosing the rank equal to smaller dimension of weight matrix (called full-rank training) achieves performance close to full parameter training. However, training with smaller rank adapters fails to achieve performance of full-parameter training. Their key observation was that a matrix can be recreated by addition of lower-dimensional matrices. Therefore, we can train multiple low-rank matrices to reparametarize the higher-rank LoRA matrices to save memory without compromising performance. This technique is named as multi-head LoRA. We can approximate this addition by using lower dimensional matrices of same-rank. If there are N low-rank heads, then the layer output $h(x)$ will be

$$h_{mhlora}(x) = Wx + \frac{s}{N}\sum_{n=1}^{N} B_n A_n x$$

The authors suggested two methods for training multi-head LoRA. A simple way is during forward pass, multiplying each LoRA head with an input batch and adding them up to get the layer output. This is called sequential method. However, this makes the LoRA heads learn identical information and does not fully utilize multiplicity. The second method is to split the input into micro-batches equal to the number of LoRA heads. Then, each micro-batch is multiplied with a LoRA head and the results are concatenated to make the layer output. This is named as parallel method.

ReLoRA [7] showed that merging the product of LoRA matrices to main weight helps achieve higher-rank during training. LTE used delayed-merge technique where instead of merging at every iteration, the LoRA heads are trained for some iterations separately and then the average of the products are accumulated to main weight periodically. In a distributed setup, this technique requires less communication among the nodes. Moreover, LTE showed that instead of reinitializing the LoRA matrices after merge, we can keep the same LoRA head weights and use an offset to balance it. This saves iterations to regain good weights in the LoRA heads. The authors name this method as reset-less version. The parallel training combined with periodic merging is shown in Figure 2.

3.2 GaLore (low rank gradients)

We explored GaLore for low-rank gradient updates in pretraining. More specifically, we measured the low-rank phenomena happening across various deep learn-

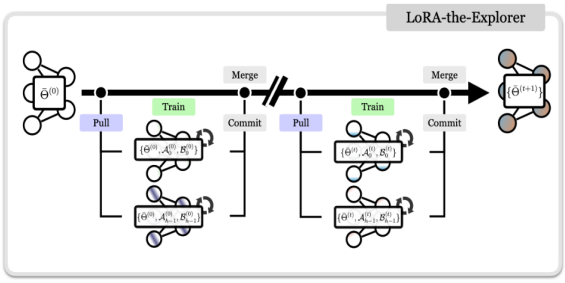


Figure 2: LTE reparametarizes a model weight matrix with multiple LoRA heads and train them independently for T iterations. So, the update to a LoRA head is $\delta_{lora_n}(x) = -\eta\Sigma_t\nabla lora_n(x[t])$. After that individual LoRA updates are accumulated by averaging the heads $\Delta_{lora}(x) = \frac{1}{N}\Sigma_n\delta_{lora_n}(x)$. The update is applied to the main weights.

ing models and confirmed we can leverage on the idea of using approximate low rank factorizations to improve efficiency.

Most of these GaLore explanations were taken straight from the paper. Thorough explanations and proofs for more details can be found in that paper. We took the highlights and put them here for convenience. Figure 3 shows GaLore training mechanism.

3.2.1 Formally defining an optimizer, such as Adam

The regular pre-training weight update can be written down as follows (η is the learning rate):

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t = W_0 + \eta \sum_{t=0}^{T-1} \rho_t(G_t)$$

where $\tilde{G}_t$ is the final processed gradient to be added to the weight matrix, and ρ_t is an entry-wise stateful gradient regularizer (e.g., Adam). The state of ρ_t can be memory-intensive.

3.2.2 Difference between GaLore and other optimizers such as Adam

Gradient low-rank projection (GaLore) denotes the following gradient update rules (η is the learning rate):

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t$$

$$\tilde{G}_t = P_t\rho_t(P_t^\top G_t Q_t)Q_t^\top$$

where $P_t \in \mathbb{R}^{m\times r}$ and $Q_t \in \mathbb{R}^{r\times n}$ are projection matrices.

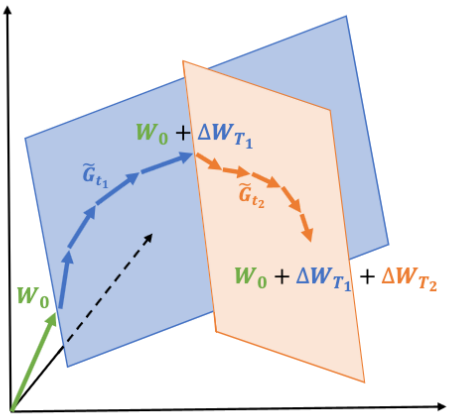


Figure 3: Quoted from GaLore authors [11]: "Learning through low-rank subspaces ΔW_{T_1} and ΔW_{T_2} using GaLore. For $t_1 \in [0, T_1 - 1]$, W are updated by projected gradients G_{t_1} in a subspace determined by fixed P_{t_1} and Q_{t_1}. After T_1 steps, the subspace is changed by recomputing P_{t_2} and Q_{t_2} for $t_2 \in [T_1, T_2 - 1]$, and the process repeats until convergence."

3.3 Difference between GaLore and LoRA

Different from LoRA, GaLore explicitly utilizes the low-rank updates instead of introducing additional low-rank adaptors and hence does not alter the training dynamics of optimizers such as Adam. To verify claim that GaLore does not alter training dynamics, see the original paper for proofs, as well as Figure 4 for experimental evidence.

While both GaLore and LoRA have "low-rank" in their names, they follow very different training trajectories. For example, when $r = min(m, n)$, GaLore with $\rho_t \equiv 1$ follows the exact training trajectory of the original model, as $\tilde{G}_t = P_t P_t^\top G_t Q_t Q_t^\top = G_t$. On the other hand, when BA reaches full rank, optimizing B and A simultaneously follows very different training trajectory from the original model.

4 Experiments and Results

For running experiments, we had access to 4 CloudLab GPUs. We each did our own experiments on each individually, or coordinated multiple at once for distributed data parallel training. All experiments performed on NVIDIA P100 GPUs, each with 12 GB of memory.

Before analyzing the effect of the low rank training systems, we would like to point out a small distinction. Another effective way to reduce memory load on a GPU is by reducing the batch size for the training data. However, the model's trainable parameters also take up a significant amount of memory by themselves. The model

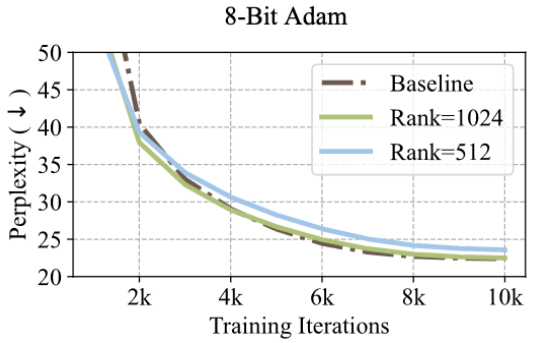


Figure 4: This graph from the GaLore paper shows the comparable performance of GaLore to popular optimizers like Adam. It is important to note that GaLore is an efficiency strategy added on top of existing optimizers like Adam, so this plot is the performance of GaLore on 8-bit Adam specifically.

parameter and gradient chunks of memory is where we will be seeing the savings with the methods we are evaluating.

4.1 LTE

We trained a GPT-2 model [4] with multi-head LoRA architecture from scratch on CNN Dailymail dataset. The model used 512 context length and has about 124 million parameters. We used weight-tying between the token embedding and final linear layer.

For Figure 5 and Figure 6, we replaced only the attention layers with multi-head LoRA layers. Figure 5 shows that parallel-merge learns better than sequential approach in same number of iterations. It is also evident that in parallel-merge, multiple heads performs better than single-head. Interestingly, in parallel-merge, the model with rank 4 performs slightly better than the model with rank 64.

Figure 6 compares the training speed and number of trainable parameters in sequential and parallel-merge method. It shows that sequential multi-head takes 25% more time than parallel multi-head method. However, parallel multi-head training with rank 1 and 4 take about the same time as full parameter training, but with 24% less trainable parameters.

Table 1 shows how the model performs at different LTE hyperparameter settings. Here, we also replaced only the attention layers with LTE layers. We trained the models with batch size 8 and LTE parallel-merge method for 500 iterations. The results show that models with 4 heads perform better than models with 2 heads. However, increasing heads to 8 does not improve performance and can have higher loss. Increasing the merge interval

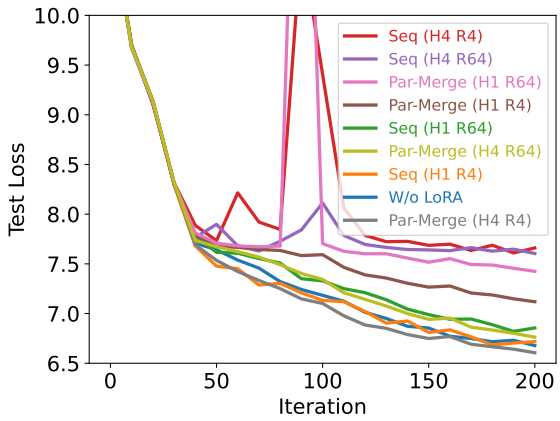


Figure 5: Test loss in sequential and parallel-merge approach of multi-head LoRA

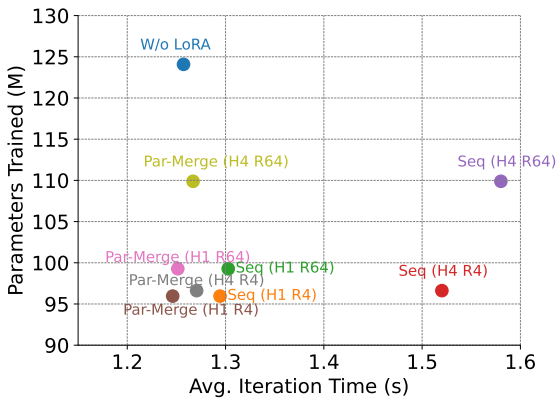


Figure 6: Parameters trained vs iteration time for multi-head LoRA

Table 1: Performance with different number of heads, ranks and merge steps

	Heads	Rank	Merge	Test Loss	Train Time (s)
Heads and Ranks	2	1	10	6.35	788
	2	4	10	6.32	789
	2	16	10	6.33	789
	2	64	10	6.52	790
	4	1	10	6.23	793
	4	4	10	6.23	798
	4	16	10	6.27	794
	4	64	10	6.28	795
	8	1	10	6.25	798
	8	4	10	6.23	809
	8	16	10	6.24	801
	8	64	10	6.39	805
Merge	4	8	10	6.23	795
	4	8	20	6.25	794
	4	8	30	6.29	794
	4	8	40	6.33	793

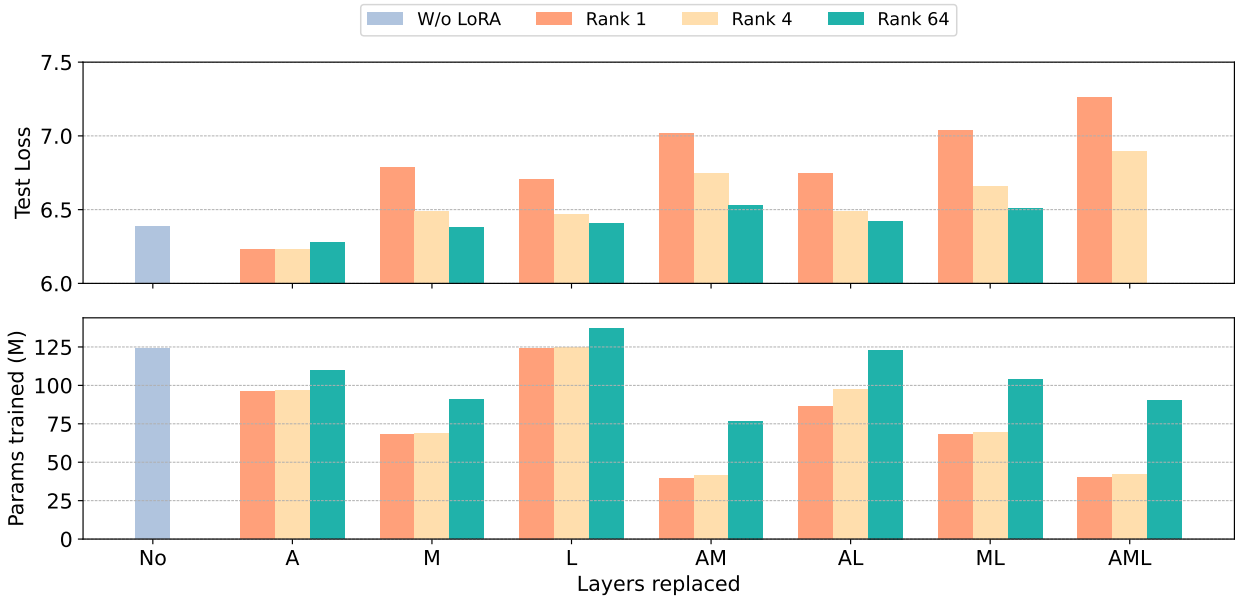


Figure 7: Test loss and number of trainable parameters when different layers are replaced by multi-head LoRA layer. (A = Attention, M = MLP, L = Logit)

reduces training time, but also deteriorates performance by a small amount.

Figure 7 shows the test loss and number of parameters trained when different combinations of layers are replaced by multi-head LoRA layers. We used 4 heads in parallel-merge method and trained GPT-2 model for 500 iterations. Replacing attention layer with LoRA is most effective while replacing logit layer is least effective. Replacing MLP layers in decoder blocks reduces parameters most. However, it also degrades performance. As expected, replacing multiple types of layers have higher test loss compared to replacing single type of layer. Replacing logit layer with multi-head LoRA increased number of parameters because we used weight tying between embedding and logit layer in GPT-2 model.

4.2 GaLore

We experimented training a LLaMA 1B language model using the C4 common crawl dataset for a small number of epochs and noticed a 30% decrease in memory footprint compared with the full training approach (see Figure 8). Since the model hidden size used is about 2K, we employed a GaLore rank of 1024.

Additionally, we found that varying the GaLore rank has little effect on accuracy or memory load, at least in our experiments with this scale of model sizes. We are not reporting tables, because there was simply not enough change in results for it to be an interesting figure. Everything would be identical. It is enough to say

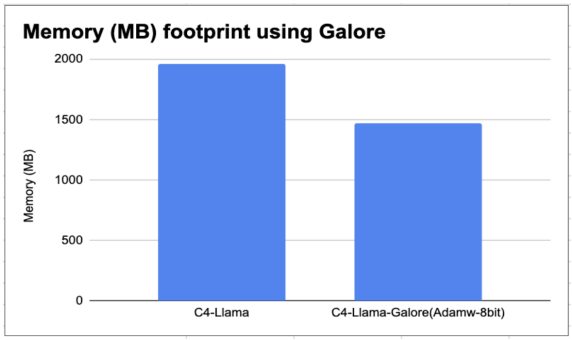


Figure 8: Comparing memory footprint of using Adam optimizer vs GaLore optimizer

that this further confirms the claims from GaLore authors that training dynamics should not be affected, other than some memory load savings (example, see Figure 4).

4.3 LoRA in practical vision tasks

For the sake of exploring more domains than just NLP, we also evaluated the generality of low rank techniques across different domains by also exploring utility in vision tasks. A practical use case of using low rank adaptations to fine tune a model is in the medical field using foundation models trained on natural images and adapting them to medical images for computer aided diagnosis. Therefore, our vision experiments provide evidence for the practical importance of these low rank systems.

LoRA Rank	Trainable Params	Total Params	Memory Load	Accuracy	Training Time	Training Speed
4	0.150M	81.974M	6079MiB	0.7692200557	~23:30 min	1.01s/it
8	0.290M	82.114M	6085MiB	0.9135097493	~23:30 min	1.01s/it
16	0.571M	82.395M	6105MiB	0.8956824513	~23:30 min	1.01s/it
32	1.134M	82.958M	6087MiB	0.8862116992	~23:30 min	1.01s/it
64	2.259M	84.083M	6185MiB	0.8440111421	~23:30 min	1.01s/it
Full Fine Tuning	82.557M	82.557M	9225MiB	0.5603064067	~32:00 min	1.27s/it

Figure 9: Evaluating vit_base_patch16_224 using regular LoRA at differing ranks compared to full fine tuning for **only 1 epoch**. For the sake of the main conclusions that we want to present (and time it would take to run and record experimental data), we are only presenting results for the base size.

We did reach our end goal to implement both LTE and GaLore for vision, but ran out of time to properly evaluate their memory efficiency and performance.

We did some experiments with GaLore, but they proved to be uninteresting for the most part (no new results compared to NLP). We compared the memory loads of using the Adam optimizer with and without GaLore. The amount of savings was proportionally similar to the NLP experiment that produced Figure 8. Just like in NLP tasks, we found that varying the GaLore rank has little effect on accuracy or memory load, at least in our experiments with this scale of model sizes.

Most of our experiments evaluated the memory savings of simply using regular LoRA for fine tuning, testing the effects of varying the ranks and such. GaLore-AdamW was used as the optimizer. No evaluation was done varying LTE parameters, as the original LTE paper went quite in depth evaluating ViTs on their own.

The dataset used is the 3x224x224 image size PathMNIST dataset from the larger MedMNIST dataset. This has been established as a benchmark for medical foundation models, especially with the 224 image size being a larger and newer addition.

The two models tested were vit_base_patch16_224 and vit_large_patch16_224, as they were the largest pretrained models that we could find and use easily.

We compared the memory load of varying LoRA rank (see Figure 9). Varying the rank of regular LoRA had only small but noticeable affects of changing the memory load. All ranks, however, were significantly lighter on memory load and converged faster than full fine tuning. The base size and large size ViTs had the same memory saving trends, with large size just scaled up to about 11900 MB rather than 6100MB. In Figures 9 and 10, we can see that LoRA converges significantly faster with much lower memory load compared to full fine tuning. We believe that this is because the smaller parameter count is just a compression of the larger full model, so it still includes all of the details, yet does not suffer the curse of dimensionality of tuning exponentially more parameters. FFT would catch up eventually, but will take significantly longer than LoRA.

5 Conclusions

We explored the utility of multiple low rank methods for training. We then applied these methods across task domains (vision and language). Our primary motivation was to reduce memory costs, as these low rank methods are a form of compression. We hoped to learn which methods provide the most savings on memory load. Additionally, adapting methods to new domains proves generality and utility of these methods.

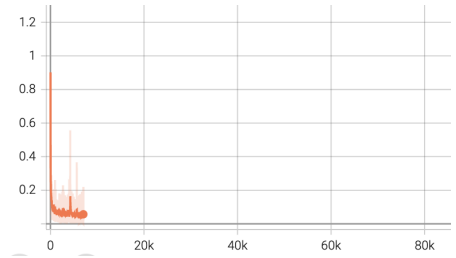
5.1 LTE

Parallel training of multiple LoRA heads with periodic merge improves training performance and with right hyperparameter settings can surpass full parameter training. Because of its parallel nature, the training time is close to full parameter training. While replacing all possible layers can reduce parameters greatly, it also hurts performance. Our results show that replacing only attention layer is a good balance between training accuracy and memory savings. Replacing attention layer in GPT-2 architecture reduces number of trainable parameters by 24%.

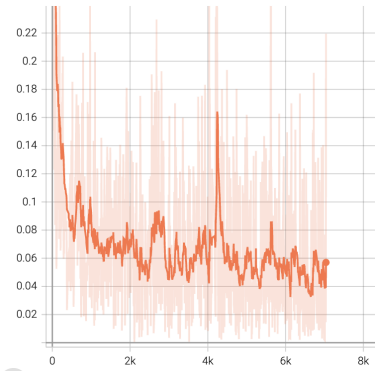
5.2 GaLore

GaLore's authors claim running on a "consumer-grade" (still top tier and expensive) RTX 4090 GPU with 25GB of memory, but we dispute GaLore's effectiveness across scale. In our experience, we trained with 12GB of memory and found that GaLore's benefits were minimal for the ViTs because of their lower param count ($\sim$82M for base, $\sim$220M for large, $\sim$630M for huge) compared to models like Llama (1B params, and goes bigger). GaLore is only substantial for models with billions of parameters and GPUs with significant memory already (see Figure 11).

(a) Full Fine Tuning. Batch size: 80; Epochs: 60; Final test acc.: 71.41%



(b) Rank 8 LoRA; Batch size: 128; Epochs: 10; Final test acc.: 94.53%



(c) Rank 8 LoRA, plot scaled up

Figure 10: Above you will see the training loss curves of the Full Fine Tuning (FFT) strategy and regular LoRA using a rank of 8 on ViT-base. (a) and (b) have x and y axis scaled similarly to show direct comparison, while (c) is just (b) but zoomed in to show details. The only difference in hyperparameters are the batch size and number of epochs. We attempted to utilize the max amount of GPU memory for each (both about a 11500MB load). Even though FFT was given more epochs and more time to train, the LoRA adapted model took up less space in memory, so we could use a larger batch size, which led to less iterations (that took longer), LoRA still achieved the higher accuracy at a much quicker convergence time. Remember, LoRA with rank 8 has 0.290M trainable parameters compared to FFT tuning all 82.114M

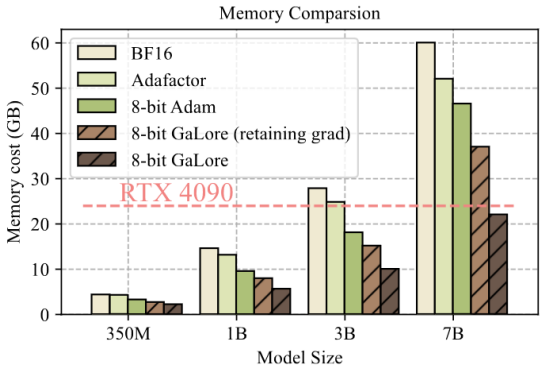


Figure 11: From the GaLore paper, we can conclude that memory savings do exist for small models, but GaLore is only substantial for models with billions of parameters and GPUs with significant memory already.

6 Future Work

We ran out of time to fully evaluate this final goal of combining the complementary benefits of both LTE and GaLore to further improve memory efficiency, but we do have some preliminary versions in our git repo. We hope to pursue these evaluations in future work. We expect that combining both methods will further improve memory efficiency.

We hope to perform deeper studies to derive heuristics for setting hyperparameters (rank, etc) for a given task, whether it be broad performance evaluation of Vision or NLP tasks, or something more specific within those broad fields. We also believe that there can be more studies on the tradeoff between memory, training time, and test loss in LoRA training.

An example application of LoRAs is in hospital systems. Adaptors could be multilayered, with a medical foundation model for different diagnosis tasks that can be adapted off of a regular ViT pre-trained on natural images ImageNet [13]. Federated learning can protect data privacy, maintain adaptability, while still contributing to model performance [10].

Figure 12 is taken from [13], which talks about the practical use of LoRAs on ViTs in use for medical data. We include this chart to show that increasing model size does increase performance, so this does provide motivation for trying to bring down memory loads. Hospitals can save money on hardware if they don't need to use as expensive hardware. We hope to test GaLore on larger ViT models, as we expect GaLore to be more effective at this scale of 1.75B parameters for example.

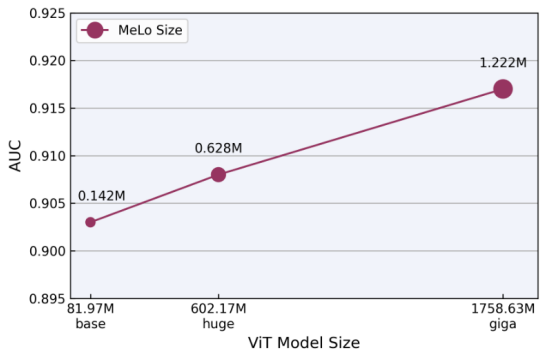


Figure 12: Results from MeLo paper[13]: "The AUC gradually increases as the ViT model size expands while the trainable parameters of corresponding MeLo modules remain consistently low."

7 Contributions

We all discussed together what the main thesis, focus, and results of our findings are, but we each had distinct focuses.

- Ivan mainly focused on GaLore and getting it working in the NLP domain. He also made a really nice template design for the poster.
- Laik mainly focused on GaLore and getting it working in the Vision domain. He also picked out the majority of graphics and organization for the poster.
- Sadman mainly focused on LTE and getting it working with nanoGPT. Sadman also did the majority of the Cloudlab setup.
- Zheyang mainly focused on LTE and getting it working with nanoGPT. He did the setup for the overleaf LaTeXdocuments.

References

[1] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, W. Chen, and T.-Y. Liu. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[2] M. Huh, B. Cheung, J. Bernstein, P. Isola, and P. Agrawal. Training neural networks from scratch with parallel low-rank adapters. *arXiv preprint arXiv:2402.16828*, 2024.

[3] M. Huh, H. Mobahi, R. Zhang, B. Cheung, P. Agrawal, and P. Isola. The low-rank simplicity bias in deep networks. *Transactions on Machine Learning Research*, 2023.

[4] A. Karpathy. nanogpt: Minimal gpt implementation for educational purposes. https://github.com/karpathy/nanoGPT, 2023.

[5] T. Le and S. Jegelka. Training invariances and the low-rank phenomenon: beyond linear networks. In *International Conference on Learning Representations*, 2022.

[6] T. Li, L. Tan, Z. Huang, Q. Tao, Y. Liu, and X. Huang. Low dimensional trajectory hypothesis is true: Dnns can be trained in tiny subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3411–3420, 2023.

[7] 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.

[8] X.-Y. Liu, J. Zhang, G. Wang, W. Tong, and A. Walid. Fingpt-hpc: Efficient pretraining and finetuning large language models for financial applications with high-performance computing. *arXiv preprint arXiv:2402.13533*, 2024.

[9] N. Rieke. What is federated learning? NVIDIA Blog, October 2019.

[10] L. Yi, H. Yu, G. Wang, and X. Liu. Fedlora: Model-heterogeneous personalized federated learning with lora tuning. *arXiv preprint arXiv:2310.13283*, 2023.

[11] J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024.

[12] Z. Zhao, L. Gan, G. Wang, W. Zhou, H. Yang, K. Kuang, and F. Wu. Loraretriever: Input-aware lora retrieval and composition for mixed tasks in the wild. *arXiv preprint arXiv:2402.09997*, 2024.

[13] Y. Zhu, Z. Shen, Z. Zhao, S. Wang, X. Wang, X. Zhao, D. Shen, and Q. Wang. Melo: Low-rank adaptation is better than fine-tuning for medical image diagnosis. *arXiv preprint arXiv:2311.08236*, 2023.