Approximations may be all you need: Towards Pre-training LLMs with Low-Rank Decomposition and Optimizers

Namrata Shivagunde[1,2] * zshivagu@amazon.com
Mayank Kulkarni[1] maykul@amazon.com
Giannis Karamanolakis[1] karamai@amazon.com
Jack FitzGerald[1] jgmf@amazon.com
Yannick Versley[1] yversley@amazon.com
Saleh Soltan[1] ssoltan@amazon.com
Volkan Cevher[1,3] volkcevh@amazon.com
Jianhua Lu[1] jianhual@amazon.com
Anna Rumshisky[1,2] arrumshi@amazon.com

[1]Amazon AGI [2]University of Massachusetts Lowell [3]EPFL (LIONS)

Abstract

Large language models (LLMs) have achieved remarkable performance on various natural language processing tasks, but training LLMs at scale is extremely resource-intensive, requiring substantial computational power, memory, and energy consumption. This has motivated research into efficient training methods, particularly during the pre-training phase. There are two main approaches to approximate full-rank training which have emerged to address this challenge: low-rank model decomposition (e.g., ReLoRA) and memory-efficient optimizers (e.g., GaLore). In this work, we systematically evaluate both lines of research on a range of metrics, including validation perplexity, memory usage and throughput. Additionally, we propose improvements on both low-rank decomposition methods, by improving the low-rank matrices decomposition and initialization, and memory efficient optimizer methods via the introduction of error feedback and dynamic update steps. Our comprehensive evaluation under the same experimental setting shows that our proposed optimizations outperform all previous methods, achieving almost same throughput as full-rank training, saving 9% in memory traded off for a 1.5% increase in validation perplexity.

1 Introduction

Large language models (LLMs) have achieved remarkable performance on a wide range of natural language processing tasks. However, training large-scale LLMs is extremely resource-intensive and expensive, requiring substantial computational power, memory, and energy consumption. As a result, there is a pressing need for efficient training methods that can reduce the computational burden while maintaining model performance. One promising line of research focuses on efficient fine-tuning techniques [Hu et al., 2022, Liu et al., 2024, Meng et al., 2024, Lialin et al., 2023, Xu et al., 2023, Han et al., 2024b], which have demonstrated significant gains in training efficiency. However, a question remains: can similar efficiency be realized during the *pre-training* phase of LLMs?

*Work done during an internship at Amazon. This paper represents the work performed by all authors at Amazon. Correspondence to namratashivagunde@gmail.com

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

To address this question, two main approaches to approximate full-rank training have emerged, namely low-rank decomposition and memory-efficient optimizers. The first approach involves decomposing the LLM's weights into low-rank matrices and training these low-rank matrices. Methods like ReLoRA [Lialin et al., 2024] have shown promising results in this direction, requiring fewer trainable parameters and better throughput during pre-training. The second approach aims to improve the optimization algorithms such as GaLore [Zhao et al., 2024] which project the gradient into low-rank and reduce the memory requirements for optimizer states but has reduced throughput than full-rank training.

In this work, we aim to systematically evaluate both lines of research across a wide range of practical dimensions, namely validation loss, memory usage and throughput and propose efficient enhancements to better balance their trade-off. Specifically, we propose three novel low-rank decomposition methods ReDoRA, RePiSSA and ReLoQRa which show improvement over ReLoRA. On the optimizer side, we introduce two novel techniques, GaLore with error feedback (GaLore-error-feedback) and GaLore with dynamic update steps (GaLore-dynamic). We further improve GaLore by combining these two techniques and call the new method GaLore-dynamic-error-feedback.

Our experiments demonstrate that ReLoQRa with reset frequency of 10k and warm start yields the best performance among low-rank decomposition method whereas GaLore-dynamics-error-feedback with weight of 0.2 is the best method overall, having the lowest validation perplexity of 20.66, which is 1.5% less than full-rank training (with AdamW). GaLore-dynamics-error-feedback also has a throughput of 141.4 examples per second, which is almost same as full-rank training while it reduces the memory footprint by 9% when compared against full-rank training.

2 Related work

Low-rank model decomposition methods: Earlier work shows huge potential to pre-train LLMs using low-rank model decomposition methods [Lialin et al., 2024, Jiang et al., 2024, Han et al., 2024a, Loeschcke et al., 2024]. ReLoRA [Lialin et al., 2024] decomposes weight matrix into low-rank LoRA Hu et al. [2022] matrices and uses merge and reinit technique to attain higher rank during training LLMs. MoRA [Jiang et al., 2024] uses a compression and decompression operator on inputs and a single square matrix of higher rank than two low-rank lora matrices to improve over LoRA and introduce ReMoRA which improve over ReLoRA, however it is only compared against ReLoRA and trained for 10k with rank 128 and do not discuss throughput. SLTrain [Han et al., 2024a] parameterize the weights as a sum of low-rank and sparse matrices achieve memory efficiency during pre-training. However, SLTrain throughput is lower than Full-rank training. LoQT [Loeschcke et al., 2024] is an low-rank training method for quantized LLMs. In this method, LoRA matrices are initialized with gradients of the dequantized weight using SVD and quantization error. Except ReLoRA, all of these methods suffer from lower throughput than full-rank training. There are various improved low-rank fine-tuning methods which have not been explored for pre-training. Methods like DoRA [Liu et al., 2024] decomposes the weight into magnitude and direction and applies LoRA only to the direction component. On the other hand, PiSSA [Meng et al., 2024] decomposes the weight matrix into primary and residual component and only trains primary component using LoRA matrices. DoRA and PiSSA is shown to perform better than LoRA for fine-tuning and have not been explored for pre-training LLMs.

Memory-efficient optimizers: Recent work shows how various memory efficient optimizer can help in saving memory during pre-training LLMs [Zhao et al., 2024, Zhang et al., 2024, Modoranu et al., 2024, Muhamed et al., 2024]. GaLore [Zhao et al., 2024] projects the gradient into a low-rank subspace, updates it using Adam and projected back to the original space, thereby using less memory for optimizer states. However, it suffers from lower throughput than Full-rank training. On the other hand, Adam-mini [Zhang et al., 2024] is a less adaptive version of AdamW, which saves on memory by adopting a single learning rate for blocks of parameters in attention and MLP layers, while giving comparable performance to AdamW. Adam-mini has been implemented and compared against AdamW for float32 and it is still not known how well it performs for reduced precision like bfloat16. MicroAdam [Modoranu et al., 2024] compresses the gradients via TopK sparsification and uses quantized error feedback to correct for the gradient but it is used for fine-tuning only. Grass [Muhamed et al., 2024] uses product of selection and scaling matrix based on row-norm of gradient as projection matrix. On 350M scale, Grass achieves memory saving and higher throughput however

have a higher validation perplexity than GaLore. Grass mentions throughput for rank 64 whereas validation perplexity is for higher ranks (e.g. 128, 256). Most of the optimizers are compared only against other optimizers and lacks analysis on how they compare against other efficient pre-training techniques under same experiment set-up.

In our study, we compare and advances both low-rank model decomposition techniques and optimizers to attain efficient pre-training of LLMs.

3 Background

We provide background on the methods we are evaluating in this study, namely full-rank training, low-rank decomposition (ReLoRA, DoRA, PiSSA), and memory-efficient optimizers (GaLore, Adam-mini).

Full-rank training: Following previous work [Lialin et al., 2024, Zhao et al., 2024], in full-rank training, we train the entire weight matrix using AdamW as optimizer.

ReLoRA: ReLoRA [Lialin et al., 2024] approximates the weight $W \in \mathbb{R}^{m \times n}$ matrix by LoRA matrices $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$ where $r \ll \min(m, n)$. During training, the original weights W are kept frozen, and A and B are trained. The LoRA matrices are merged back to the W after every reset step T. This brings the total updates to the W to the given ΔW.

$$\Delta W = \sum_{t=0}^{T_1} \delta W_t + \sum_{t=T_1}^{T_2} \delta W_t + \cdots + \sum_{t=T_{N-1}}^{T_N} \delta W_t = sW_A^1 W_B^1 + sW_A^2 W_B^2 + \cdots + sW_A^N W_B^N \quad (1)$$

DoRA: In DoRA [Liu et al., 2024], the weight matrix $W \in \mathbb{R}^{m \times n}$ is decomposed into a magnitude vector $m \in \mathbb{R}^{1 \times m}$ and a directional matrix $V \in \mathbb{R}^{m \times n}$ as follows:

$$W = m \frac{V + \Delta V}{\|V + \Delta V\|_c} = m \frac{W_0 + BA}{\|W_0 + BA\|_c} \quad (2)$$

where $\|.\|_c$ represents the vector-wise norm across each column. W_0 is kept frozen and m and directional component LoRA matrices A and B are trained.

PiSSA: PiSSA [Meng et al., 2024] decomposes the weight matrix $W \in \mathbb{R}^{m \times n}$ into principal and residual singular values and vectors. The principal singular values and vectors are used to initialize the low-rank matrices $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$, which are trained and the residual singular values and vectors are used to construct a residual matrix W_{res}, which remains frozen during training. The weight matrix W can be stated as:

$$W = W_{\text{res}} + B \quad (3)$$

GaLore: GaLore [Zhao et al., 2024] is an optimizer where the gradients G_t are projected into a subspace using a projection matrix P_t which is derived from the SVD decomposition of the gradient, updated using AdamW (ρ_t) and projected back to the original space using transpose of the same projection matrix P_t, after every update frequency T steps. GaLore has the following gradient update rules where η is the learning rate:

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \tilde{G}_t, \quad \tilde{G}_t = P_t \rho_t(P_t^\top G_t Q_t) Q_t^\top \quad (4)$$

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

Adam-mini: Adam-mini [Zhang et al., 2024] reduces the memory footprint of the Adam optimizer by partitioning the model parameters into blocks and using a single learning rate for each block. Adam-mini partitions all Queries and Keys by heads and uses the default PyTorch partition for the rest of the model. For each parameter block outside the embedding layers, a single learning rate is

used. To compute this learning rate, Adam-mini uses the square root of the mean of the second-order momentum values within that block. Adam-mini results were shown for float32 [Zhang et al., 2024], we show the results for bfloat16.

All of the low-rank methods are applied on attention and MLP layers and other layers are trained using AdamW.

4 Method

We introduce three novel low-rank model decomposition methods, ReDoRA, RePiSSA and ReLoQRa and two techniques, dynamic projection matrix update frequency and error feedback to improve GaLore.

4.1 Low-rank model decomposition methods

We propose a new method, namely ReLoQRa, where we initialize LoRA matrices A and B with the QR decomposition of the current weight. We use this technique as this allows better initialization of LoRA matrices than ReLoRA and allow exploration of orthogonal spaces. We provide more detail in next section.

Earlier work [Liu et al., 2024, Meng et al., 2024] show that DoRA and PiSSA are better than LoRA for fine-tuning. We investigate when these methods are combined with re-initialization and merge technique similar to ReLoRA, how much improvement we can achieve for pre-training. We call these methods ReDoRA and RePiSSA.

ReLoQRa: We observe in Figure 1 that matrix norm of memory efficient optimizer GaLore increases gradually whereas the low-rank model decomposition method ReLoRA suffers from huge drop after each reset step. This is due to re-initialization of LoRA matrices i.e. A with kaiming initialization and B with zeros. We hypothesize that re-initializing LoRA matrices with this strategy doesn't allow ReLoRA to explore sub spaces optimally. Note that completely removing the LoRA matrices resets is equivalent to LoRA and it perform worse than ReLoRA [Lialin et al., 2024].

Therefore, we introduce ReLoQRa where we initialize LoRA matrices A and B with QR decomposition of the weight. This allows to have a better starting state for A and B after each reset and allows to explore orthogonal spaces. This results in a much higher matrix norm, likely resulting in better exploration of spaces through larger steps and thus potentially finding better local minima. Our result shows that ReLoQRa perform better than ReLoRA.

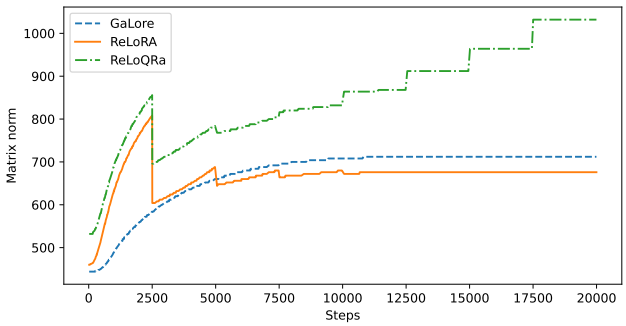

Figure 1: Matrix norm of LLM training methods.

In ReLoQRa, we initialize the LoRA matrices A and B using QR decomposition of the current weight matrix W. Specifically, we decompose W $\in \mathbb{R}^{m\times n}$ into an orthogonal matrix $Q_t \in \mathbb{R}^{m\times n}$ and an upper triangular matrix $R_t \in \mathbb{R}^{n\times n}$. We pick the top "r" rank components from Q and R to initialize LoRA matrices and train them keeping the original weight frozen. Similar to ReLoRA, the LoRA matrices are merged back to original matrix after T steps and reinitialized using ReLoQRa technique.

$$W = W + Q[:, :r]R[r:, :] \tag{5}$$

ReDoRA: ReDoRA is a variant of the DoRA method, where it incorporates the merging and re-initialization strategy from the ReLoRA approach. During the re-initialization step, ReDoRA takes the current weight matrix W and decomposes it using the DoRA method to initialize the magnitude parameter m and the low-rank matrices A and B of the direction component. The LoRA matrices are trained and merged back to W similar to ReLoRA.

RePiSSA: RePiSSA uses the PiSSA matrix decomposition technique on Weight matrix and use the principal component to initialize LoRA matrices keeping residual component frozen. Similar to ReDoRA, we incorporates the ReLoRA style merging and re-initialization strategy to use PiSSA for pre-training and call this method as RePiSSA.

4.2 GaLore improvement methods

GaLore maintains momentum for only the top SVD subspace of the gradients [Zhao et al., 2024]. The residual gradient still occupies memory and is not used. Inspired by [Vyas et al., 2024], we use this residual gradient as error feedback to the gradient for next step. We call this method GaLore-error-feedback.

Following Loeschcke et al. [2024], we hypothesize that GaLore need not learn all sub-spaces equally. We observe that it learns more spaces at the start of the training as compared to the end of the training. We also note that GaLore uses SVD operations which may contribute towards the lower throughput compared to the baseline [Zhao et al., 2024, Muhamed et al., 2024]. To address these points, we introduce Galore-dynamics, where we use a dynamically changing projection matrix update frequency in GaLore. This method reduces SVD operations in GaLore from 400 to 15.

GaLore-Error-Feedback. As GaLore uses Adam on top "r" gradient components, we use the residual gradient as a feedback to the gradient in the next iteration. The residual gradient is computed similar to Vyas et al. [2024]. R_t is the low-rank gradient and S_t is the residual gradient which is added to the gradient in the next iteration.

$$S_t = G_t - P_t * R_t, \quad R_t = Project(G_t) \tag{6}$$

$$G_{t+1} = G_{t+1} + S_t \tag{7}$$

GaLore-dynamics. Loeschcke et al. [2024] work with a quantized model and use exponential increase in the update frequency. Instead, we start from 200th step and follow a simpler update rule update_step = update_steps + 0.5 * update_steps, the maximum update frequency is set to 2500 Loeschcke et al. [2024]. For our update rule, starting earlier than 200 steps led to worse performance than GaLore. The dynamic update frequency allows to have faster updates in the initial part of the training and slower updates as training progresses.

GaLore-dynamics-error-feedback. To combine the benefits of both dynamics steps and error feedback, we combine GaLore-dynamics with error-feedback and call this method GaLore-dynamics-error-feedback.

5 Experiment setup

Following [Lialin et al., 2024] experiment set up, we use a 350M parameter LLaMA-based architecture model which uses RMSNorm and SwiGLU activations [Zhang and Sennrich, 2019, Shazeer, 2020, Touvron et al., 2023]. We train the model on C4 (only en) for 4.5B tokens across all methods. For each method, we use a total batch size of 1152, micro-batch size of 48, a rank of 256, maximum sequence length of 256 and train for 20k iterations. We keep the hyperparameters same across all methods other than learning rate. The learning rate is set to 1.5e-3 for all low-rank decomposition methods and optimizers, except for the baseline AdamW and Adam-mini, which use 5e-4. This choice aligns with the recommendations from [Lialin et al., 2024] and [Zhang et al., 2024]. Additionally, we include baselines with float32 and bfloat16 precision. For computing memory usage, we follow [Muhamed et al., 2024] methodology. All experiments are conducted on 8 V100 GPUs. Note that V100 GPU does not provide native bf16 instructions and the throughput and memory requirement for the methods may vary when executed on GPUs with native bf16 support such as A100 or H100.

For ReLoRA-style methods, we employ a reset frequency of 2500 and warmup steps of 100 as recommended by [Lialin et al., 2024]. In the warm-start runs, we initially train the full-rank model with AdamW until 5k steps, and then switch to low-rank decomposition methods for the remaining steps from 5k to 20k.

For the GaLore optimizer, we explore learning rates of 1e-2, 5e-3, and 1.5e-3, projection weight update frequencies of 200, 100, and 50, and GaLore scales of 0.25, 0.5, and 1. We select the configuration with showed best results i.e. learning rate of 1.5e-3, projection matrix update frequency of 50, and a GaLore scale of 1. We keep these GaLore hyperparameters same across all experiments which are conducted using GaLore which includes GaLore-error-feedback, GaLore-dynamics and GaLore-dynamics-error-feedback.

Galore-error-feedback and GaLore-dynamics-error-feedback have an additional error-feedback weight hyperparameter. We show results for error-feedback weights of both 0.1 and 0.2 as we found 0.1 works better for Galore-error-feedback and 0.2 for GaLore-dynamics-error-feedback.

6 Results

Table 1 shows validation perplexity, memory usage and throughput for baselines and all efficient training methods included in this study.

ReLoQRa is best among low-rank model decomposition methods. Among the low-rank model decomposition methods, we found ReLoQRa to be better than ReLoRA, RePiSSA and ReDoRA by a perplexity of 2.16, 1.17 and 0.55 respectively. We observed that the reset frequency of the LoRA matrices was set for ReLoRA initialization technique. As we observed different reset pattern for ReLoQRa in Figure 1, we increased the reset frequency from 2.5k to 10k, this not only improved the throughput but also the perplexity improved from 25.43 to 23.50.

ReLoRA originally uses warm start to obtain optimal performance, hence we also experiment with warm-start with our best method of ReLoQRa with reset frequency 10k. We demonstrate a 0.19 perplexity improvement when compared to using ReLoRA with similar warm start and 10k reset frequency. Even though ReLoQRa with reset frequency 10k is better than all low-rank decomposition methods, it still lags behind the baseline Full-rank with AdamW where both weight and optimizer states are in bfloat16, by 0.68.

In terms of memory consumption, all ReLoRA-style methods, including ReLoRA, ReDoRA, RePiSSA, and their variants, exhibit a significantly lower memory footprint compared to the baseline full-rank methods. These low-rank methods require approximately 2.31 GB of memory when considering all parameters and 1.75 GB when considering only trainable parameters (see Table 3 in the Appendix B). Compared to the 3.11 GB required for all parameters in full-rank training.

We found that the throughput for low-rank decomposition methods of 120 examples/second to be lower than baseline which is 141.5 examples/sec. This is contradicts the findings in [Lialin et al., 2024] and we show an analysis on this in the next section. We show training loss and evaluation loss for baselines and all low-rank model decomposition methods in Figures 4 and 5 respectively in Appendix A.

Why ReLoRA style methods have lower throughput than Full-rank training? The key factor contributing to the lower throughput of ReLoRA methods is the additional computation required during the forward pass. For the full-rank AdamW method, the forward pass involves only W_x where W is weight matrix and the x is the input. However, in ReLoRA, the forward pass needs an additional forward pass of ABx, where A and B are the low-rank decomposition matrices. This additional forward pass introduces overhead, leading to a lower throughput for ReLoRA style methods. Table 2 shows that for a micro-batch size of 24 and rank 256, the baseline achieves a throughput of 137.3 examples/sec, while the ReLoRA has a lower throughput of 115.2 examples/sec.

As we decrease the ReLoRA rank from 256 to 128 and 64 keeping the micro-batch same, we see that the throughput increases from 115.2 to 129.0 and 135.2 respectively. The throughput further increases if we double the micro-batch size from 24 to 48. With hidden dimension to rank ratio as 16 and double the micro-batch, we can achieve better throughput than baseline for low-rank model decomposition methods. Lialin et al. [2024] mentions that for 1B scale, with ReLoRA rank as

128, which is 16 times less than full-rank and with a double micro-batch size, we can get higher throughput than full-rank training. This aligns with our analysis that in order to get higher throughput with low-rank model decomposition methods, we need atleast 16 times less rank and double the micro-batch size as compared to full-rank training.

GaLore-dynamics-error-feedback is best among optimizers. We find that GaLore outperforms Adam-mini being lower on perplexity by 0.15 points. We further improve this performance by using a dynamics projection matrix update frequency (GaLore-dynamics) resulting in a 0.75 point improved perplexity. Orthogonal to this, we also found that the Weighted Error Feedback (GaLore-EF) also improved upon GaLore's validation perplexity by a more conservative 0.03. We combined both these methods as GaLore-dynamics-error-feedback and with a weighted error-feedback of 0.2, we were able to achieve the lowest validation perplexity of 20.66, outperforming GaLore and Adam-mini by 0.35 and 0.5. However, while its throughput is comparable to full rank AdamW, it lags behind in perplexity by 0.31.

GaLore-dynamics-error-feedback has a memory footprint of 2.83 GB which is higher than Adam-mini and GaLore by 0.28 and 0.57 GB respectively but lower than Full-rank AdamW by 0.28 GB. GaLore and GaLore-dynamics uses 2.26 GB memory which is the least memory among all optimizers.

GaLore variants surpasses all methods. Among low-rank model decomposition methods, ReLoQRa with reset at 10k and warm-start method achieves the lowest validation perplexity of 20.82 with a memory requirement of 2.31 GB and throughput of 120 examples per sec. Even though ReLoQRa has a lower memory footprint, it has a reduced throughput when compared against full-rank training and requires a warm start. ReLoQRa with reset step 10k with warm start is better than Adam-mini and GaLore by a perplexity of 0.34 and 0.19 respectively, however has a reduced throughput by 14%. GaLore variants GaLore-dynamics, GaLore-error-feedback and GaLore-dynamics-error-feedback surpasses all low-rank model decomposition methods, Adam-mini and vanilla GaLore. GaLore-dynamics-error-feedback with error-feedback weight of 0.2 is found to be the best method overall, with a validation perplexity of 20.66, memory usage of 2.83 GB and throughput of 141.4 examples per second. We show training loss and evaluation loss for baselines and best methods in each of the low-rank model decomposition method and optimizers categories in Figure 2 and 3 respectively. Full training loss curve is shown in Figure 8 in Appendix A.

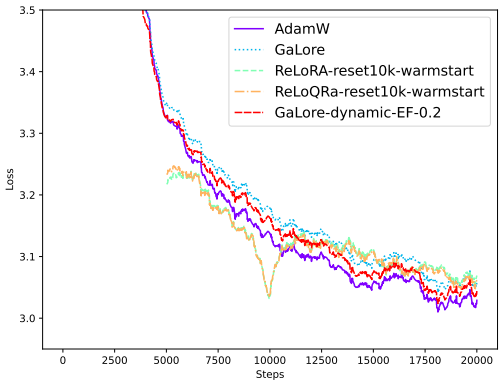

Figure 2: Training loss curves for overall best methods and baselines.

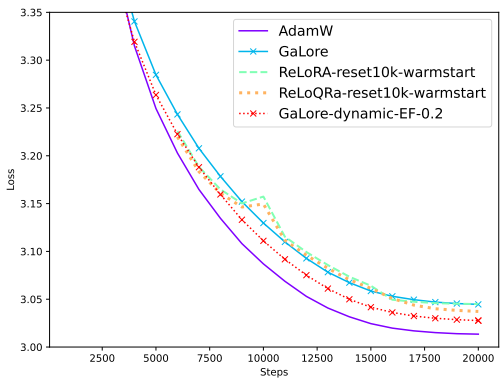

Figure 3: Validation loss curves for overall best methods and baselines.

7 Conclusion

In this study, we conducted a comprehensive evaluation of low-rank model decomposition techniques and memory-efficient optimizers for efficient LLMs pre-training under the same experiment setting. We introduce three novel low-rank model decomposition methods, ReDoRA, RePiSSA and ReLoQRa which improves on the ReLoRA. We also improve the GaLore by using dynamic steps and error feedback. Our results revealed that ReLoQRa with reset at 10k and warm-start emerged as the best-performing low-rank decomposition method, achieving a validation perplexity of 20.82 while

Methods	Perplexity	Memory in GB						Thru.
		Param	Grad	OS	Act	Extra	Total	put
Baselines								
FT + AdamW (w 32, os 32)	17.24	1.37	1.37	2.74	0.58	0.12	6.18	160.5*
FT + AdamW (w 32, os 16)	18.34	1.37	1.37	1.37	0.30	0.12	4.54	156.2*
FT + AdamW (w 16, os 16)†	20.35	0.69	0.69	1.37	0.30	0.06	3.11	141.5
Low-rank model decomposition methods								
ReLoRA†	27.61	0.91	0.35	0.69	0.30	0.06	2.31	120.0
ReDoRA	27.06	0.91	0.35	0.69	0.30	0.06	2.31	118.0
RePiSSA	26.44	0.91	0.35	0.69	0.30	0.06	2.31	120.2
ReLoQRa	25.43	0.91	0.35	0.69	0.30	0.06	2.31	120.5
ReLoQRa with reset at 10k	23.50	0.91	0.35	0.69	0.30	0.06	2.31	122.0
ReLoRA + warm start	21.50	0.91	0.35	0.69	0.30	0.06	2.31	120.0
ReLoRA with reset at 10k + warm start	21.01	0.91	0.35	0.69	0.30	0.06	2.31	121.5
ReLoQRa with reset at 10k + warm start	20.82	0.91	0.35	0.69	0.30	0.06	2.31	120.0
Optimizers								
Adam-mini	21.16	0.69	0.69	0.81	0.30	0.06	2.54	138.0
GaLore†	21.01	0.69	0.69	0.53	0.30	0.06	2.26	140.0
GaLore-dynamics	20.76	0.69	0.69	0.53	0.30	0.06	2.26	138.8
GaLore-EF (0.1)	20.98	0.69	0.69	1.09	0.30	0.06	2.83	139.0
GaLore-EF (0.2)	21.35	0.69	0.69	1.09	0.30	0.06	2.83	139.0
GaLore-dynamics-EF (0.1)	20.86	0.69	0.69	1.09	0.30	0.06	2.83	141.5
GaLore-dynamics-EF (0.2)	**20.66**	0.69	0.69	1.09	0.30	0.06	2.83	141.4

Table 1: Comparison of parameter-efficient pre-training methods. “Param” is parameters, “Grad” is gradients, “OS” is optimizer states, “Act” is activation states, “Extra” is extra memory for largest tensor. * stands for throughput for micro-batch 24, the throughput will be slightly higher (0.04%) than this value for micro-batch size of 48. All results are for weight and optimizer states in bfloat16 unless w and os are mentioned. “w” and “os” stands for weights and optimizer states of the model. Numbers in parentheses “(*)” is the error-feedback (EF) weighting. †Methods highlighted in blue are our evaluation of the existing methods AdamW, GaLore [Zhao et al., 2024], ReLoRA [Lialin et al., 2024]. Overall best method is shown in bold.

Methods	Param Rank	d/r	Forward Pass	Throughput (examples/sec)	Micro-batch Size
Full-rank	1024	1	Wx	137.3	24
GaLore	256	4	Wx	136.5	24
ReLoRA	256	4	(W + AB)x	115.2	24
ReLoRA	128	8	(W + AB)x	129.0	24
ReLoRA	64	16	(W + AB)x	135.2	24
ReLoRA	256	4	(W + AB)x	120.6	48
ReLoRA	128	8	(W + AB)x	133.6	48
ReLoRA	64	16	(W + AB)x	142.1	48

Table 2: Throughput analysis of ReLoRA with respect to rank, d/r (hidden size / rank), throughput, and micro-batch size.

requiring 2.31 GB of memory and a throughout of 120 examples per second. Overall, GaLore-dynamics-error-feedback (scale 0.2) outperformed all other optimizers as well as all low-rank model decomposition methods, achieving the lowest validation perplexity of 20.66, memory usage of 2.83 GB and throughput of 141.5 examples per second. We show that in order to achieve better throughput with ReLoRA style methods, we need to have higher hidden dimension to rank ratio and higher micro-batch size. This is important to understand the true saving when we are using low-rank model decomposition methods to train large-scale LLMs. Furthermore, we demonstrate other avenues like using residual gradient as error-feedback and dynamic update frequency to improve GaLore. As next steps, we will explore how our proposed efficient methods perform for training larger-scale LLMs.

8 Limitations

Our experiments were conducted on V100 GPU, which lack native bf16 support, potentially affecting the reported throughput and memory requirements of the methods included in this study. While our results are comparable within this setup, the performance and hence rankings of methods may vary on other GPU such as A100 or H100 which have native bf16 support Future work will include verifying the results on A100 or H100.

References

A. Han, J. Li, W. Huang, M. Hong, A. Takeda, P. Jawanpuria, and B. Mishra. SLTrain: a sparse plus low-rank approach for parameter and memory efficient pretraining. *arXiv preprint arXiv:2406.02214*, 2024a.

Z. Han, C. Gao, J. Liu, S. Q. Zhang, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024b.

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. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

T. Jiang, S. Huang, S. Luo, Z. Zhang, H. Huang, F. Wei, W. Deng, F. Sun, Q. Zhang, D. Wang, and F. Zhuang. MoRA: High-Rank Updating for Parameter-Efficient Fine-Tuning. *CoRR*, abs/2405.12130, 2024. doi: 10.48550/ARXIV.2405.12130. URL `https://doi.org/10.48550/arXiv.2405.12130`.

V. Lialin, V. Deshpande, and A. Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

V. Lialin, S. Muckatira, N. Shivagunde, and A. Rumshisky. ReLoRA: High-Rank Training Through Low-Rank Updates. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=DLJznSp6X3`.

S. Liu, C. Wang, H. Yin, P. Molchanov, Y. F. Wang, K. Cheng, and M. Chen. DoRA: Weight-Decomposed Low-Rank Adaptation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=3d5CIRG1n2.

S. Loeschcke, M. Toftrup, M. J. Kastoryano, S. Belongie, and V. Snæbjarnarson. LoQT: Low Rank Adapters for Quantized Training. *arXiv preprint arXiv:2405.16528*, 2024.

F. Meng, Z. Wang, and M. Zhang. PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models. *arXiv preprint arXiv:2404.02948*, 2024.

I.-V. Modoranu, M. Safaryan, G. Malinovsky, E. Kurtic, T. Robert, P. Richtarik, and D. Alistarh. MicroAdam: Accurate Adaptive Optimization with Low Space Overhead and Provable Convergence. *arXiv preprint arXiv:2405.15593*, 2024.

A. Muhamed, O. Li, D. Woodruff, M. Diab, and V. Smith. Grass: Compute efficient low-memory llm training with structured sparse gradients. *arXiv preprint arXiv:2406.17660*, 2024.

N. M. Shazeer. Glu variants improve transformer. *ArXiv*, abs/2002.05202, 2020. URL https://api.semanticscholar.org/CorpusID:211096588.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

N. Vyas, D. Morwani, and S. M. Kakade. AdaMeM: Memory Efficient Momentum for Adafactor. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024. URL https://openreview.net/forum?id=fZqMVTz7K5.

L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*, 2023.

B. Zhang and R. Sennrich. Root Mean Square Layer Normalization. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

Y. Zhang, C. Chen, Z. Li, T. Ding, C. Wu, Y. Ye, Z.-Q. Luo, and R. Sun. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*, 2024.

J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=hYHsrKDiX7.

A Training loss and evaluation loss curves for all methods

We show the training and evaluation loss for all low-rank model decomposition methods in Figures 4 and 5. We see ReLoQRa with reset frequency 10k and warm start gives the best validation perplexity among all low-rank decomposition techniques.

In Figures 6 and 7 show the training and evaluation loss for all optimizers. We see GaLore-dynamics-EF-0.2 gives the best validation perplexity among optimizers.

Figures 8 and 9 show the training and evaluation loss best methods in each category.

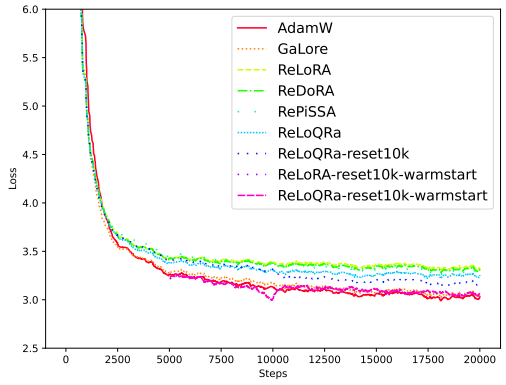

Figure 4: Training loss for low-rank decomposition methods and baselines

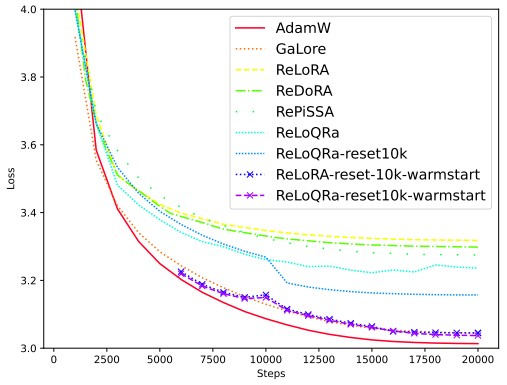

Figure 5: Evaluation loss for low-rank decomposition methods and baselines

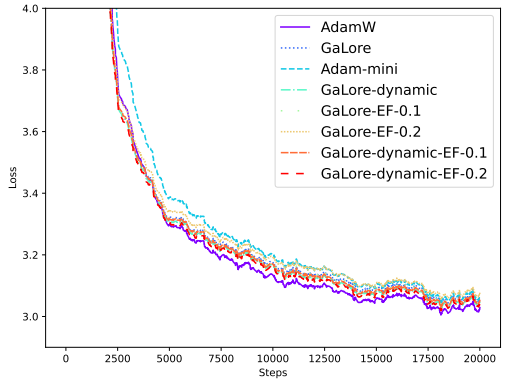

Figure 6: Training loss for optimizers and baselines

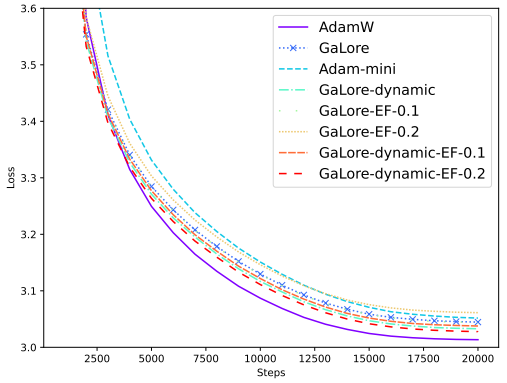

Figure 7: Evaluation loss for optimizers and baselines

B Low-rank model decomposition method trainable parameter memory usage

Methods	Perplexity	Memory in GB						Thru.
		Param	Grad	OS	Act	Extra	Total	put
ReLoRA	27.61	0.91	0.35	0.69	0.30	0.06	2.31	120.0
ReLoRA (T)		0.35	0.35	0.69	0.30	0.06	1.75	
ReDoRA	27.06	0.91	0.35	0.69	0.30	0.06	2.31	118.0
ReDoRA (T)		0.35	0.35	0.69	0.30	0.06	1.75	
RePiSSA	26.44	0.91	0.35	0.69	0.30	0.06	2.31	120.2
RePiSSA (T)		0.35	0.35	0.69	0.30	0.06	1.75	
ReLoQRa	25.43	0.91	0.35	0.69	0.30	0.06	2.31	120.5
ReLoQRa (T)		0.35	0.35	0.69	0.30	0.06	1.75	

Table 3: Low-rank decomposition methods trainable parameters consumes 0.35GB which is one third of its total parameters. Low-rank methods has lowest number of trainable parameters. 'T' stands for Trainable parameters.

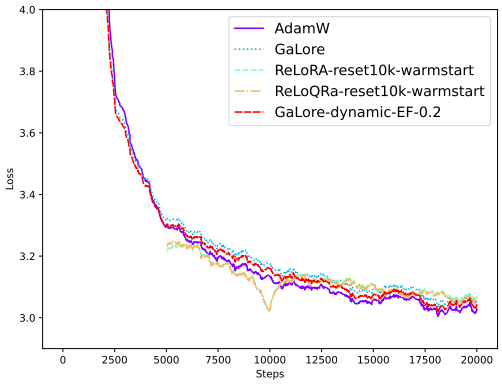

Figure 8: Training loss for overall best methods and baselines

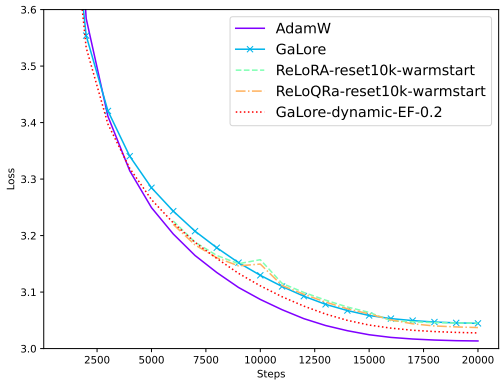

Figure 9: Evaluation loss for overall best methods and baselines