NATURAL GALORE: ACCELERATING GALORE FOR MEMORY-EFFICIENT LLM TRAINING AND FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training LLMs present significant memory challenges due to the growing size of data, weights, and optimizer states. Techniques such as data and model parallelism, gradient checkpointing, and offloading strategies address this issue but are often infeasible because of hardware constraints. To mitigate memory usage, alternative methods like Parameter Efficient Fine-Tuning (PEFT) and GaLore approximate weights or optimizer states. PEFT methods, such as LoRA and ReLoRa, have gained popularity for fine-tuning LLMs, though they are not appropriate for pretraining and require a full-rank warm start. In contrast, GaLore allows full-parameter learning while being more memory-efficient. In this work, we introduce *Natural GaLore*, which efficiently applies the inverse Empirical Fisher Information Matrix to low-rank gradients using the Woodbury Identity. We demonstrate that incorporating second-order information significantly improves the convergence rate, especially when the iteration budget is limited. Empirical pre-training on 60M, 300M, and 1.1B parameter Llama models on C4 data demonstrates significantly lower perplexity over GaLore, without additional memory overhead. Furthermore, fine-tuning the TinyLlama 1.1B model for function calling using the TinyAgent framework shows that *Natural GaLore* achieving 83.5% accuracy on the TinyAgent dataset, significantly outperforms LoRA 79% and even GPT4-Turbo with 79.08%, all while using 30% less memory.

1 INTRODUCTION

Large Language Models (LLMs) have achieved remarkable performance across various disciplines, including conversational AI and language translation. However, training and fine-tuning these models demand enormous computational resources and are highly memory-intensive. This substantial memory requirement arises not only from storing billions of trainable parameters but also from the need to store associated gradients and optimizer states—such as gradient momentum and variance in optimizers like Adam and AdamW—which can consume even more memory than the parameters themselves (Raffel et al., 2020; Touvron et al., 2023; Chowdhery et al., 2022).

To quantify this, consider a model with Ψ parameters. Storing these parameters and their gradients in 16-bit precision formats like FP16 or BF16 requires 2Ψ bytes each. Optimizer states are typically stored in 32-bit precision (FP32) for numerical stability, necessitating an additional 4Ψ bytes for each of the parameters, gradient momentum and variance, amounting to 12Ψ bytes. Therefore, the total memory requirement sums up to 16Ψ bytes. When accounting for model-dependent memory such as activations during forward and backward passes, as well as residual memory like temporary buffers and memory fragmentation, the overall memory footprint can easily exceed 18Ψ bytes.

This enormous memory demand poses significant challenges, especially when training LLMs on hardware with limited memory capacity. As models continue to scale in size, efficient memory utilization becomes critical for making training feasible and accessible.

Parallel and Distributed Training Techniques To mitigate the substantial memory requirements in training large language models, researchers have developed various distributed computing techniques that leverage system-level optimizations and hardware resources. One prominent framework is *Distributed Data Parallel (DDP)* that combines *data parallelism* where the training dataset is

partitioned across multiple devices or nodes, each holding a replica of the model with efficient gradient synchronization mechanisms, minimizing communication overhead. While data parallelism efficiently utilizes multiple GPUs, it can still face memory bottlenecks when model sizes exceed the memory capacity of a single device.

Model parallelism addresses this limitation by partitioning the model itself across multiple devices, allowing for the training of models that are too large to fit into the memory of a single GPU. Techniques like pipeline parallelism (Huang et al., 2019) and tensor parallelism (Shoeybi et al., 2019) enable the distribution of different layers or partitions of layers across devices. However, model parallelism introduces communication overhead and can be complex to implement effectively.

Another effective technique is *gradient checkpointing* (Chen et al., 2016), which reduces memory usage by selectively storing only a subset of activations during the forward pass and recomputing them during the backward pass as needed. This approach trades increased computational overhead for reduced memory consumption, enabling the training of deeper models without exceeding memory constraints.

Memory offloading strategies, such as those implemented in ZeRO-Offload (Rajbhandari et al., 2020), move optimizer states and gradients to CPU memory when not actively in use, freeing up GPU memory for other operations. The *Zero Redundancy Optimizer (ZeRO)* (Rajbhandari et al., 2020) further partitions optimizer states and gradients across data-parallel processes, eliminating redundancy and significantly reducing memory footprint. *Fully Sharded Data Parallel (FSDP)* (Zhao et al., 2020) extends this concept by sharding model parameters in addition to optimizer states and gradients.

These system-level optimizations have been instrumental in training state-of-the-art LLMs such as LLaMA 1/2/3 (Touvron et al., 2023), GPT-3/4 (Brown et al., 2020), Mistral (Jiang et al., 2023), and Gopher (Rae et al., 2021) on multi-node, multi-GPU clusters.

While these distributed computing solutions enable the training of large models by leveraging extensive hardware resources, they come with increased system complexity and operational costs. Setting up and managing large-scale distributed training infrastructure can be challenging and may not be accessible to all researchers or organizations. Moreover, communication overhead and synchronization issues can impact training efficiency, particularly as models and datasets continue to grow in size.

Therefore, there is a pressing need for alternative approaches that reduce memory consumption without relying solely on distributed computing resources. Optimization techniques that approximate parameters or optimizer states, offer a promising direction for making LLM training more accessible and efficient.

Parameter-Efficient Fine-Tuning Parameter-Efficient Fine-Tuning (PEFT) techniques allow for the efficient adaptation of pre-trained language models to various downstream applications without the need to fine-tune all the model's parameters (Ding et al., 2022). By updating only a small subset of parameters, PEFT methods significantly reduce the computational and memory overhead associated with full-model fine-tuning.

Among these techniques, the popular Low-Rank Adaptation (LoRA) (Hu et al., 2022) *reparameterizes* a weight matrix $W \in \mathbb{R}^{m \times n}$ as:

$$W = W_0 + BA, \tag{1}$$

where W_0 is a frozen full-rank pre-trained weight matrix, and $B \in \mathbb{R}^{m \times r}$ and $A \in \mathbb{R}^{r \times n}$ are trainable low-rank adapters to be learned during fine-tuning. Since the rank $r \ll \min(m, n)$, the adapters B and A contain significantly fewer trainable parameters, leading to reduced memory requirements for both parameter storage and optimizer states.

LoRA has been extensively used to reduce memory usage during fine-tuning, effectively enabling large models to be adapted to new tasks with minimal additional memory overhead. Its variant, ReLoRA (Lialin & Schatz, 2023), extends this approach to pre-training by periodically updating the frozen weight matrix W_0 using the previously learned low-rank adapters. This incremental updating allows for continual learning without the need to store full optimizer states for all parameters,leading to faster training times and lower computational costs. Furthermore, this allows for rapid adaptation

of large models to multiple downstream tasks without the need to store separate copies of the entire model for each task.

There are a few variants of LoRA proposed to enhance its performance (Renduchintala et al., 2023; Sheng et al., 2023; Zhang et al., 2023; Xia et al., 2024), supporting multi-task learning (Wang et al., 2023b), and further reducing the memory footprint (Dettmers et al., 2023). Lialin & Schatz (2023) proposed ReLoRA, a variant of LoRA designed for pre-training, but requires a full-rank training warmup to achieve comparable performance as the standard baseline. Inspired by LoRA, Hao et al. (2023) also suggested that gradients can be compressed in a low-rank subspace, and they proposed to use random projections to compress the gradients. There have also been approaches that propose training networks with low-rank factorized weights from scratch (Kamalakara et al., 2022; Wang et al., 2023a; Zhao et al., 2023).

Despite their benefits, recent works have highlighted several limitations of low-rank reparameterization approaches. LoRA does not always achieve performance comparable to full-rank fine-tuning, particularly in complex tasks (Xia et al., 2024). In pre-training from scratch, methods like ReLoRA require an initial phase of full-rank model training as a warm-up before optimizing in the low-rank subspace (Lialin & Schatz, 2023).

These limitations may stem from inadequate low-rank approximation of the optimal weight matrices in large models, as well as altered training dynamics due to the reparameterization introduced by low-rank adapters. The shortcomings of low-rank parameter reparameterization suggest that alternative strategies are needed to achieve both memory efficiency and high performance.

Gradient Low-Rank Projection (GaLore) One promising direction is to approximate the *optimizer states* instead of the parameters themselves. By reducing the memory footprint associated with optimizer states, it is possible to maintain full-parameter learning—thus preserving model capacity and performance—while still achieving significant memory savings.

The core idea behind GaLore is to harness the slowly changing low-rank structure of the *gradient* matrix $G \in \mathbb{R}^{m \times n}$ corresponding to the weight matrix W, rather than approximating W itself as a low-rank matrix. During neural network training, gradients naturally exhibit low-rank properties—a phenomenon studied extensively in both theoretical and practical settings (Zhao et al., 2022; Cosson et al., 2023; Yang et al., 2023). This intrinsic low-rank structure of gradients has been applied to reduce communication costs (Wang et al., 2018; Vogels et al., 2020) and to decrease memory footprints during training (Gooneratne et al., 2020; Huang et al., 2023; **?**).

Specifically, GaLore computes two projection matrices $P \in \mathbb{R}^{m \times r}$ and $Q \in \mathbb{R}^{n \times r}$ to project the gradient matrix G into a low-rank form:

$$G_{\text{low-rank}} = P^\top G Q. \tag{2}$$

Here, $r \ll \min(m, n)$ is the target rank, and $G_{\text{low-rank}}$ serves as an efficient approximation of the original gradient. The projection matrices P and Q are updated periodically (e.g., every 200 iterations) based on the principal components of recent gradients, which incurs minimal amortized computational cost.

By operating on low-rank approximations of the gradients, GaLore significantly reduces the memory footprint associated with storing optimizer states that rely on element-wise gradient statistics. In practice, this can yield up to **30%** memory reduction compared to methods like LoRA during pre-training. Moreover, GaLore maintains full-parameter learning, allowing for updates to all model parameters, which can lead to better generalization and performance compared to low-rank adaptation methods. Further, GaLore is agnostic to the choice of optimizer and can be easily integrated into existing optimization algorithms with minimal code modifications.

While GaLore offers significant memory savings and enables full-parameter learning, its performance has not yet matched that of original optimizers like Adam or AdamW. Specifically, GaLore's reliance on low-rank gradient approximations can lead to suboptimal convergence rates and may not fully capture the rich optimization dynamics that these standard optimizers achieve with full gradients and optimizer states. These limitations suggest that while GaLore is a valuable step toward memory-efficient training, further enhancements are necessary to bridge the performance gap with standard optimizers.

Our Approach To overcome the limitations of GaLore—particularly its performance gap with standard optimizers like Adam and AdamW—we introduce *Natural GaLore*. This method enhances GaLore by incorporating second-order information, specifically the curvature of the loss landscape, into the optimization process. By accounting for this curvature, Natural GaLore adjusts parameter updates more effectively, leading to faster convergence.

Natural GaLore efficiently applies the inverse of the Empirical Fisher Information Matrix (FIM) to the low-rank gradients obtained from GaLore. Instead of computing and storing the full inverse FIM—which is computationally infeasible for large-scale models—we utilize the Woodbury Identity to perform this operation efficiently within the low-rank subspace. This approach allows us to incorporate second-order information without incurring significant computational or memory overhead. To address this challenge, we propose *Natural GaLore*, an online natural gradient algorithm that operates in a low-rank subspace of the gradient space. By projecting gradients onto this subspace and approximating the FIM within it, we efficiently compute natural gradient updates without explicit layer-wise information or significant computational overhead as is seen in K-Fac, INGD, SINGD, etc.

By integrating second-order information, Natural GaLore significantly improves the convergence rate, especially when the iteration budget is limited. This enhancement brings the performance of GaLore closer to that of standard optimizers like Adam or AdamW, effectively bridging the performance gap observed in previous methods.

We validate the effectiveness of Natural GaLore through extensive empirical evaluations. Pretraining experiments on LLaMA models with 60M, 300M, and 1.1B parameters using the C4 dataset demonstrate that Natural GaLore achieves significantly lower perplexity compared to GaLore, all without additional memory overhead. This indicates that our method converges faster and reaches better optima within the same computational budget.

Furthermore, we showcase the practical benefits of Natural GaLore in fine-tuning tasks. Specifically, we fine-tune the TinyLlama 1.1B model for function calling using the TinyAgent framework. Our results show that Natural GaLore significantly outperforms LoRA in this setting, achieving an accuracy of **83.5%** on the TinyAgent dataset. This performance not only surpasses LoRA but also exceeds that of GPT-4o, which achieves **79.08%** accuracy—all while using **30%** less memory.

In summary, Natural GaLore addresses the shortcomings of previous methods by efficiently incorporating second-order information into the optimization process. This leads to faster convergence rates and improved performance without additional memory requirements, making it a promising approach for training large-scale language models under memory constraints.

2 Accelerating GaLore with Natural Gradients

2.1 Causal Language Model Objective

Generative LLMs are trained with respect to the Causal Language Model (CLM) objective, where the task is to predict the next token in a sequence based solely on the tokens that have come before it. This approach is called "causal" because it respects the temporal order of language, ensuring that the model's predictions at any point depend only on past and not future inputs.

Given a sequence of tokens $x = (x_1, x_2, \ldots, x_T)$, the causal language model aims to maximize the likelihood of the sequence by decomposing it into a product of conditional probabilities:

$$\text{Prob}_\theta(x) = \prod_{t=1}^{T} \text{Prob}_\theta(x_t \mid x_{<t})$$

where:

- $x_{<t} = (x_1, x_2, \ldots, x_{t-1})$ represents all tokens before position t.
- $\text{Prob}_\theta(x_t \mid x_{<t})$ is the probability of the next token given all previous tokens and the parameter $\theta \in \mathbb{R}^{n \times m}$.

The training objective is to minimize the *negative log-likelihood (NLL)* of the observed sequences, which is equivalent to minimizing the *cross-entropy loss* between the predicted probability distribution and the actual next token:

$$\Phi(\theta) = -\sum_{t=1}^{T} \log \text{Prob}_\theta(x_t \mid x_{<t}) \tag{3}$$

This loss penalizes the model more when it assigns lower probabilities to the correct next token. By minimizing this loss, the model learns to assign higher probabilities to appropriate continuations of text. However, the loss is non-convex and high-dimensional, making optimization challenging.

2.2 Low Rank Gradient Descent

Now stochastic gradient descent algorithms are iterative, where the goal at each step is to find the optimal update direction that minimizes the loss function locally. Now in the case of GaLore, the update direction, say u_k is restricted to the affine subspace $u_k \in \theta_k + \text{Range}\left(\mathbf{P}_k^T\right)$, where $\mathbf{P}_k \in \mathbb{R}^{r \times n}$ is the projection matrix defined by the top r left singular vectors of the gradient $\nabla_\theta \Phi(\theta)$. Then the local neighborhood around this update can be define using the Taylor series expansion as:

$$\Phi(\theta_k + \mathbf{P_k^T u_k}) \approx \Phi(\theta_k) + \mathbf{g_k}^T \mathbf{u_k} + \frac{1}{2} \mathbf{u_k^T H_k u_k} \tag{4}$$

where $\mathbf{g_k} = P_k \nabla_\theta \Phi(\theta_k)$ is the low rank projected gradient and $\mathbf{H_k} = P_k \nabla_\theta^2 \Phi(\theta) P_k^T$ is the Hessian matrix.

However, the Hessian matrix $\mathbf{H_k}$ is often computationally expensive to compute and store, especially for large-scale language models (LLMs) with billions of parameters. Fortunately, exactly under the condition that the loss function can be represented in terms of KL divergence between the true and approximated distributions i.e. the CLM objective 3, then $\mathbf{H_k}$ can be approximated by the Fisher Information Matrix (FIM). The FIM is defined as the expectation of the Hessian of the negative log-likelihood with respect to the data distribution:

$$\mathbf{F_k} = \mathbb{E}_{x \sim p_{\text{data}}} [\mathbf{H_k}]$$

The FIM captures the curvature of the loss landscape and provides a natural metric for the optimization process. Hence is able to better adjust parameter updates according to the geometry of the parameter space. However, as the theoretical data distribution is unknown, in practice we need to estimate is using the empirical FIM (Martens, 2014) defined by:

$$\hat{\mathbf{F}}_k = \frac{1}{h} \sum_{k=1}^{h} \mathbf{g_k} \mathbf{g_k}^T$$

where h is the history of gradients from past batches we would like to consider.

Then the optimal direction u_k^* which minimizes the loss in this local neighborhood is given by (cite Fuji et al paper):

$$\mathbf{u_k^*} = \hat{\mathbf{F}}_k^{-1} \mathbf{g_k} \tag{5}$$

This leads to the optimal gradient descent update step:

$$\theta_{k+1} = \theta_k - \eta \mathbf{P_k^T u_k^*}$$

for some learning rate η.

Many popular stochastic optimization algorithms approximate the diagonal of the empirical FIM using second moment estimates of the gradient $\mathbf{g_k}$, which when added with Polyak style parameter averaging (i.e. momentum), these methods asymptotically achieve the optimal Fisher efficient convergence rate (Martens, 2020).

For example, in the case of Adam (), the optimal update step is approximated by including the momentum term $m_k \in \mathbb{R}^{r \times m}$ and the learning rate η is scaled by the square root of the second moment estimate $v_k \in \mathbb{R}^{r \times m}$. With all operations being elementwise, the update direction becomes:

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k \tag{6}$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2 \tag{7}$$

$$\mathbf{u_k^*} = m_k / \sqrt{v_k + \epsilon} \tag{8}$$

This update when applied to 2.2, gives the GaLore optimization algorithm, which is memory efficient as it only requires storing the projection matrix and the costly optimizer states (g_k, m_k, v_k) are now significantly reduced by a factor of $\frac{n}{r}$, where r the rank, can be chosen based on memory limitations.

2.3 Natural GaLore and Fisher Efficiency

However, the performance of GaLore is still not on par with the Adam optimization on the original space. To bridge this gap, we propose Natural GaLore, which uses the full empirical FIM, thereby incorporating the missing second-order interaction information in the optimization process.

As we will now argue, that this leads to a much more favorable dependence on the starting point, which means that they can make much more progress given a limited iteration budget. Further, when using a decaying learning rate schedule like with AdamW (reference), the asymptotic convergence rate can be faster by a large constant factor (Martens, 2020).

Natural gradient descent is known (Martens, 2020) to be *Fisher Efficient*, exactly for our loss function 3. Fisher efficiency means that the natural gradient estimator achieves the lowest possible variance among all unbiased estimators of the gradient.

For Natural GaLore, the gradient descent update 2.2 leads to a sequence of estimates θ_k whose variance satisfies (Amari, 1998):

$$\text{Var}[\theta_k] = \frac{1}{mk} \mathbf{F}_k^{-1}(\theta_k^*) + \mathcal{O}\left(\frac{1}{k^2}\right) \tag{9}$$

which is asymptotically the smallest possible variance matrix, i.e. the Cramér-Rao lower bound, that any unbiased estimator computed from mk training samples can have, with m being the batch size.

Here θ_k^* is the local optima, in the neighborhood defined by the Taylor series expansion 4 around the update direction. This is an important caveat, as the guarantee is only for local convergence, in a convex neighborhood. As the loss function is non-convex, the property can not be stated to hold for the global optimum.

Now the result also relies on the computation of the exact FIM $\mathbf{F}_k(\theta_k)$ using the entire data distribution. This is obviously impractical and hence we use the empirical FIM $\hat{\mathbf{F}}_\mathbf{k}$ instead. The Fisher efficiency guarantee is then only approximately satisfied. However, even in that case we get a variance reduction in the gradient estimates. This can lead to faster convergence and better optimization performance in the early stages of training large-scale models, making it especially valuable for training with limited iteration budget.

Further, incorporating second-order information through the empirical FIM allows the optimizer to account for the curvature of the loss landscape. This enables natural gradient descent to take more informed steps compared to standard gradient descent, potentially escaping flat regions or navigating steep ravines more effectively.

In (Martens, 2020) it was shown that the expected update direction can be expressed as a sum of two terms, one that scales as $\mathcal{O}(1/k)$, which is independent of the starting point and another that scales as $\mathcal{O}(1/k^2)$, which is dependent on the starting point. In the case of using Polyak averaging (i.e. momentum) the first term becomes independent of the choice of FIM estimator and the standard stochastic gradient descent step becomes Fisher efficient. However, using the empirical FIM estimator can significantly reduce the constant factor associated with the starting point dependent

second term, which can lead to practical performance gains in finite iteration regimes (despite being negligible for large k).

Finally, the Fisher efficiency result also assumes that the model is capable of perfectly capturing the true data distribution—a condition known as *realizability*. However, with growing size of LLMs, this assumption is likely to hold, and thereby leading to the guarantee being satisfied.

In conclusion, while the theoretical Fisher efficiency of natural gradient descent comes with significant assumptions that may not hold exactly, in practical low resource settings the method offers substantial advantages when the number of iterations is limited. This makes Natural GaLore a promising approach for training large-scale language models under memory constraints.

2.4 Algorithm Overview

Our Natural GaLore algorithm is designed to efficiently apply the inverse Empirical Fisher Information Matrix to low-rank gradients using the Woodbury Identity. Most of the steps in the algorithm are similar to GaLore, with the key difference being the incorporation of the natural gradient transform. The outline of the GaLore algorithm is as follows:

In order to implement the natural gradient transform, we need to compute the inverse of the Empirical Fisher Information Matrix and apply it to the gradient $\mathbf{g_k}$. This can be done using the Woodbury Identity, which allows us to efficiently compute the inverse of a matrix of the form $A+UBU^T$. The Woodbury Identity states that:

$$(A + UBU^T)^{-1} = A^{-1} - A^{-1}U(B^{-1} + U^T A^{-1} U)^{-1} U^T A^{-1}$$

Now if we choose

$$\hat{\mathbf{F}}_k = \lambda I + GG^T \tag{10}$$
$$A = \lambda I \tag{11}$$
$$U = G \tag{12}$$
$$B = I \tag{13}$$
$$\tag{14}$$

where $G = [\text{vec}(\mathbf{g_k}), \text{vec}(\mathbf{g_{k-1}}), \ldots, \text{vec}(\mathbf{g_{k-s}})]$ is the stacked gradient matrix over the past $s \approx 20$ gradients and λ is a small constant for Tikhonov regularization, then the inverse of the empirical FIM applied to the gradient $\mathbf{g_k}$ i.e. the natural gradient $\tilde{\mathbf{g}}_k = \hat{\mathbf{F}}_k^{-1}\mathbf{g_k}$ can be computed as:

$$\tilde{\mathbf{g}}_k = \frac{1}{\lambda}\mathbf{g_k} - \frac{1}{\lambda^2} G \left(I + \frac{1}{\lambda} G^T G \right)^{-1} G^T \mathbf{g_k} \tag{15}$$

To compute the above formula efficiently, let $S = I + \frac{1}{\lambda} G^T G \in \mathbb{R}^{s \times s}$ and $y = G^T \mathbf{g_k}$. The idea is to use Cholesky decomposition to solve for z in

$$Sz = y \tag{16}$$

which can be done in $\mathcal{O}(s^2)$ time. Then the natural gradient estimate can be computed using only matrix vector products, which is very memory efficient:

$$\tilde{\mathbf{g}}_k = \frac{1}{\lambda}\mathbf{g_k} - \frac{1}{\lambda^2} Gz \tag{17}$$

This natural gradient estimate can then be applied to the Adam optimizer 8 and we update the model parameters, the same way as in GaLore. This allows us to efficiently incorporate second-order information into the optimization process, leading to faster convergence and better performance, especially in low-resource settings.

3 Experiments

We evaluate Natural GaLore on both pre-training and fine-tuning of LLMs. All experiments run on NVIDIA A100 GPUs.

Table 1: Pre-training LLaMA 7B on C4 dataset for 150K steps. Validation perplexity and memory estimate are reported.

	Mem	40K	80K	120K	150K
8-bit Natural GaLore	18G	17.94	15.39	14.95	14.65
8-bit Adam	26G	18.09	15.47	14.83	14.61
Tokens (B)		5.2	10.5	15.7	19.7

Table 2: Comparison with low-rank algorithms on pre-training various sizes of LLaMA models on C4 dataset. Validation perplexity is reported, along with a memory estimate of the total of parameters and optimizer states based on BF16 format. The actual memory footprint of Natural GaLore is reported in Fig. 2.

	60M	130M	350M	1B
Full-Rank	34.06 (0.36G)	25.08 (0.76G)	18.80 (2.06G)	15.56 (7.80G)
Natural GaLore	**34.88** (0.24G)	**25.36** (0.52G)	**18.95** (1.22G)	**15.64** (4.38G)
Low-Rank	78.18 (0.26G)	45.51 (0.54G)	37.41 (1.08G)	142.53 (3.57G)
LoRA	34.99 (0.36G)	33.92 (0.80G)	25.58 (1.76G)	19.21 (6.17G)
ReLoRA	37.04 (0.36G)	29.37 (0.80G)	29.08 (1.76G)	18.33 (6.17G)
r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

Pre-training on C4. To evaluate its performance, we apply Natural GaLore to train LLaMA-based large language models on the C4 dataset. C4 dataset is a colossal, cleaned version of Common Crawl's web crawl corpus, which is mainly intended to pre-train language models and word representations (**?**). To best simulate the practical pre-training scenario, we train without data repetition over a sufficiently large amount of data, across a range of model sizes up to 7 Billion parameters.

Architecture and hyperparameters. We follow the experiment setup from Lialin & Schatz (2023), which adopts a LLaMA-based[3] architecture with RMSNorm and SwiGLU activations (**?**Shazeer, 2020; Touvron et al., 2023). For each model size, we use the same set of hyperparameters across methods, except the learning rate. We run all experiments with BF16 format to reduce memory usage, and we tune the learning rate for each method under the same amount of computational budget and report the best performance. The details of our task setups and hyperparameters are provided in the appendix.

Fine-tuning on GLUE tasks. GLUE is a benchmark for evaluating the performance of NLP models on a variety of tasks, including sentiment analysis, question answering, and textual entailment (Wang et al., 2019). We use GLUE tasks to benchmark Natural GaLore against LoRA for memory-efficient fine-tuning.

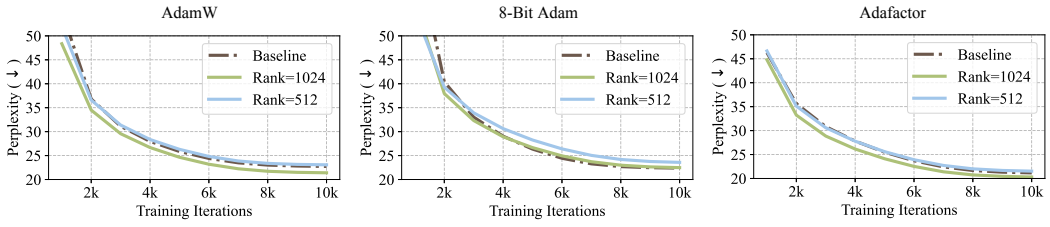

Figure 1: Applying Natural GaLore to different optimizers for pre-training LLaMA 1B on C4 dataset for 10K steps. Validation perplexity over training steps is reported. We apply Natural GaLore to each optimizer with the rank of 512 and 1024, where the 1B model dimension is 2048.

3.1 Comparison with Existing Low-Rank Methods

We first compare Natural GaLore with existing low-rank methods using Adam optimizer across a range of model sizes.

[3]LLaMA materials in our paper are subject to LLaMA community license.

Full-Rank Our baseline method that applies Adam optimizer with full-rank weights and optimizer states.

Low-Rank We also evaluate a traditional low-rank approach that represents the weights by learnable low-rank factorization: $W = BA$ (Kamalakara et al., 2022).

LoRA Hu et al. (2022) proposed LoRA to fine-tune pre-trained models with low-rank adaptors: $W = W_0 + BA$, where W_0 is fixed initial weights and BA is a learnable low-rank adaptor. In the case of pre-training, W_0 is the full-rank initialization matrix. We set LoRA alpha to 32 and LoRA dropout to 0.05 as their default settings.

ReLoRA Lialin & Schatz (2023) proposed ReLoRA, a variant of LoRA designed for pre-training, which periodically merges BA into W, and initializes new BA with a reset on optimizer states and learning rate. ReLoRA requires careful tuning of merging frequency, learning rate reset, and optimizer states reset. We evaluate ReLoRA without a full-rank training warmup for a fair comparison.

For Natural GaLore, we set subspace frequency T to 200 and scale factor α to 0.25 across all model sizes in Table 2. For each model size, we pick the same rank r for all low-rank methods, and we apply them to all multi-head attention layers and feed-forward layers in the models. We train all models using Adam optimizer with the default hyperparameters (e.g., $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). We also estimate the memory usage based on BF16 format, including the memory for weight parameters and optimizer states. As shown in Table 2, Natural GaLore outperforms other low-rank methods and achieves comparable performance to full-rank training. We note that for 1B model size, Natural GaLore even outperforms full-rank baseline when $r = 1024$ instead of $r = 512$. Compared to LoRA and ReLoRA, Natural GaLore requires less memory for storing model parameters and optimizer states. A detailed training setting of each model and memory estimation for each method are in the appendix.

3.2 Natural GaLore with Memory-Efficient Optimizers

We demonstrate that Natural GaLore can be applied to various learning algorithms, especially memory-efficient optimizers, to further reduce the memory footprint. We apply Natural GaLore to AdamW, 8-bit Adam, and Adafactor optimizers (Shazeer & Stern, 2018; Loshchilov & Hutter, 2017; Dettmers et al., 2022). We consider Adafactor with first-order statistics to avoid performance degradation.

We evaluate them on LLaMA 1B architecture with 10K training steps, and we tune the learning rate for each setting and report the best performance. As shown in Fig. 1, applying Natural GaLore does not significantly affect their convergence. By using Natural GaLore with a rank of 512, the memory footprint is reduced by up to 62.5%, on top of the memory savings from using 8-bit Adam or Adafactor optimizer. Since 8-bit Adam requires less memory than others, we denote 8-bit Natural GaLore as Natural GaLore with 8-bit Adam, and use it as the default method for the following experiments on 7B model pre-training and memory measurement.

3.3 Scaling up to LLaMA 7B Architecture

Scaling ability to 7B models is a key factor for demonstrating if Natural GaLore is effective for practical LLM pre-training scenarios. We evaluate Natural GaLore on an LLaMA 7B architecture with an embedding size of 4096 and total layers of 32. We train the model for 150K steps with 19.7B tokens, using 8-node training in parallel with a total of 64 A100 GPUs. Due to computational constraints, we compare 8-bit Natural GaLore ($r = 1024$) with 8-bit Adam with a single trial without tuning the hyperparameters. As shown in Table 1, after 150K steps, 8-bit Natural GaLore achieves a perplexity of 14.65, comparable to 8-bit Adam with a perplexity of 14.61.

3.4 Memory-Efficient Fine-Tuning

Natural GaLore not only achieves memory-efficient pre-training but also can be used for memory-efficient fine-tuning. We fine-tune pre-trained RoBERTa models on GLUE tasks using Natural

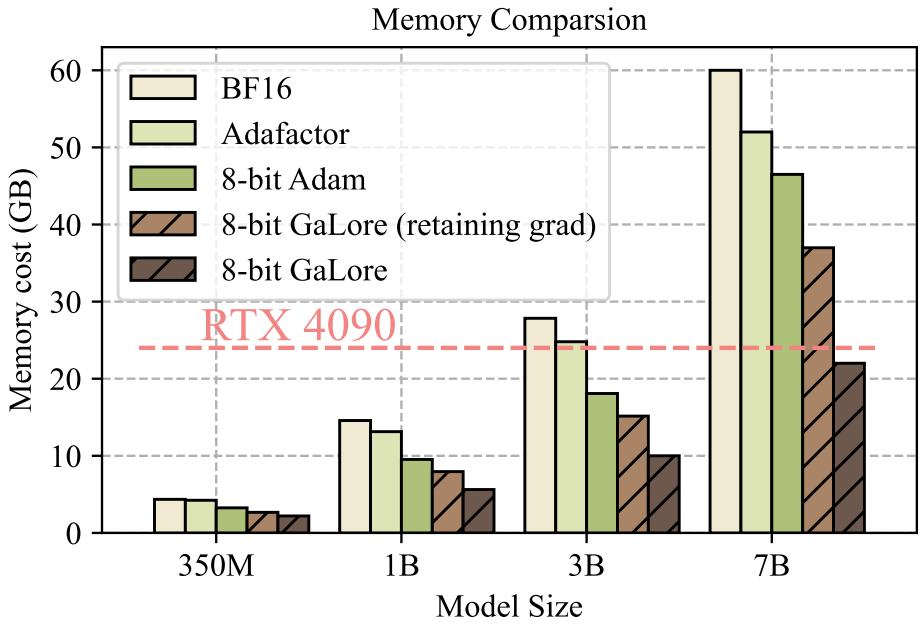

Figure 2: Memory usage for different methods at various model sizes, evaluated with a token batch size of 256. 8-bit Natural GaLore (retaining grad) disables per-layer weight updates but stores weight gradients during training.

Table 3: Evaluating Natural GaLore for memory-efficient fine-tuning on GLUE benchmark using pre-trained RoBERTa-Base. We report the average score of all tasks.

	Memory	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Full Fine-Tuning	747M	62.24	90.92	91.30	79.42	94.57	87.18	92.33	92.28	86.28
Natural GaLore (rank=4)	253M	60.35	**90.73**	**92.25**	**79.42**	**94.04**	**87.00**	**92.24**	91.06	**85.89**
LoRA (rank=4)	257M	**61.38**	90.57	91.07	78.70	92.89	86.82	92.18	**91.29**	85.61
Natural GaLore (rank=8)	257M	60.06	**90.82**	**92.01**	**79.78**	**94.38**	**87.17**	92.20	91.11	**85.94**
LoRA (rank=8)	264M	**61.83**	90.80	91.90	79.06	93.46	86.94	**92.25**	**91.22**	85.93

GaLore and compare its performance with a full fine-tuning baseline and LoRA. We use hyper-parameters from Hu et al. (2022) for LoRA and tune the learning rate and scale factor for Natural GaLore. As shown in Table 3, Natural GaLore achieves better performance than LoRA on most tasks with less memory footprint. This demonstrates that Natural GaLore can serve as a full-stack memory-efficient training strategy for both LLM pre-training and fine-tuning.

3.5 Measurement of Memory and Throughput

While Table 2 gives the theoretical benefit of Natural GaLore compared to other methods in terms of memory usage, we also measure the actual memory footprint of training LLaMA models by various methods, with a token batch size of 256. The training is conducted on a single device setup without activation checkpointing, memory offloading, and optimizer states partitioning (Rajbhandari et al., 2020).

Training 7B models on consumer GPUs with 24G memory. As shown in Fig. 2, 8-bit Natural GaLore requires significantly less memory than BF16 baseline and 8-bit Adam, and only requires 22.0G memory to pre-train LLaMA 7B with a small per-GPU token batch size (up to 500 tokens). This memory footprint is within 24GB VRAM capacity of a single GPU such as NVIDIA RTX 4090. In addition, when activation checkpointing is enabled, per-GPU token batch size can be increased up to 4096. While the batch size is small per GPU, it can be scaled up with data parallelism, which requires much lower bandwidth for inter-GPU communication, compared to model parallelism. Therefore, it is possible that Natural GaLore can be used for elastic training Lin et al. (2019) 7B models on consumer GPUs such as RTX 4090s.

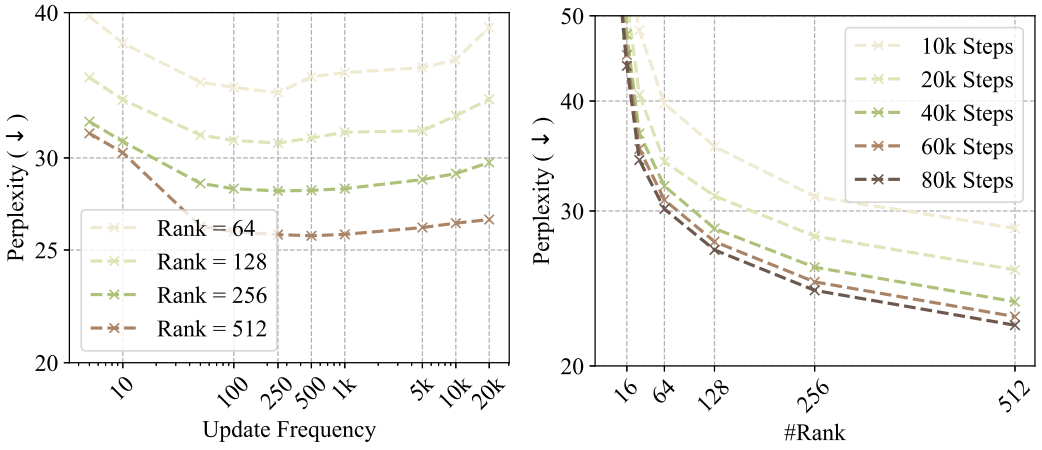

Figure 3: Ablation study of Natural GaLore on 130M models. **Left:** varying subspace update frequency T. **Right:** varying subspace rank and training iterations.

Specifically, we present the memory breakdown in Fig. **??**. It shows that 8-bit Natural GaLore reduces 37.92G (63.3%) and 24.5G (52.3%) total memory compared to BF16 Adam baseline and 8-bit Adam, respectively. Compared to 8-bit Adam, 8-bit Natural GaLore mainly reduces the memory in two parts: (1) low-rank gradient projection reduces 9.6G (65.5%) memory of storing optimizer states, and (2) using per-layer weight updates reduces 13.5G memory of storing weight gradients.

Throughput overhead of Natural GaLore. We also measure the throughput of the pre-training LLaMA 1B model with 8-bit Natural GaLore and other methods, where the results can be found in the appendix. Particularly, the current implementation of 8-bit Natural GaLore achieves 1019.63 tokens/second, which induces 17% overhead compared to 8-bit Adam implementation. Disabling per-layer weight updates for Natural GaLore achieves 1109.38 tokens/second, improving the throughput by 8.8%. We note that our results do not require offloading strategies or checkpointing, which can significantly impact training throughput. We leave optimizing the efficiency of Natural GaLore implementation for future work.

4 Ablation Study

How many subspaces are needed during pre-training? We observe that both too frequent and too slow changes of subspaces hurt the convergence, as shown in Fig. 3 (left). The reason has been discussed in Sec. **??**. In general, for small r, the subspace switching should happen more to avoid wasting optimization steps in the wrong subspace, while for large r the gradient updates cover more subspaces, providing more cushion.

How does the rank of subspace affect the convergence? Within a certain range of rank values, decreasing the rank only slightly affects the convergence rate, causing a slowdown with a nearly linear trend. As shown in Fig. 3 (right), training with a rank of 128 using 80K steps achieves a lower loss than training with a rank of 512 using 20K steps. This shows that Natural GaLore can be used to trade-off between memory and computational cost. In a memory-constrained scenario, reducing the rank allows us to stay within the memory budget while training for more steps to preserve the performance.

5 Conclusion

We propose Natural GaLore, a memory-efficient pre-training and fine-tuning strategy for large language models. Natural GaLore significantly reduces memory usage by up to 65.5% in optimizer states while maintaining both efficiency and performance for large-scale LLM pre-training and fine-tuning.

We identify several open problems for Natural GaLore, which include (1) applying Natural GaLore on training of various models such as vision transformers (Dosovitskiy et al., 2020) and diffu-

sion models (Ho et al., 2020), (2) further enhancing memory efficiency by employing low-memory projection matrices, and (3) exploring the feasibility of elastic data distributed training on low-bandwidth consumer-grade hardware.

We hope that our work will inspire future research on memory-efficient training from the perspective of gradient low-rank projection. We believe that Natural GaLore will be a valuable tool for the community, enabling the training of large-scale models on consumer-grade hardware with limited resources.

IMPACT STATEMENT

This paper aims to improve the memory efficiency of training LLMs in order to reduce the environmental impact of LLM pre-training and fine-tuning. By enabling the training of larger models on hardware with lower memory, our approach helps to minimize energy consumption and carbon footprint associated with training LLMs.

ACKNOWLEDGMENTS

We thank Meta AI for computational support. We appreciate the helpful feedback and discussion from Florian Schäfer, Jeremy Bernstein, and Vladislav Lialin. B. Chen greatly appreciates the support by Moffett AI. Z. Wang is in part supported by NSF Awards 2145346 (CAREER), 02133861 (DMS), 2113904 (CCSS), and the NSF AI Institute for Foundations of Machine Learning (IFML). A. Anandkumar is supported by the Bren Foundation and the Schmidt Sciences through AI 2050 senior fellow program.

REFERENCES

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. In *arXiv preprint arXiv:1604.06174*, 2016. URL https://arxiv.org/abs/1604.06174.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. URL https://arxiv.org/abs/2204.02311.

Victor Cosson, Baptiste Lecouat, Arthur Varre, Stéphane d'Ascoli, and Giulio Biroli. Low-rank gradient descent converges and generalizes. *arXiv preprint arXiv:2301.12995*, 2023. URL https://arxiv.org/abs/2301.12995.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2110.02861.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023. URL https://arxiv.org/abs/2305.14314.

Ning Ding, Xiang Zheng, Yujia Wang, Yifei Chen, Yichi Liu, Haitao Zheng, Xipeng Qiu, Yujun Shen, Bolin Ding, and Jie Tang. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21016–21029, 2022. URL https://proceedings.neurips.cc/paper/

2022/hash/a7663702e92787e0e3a4b0e91f1e69d3-Abstract-Conference.html.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Shamal Gooneratne, Meng Wang, Zhili Guo, Vamsi Krishna Kanuparthi, Dinesh Rajan, and Anura P Jayasumana. Low-rank gradient approximation for multi-task learning. *arXiv preprint arXiv:2011.01679*, 2020. URL https://arxiv.org/abs/2011.01679.

Yuning Hao, Shixiang Gu, and Chen Liang. Flora: Fine-grained low-rank adaptation. *arXiv preprint arXiv:2306.17878*, 2023. URL https://arxiv.org/abs/2306.17878.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Menglong Chen, Denny Chen, Zhifeng Hu, Yuxin Shen, Maxim Krikun, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, volume 32, pp. 103–112, 2019.

Zihao Huang, Lingfei Wu, and Rui Xiong. Low-rank gradient descent: Fast convergence and low memory cost. *arXiv preprint arXiv:2302.00089*, 2023. URL https://arxiv.org/abs/2302.00089.

Ye Jiang, Pengcheng Li, Zhe Gan, Jianfeng Liu, Dongdong Chen, Xiaodong Zhu, Zhangyang Li, Lijuan Wang, Jianfeng Wang, and Zicheng Liu. Mistral: Efficient composable inference for large language models. *arXiv preprint arXiv:2305.15334*, 2023.

Himanshu Kamalakara, Sneha Kudugunta, Rohit Prakash Sahu, and He He. Exploring low-rank training of deep neural networks. *arXiv preprint arXiv:2203.07261*, 2022. URL https://arxiv.org/abs/2203.07261.

Vladimir Lialin and Arthur Schatz. Relora: Low-rank finetuning reloaded. *arXiv preprint arXiv:2307.09769*, 2023. URL https://arxiv.org/abs/2307.09769.

Yuhong Lin, Chaoyue Zhao, Yongkai Wu, Dabin Luo, Lei Sun, and Bingsheng He. Dynamic mini-batch sgd for elastic distributed training. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1226–1236. IEEE, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

James Martens. New perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21:1–76, 2020.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, 2020. URL https://arxiv.org/abs/1910.02054.

Adithya Renduchintala, Pedro Rodriguez, and Mathias Creutz. Tied lora: Enhancing parameter-efficient fine-tuning with tied weights. *arXiv preprint arXiv:2306.13420*, 2023. URL https://arxiv.org/abs/2306.13420.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

Yi Sheng, Xuefei Han, Xuefeng Zhu, Yuanzhi Yang, Jiani Sun, and Guohui Zhou. S-lora: Scalable efficient model serving for massive lora models. *arXiv preprint arXiv:2306.01125*, 2023. URL https://arxiv.org/abs/2306.01125.

Mohammad Shoeybi, Mostofa Patwary, Rohan Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Thijs Vogels, Martin Jaggi, and Giorgio Patrini. Powergossip: Practical low-rank communication for decentralized optimization. In *International Conference on Machine Learning*, pp. 10592–10602, 2020. URL https://proceedings.mlr.press/v119/vogels20a.html.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

Mengzhao Wang, Zhao Liu, Yao Bai, and Yuan Gao. Cuttlefish: Low-rank model training without factorization. *arXiv preprint arXiv:2305.19635*, 2023a. URL https://arxiv.org/abs/2305.19635.

Shiqiang Wang, Gauri Joshi, Sreeram K Ghosh, and H Vincent Poor. Atomo: Communication-efficient learning via atomic sparsification. In *Advances in Neural Information Processing Systems*, volume 31, pp. 9850–9861, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/77fd8c838a3a41ee49e699528f2bbaab-Abstract.html.

Zihao Wang, Zhen Bai, and Sophia Ananiadou. Multi-lora: Efficient finetuning for democratic ai. *arXiv preprint arXiv:2305.14377*, 2023b. URL https://arxiv.org/abs/2305.14377.

Tianxiang Xia, Hao Peng, Zheyu Chen, Lemao Li, Zhiyuan He, Zhen Yang, and Wei-Ying Ma. Chain-of-thought lora: Efficient adaptation of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Zhilin Yang, Edward J Hu, Tianle Xia, Richard Socher, and Yuanzhi Li. Spectral methods in low-rank model adaptation. *arXiv preprint arXiv:2305.14683*, 2023. URL https://arxiv.org/abs/2305.14683.

Rui Zhang et al. Lora-fa: Memory-efficient low-rank adaptation via feature re-alignment. *arXiv preprint arXiv:2302.05653*, 2023. URL https://arxiv.org/abs/2302.05653.

Shangqian Zhao, Shiyu Li, and Yi Ma. Zero initialization: Initializing neural networks with zero-valued parameters. *arXiv preprint arXiv:2207.05848*, 2022. URL https://arxiv.org/abs/2207.05848.

Tianshi Zhao, Zhen Sun, Xiaodong Wang, Fei Zhou, Yang Guo, and Alexander J Smola. Extending torchelastic for stateful training jobs. *arXiv preprint arXiv:2006.06873*, 2020.

Yuwei Zhao, Yifan Zhang, et al. In-rank: Incremental low-rank learning. *arXiv preprint arXiv:2303.11246*, 2023. URL https://arxiv.org/abs/2303.11246.