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 data size, weights, and optimizer states. Techniques such as data and model parallelism, gradient checkpointing, and offloading strategies address this issue but are often infeasible due to 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, 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. This work introduces *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 speeds up optimization significantly, especially when the iteration budget is limited. Empirical pretraining on 60M, 300M, and 1.1B parameter Llama models on C4 data demonstrate 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 from storing billions of trainable parameters along with associated gradients and optimizer states.

To quantify this, consider a model with Ψ parameters which is being trained using the Adam optimizer. In this case, storing parameters and their gradients in 16-bit precision formats like FP16 or BF16 requires 2Ψ bytes each. The associated optimizer states are typically stored in 32-bit precision (FP32) for numerical stability, necessitating an additional 4Ψ bytes for each parameter, 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, and residual memory, like temporary buffers and memory fragmentation, the overall memory footprint can easily exceed 18Ψ bytes (Raffel et al., 2020; Touvron et al., 2023; Chowdhery et al., 2022).

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

Parallel and Distributed Training Techniques Researchers have developed various distributed computing techniques that leverage system-level optimizations and hardware resources to mitigate the substantial memory requirements in training LLMs.

One prominent framework is Distributed Data-Parallel (DDP) that combines data parallelism where the training dataset is partitioned across multiple devices or nodes, 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 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) enables 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. ZERO can also partition optimizer states and gradients across DDP 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 LLaMA3 (Touvron et al., 2023), GPT-3 (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. 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 PEFT techniques efficiently adapt pre-trained language models to various downstream applications without fine-tuning all the model's parameters (Ding et al., 2022), significantly reducing the computational and memory overhead.

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

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

where W_0 is a frozen full-rank pre-trained weight matrix, and $B \in \mathbb{R}^{n \times r}$ and $A \in \mathbb{R}^{r \times m}$ 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, reducing memory requirements for both parameter 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. 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., 2023), and further reducing the memory footprint (Dettmers et al., 2023). 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 storing entire 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 storing separate copies of the entire model for each task.

Despite their benefits, recent works have highlighted several limitations of low-rank reparameterization approaches. LoRA does not consistently 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 warmup before optimizing in the low-rank subspace (Lialin & Schatz, 2023). 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. By reducing the memory footprint associated with optimizer states, it is possible to maintain full-parameter learning—thus preserving model capacity and performance—while achieving significant memory savings.

The core idea behind GaLore [Ref] is to exploit the slowly changing low-rank structure of the gradient matrix $g \in \mathbb{R}^{n \times m}$, rather than approximating the weights. 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, let $g = P \Sigma Q^T$ be the the compact SVD, where $P \in \mathbb{R}^{n \times r}$ and $Q \in \mathbb{R}^{m \times r}$ are the associated semi-orthognal matrices. Then GaLore projects the gradient matrix g into a low-rank form:

$$g_{\text{low-rank}} = P^T g. \tag{2}$$

Here, $r \ll \min(n, m)$ is the target rank, n is the parameter count, m is the batch size and $g_{\text{low-rank}}$ serves as an efficient approximation of the original gradient. The projection matrix P is 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, leading to up to **30%** memory reduction compared LoRA. Moreover, GaLore maintains full-parameter learning, allowing updates to all model parameters, leading to better generalization and performance than 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 yet to match that of optimizers in full optimizer state space. Reliance on low-rank gradient approximations may not fully capture the rich optimization dynamics. 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 In this work, we propose to bridge the gap by incorporating a second-order regularizer into the low-rank gradient estimate, which adjusts parameter updates more effectively, leading to faster convergence. We show that applying the inverse of the Empirical Fisher Information Matrix (FIM) to the low-rank gradients leads to variance reduction of the gradient estimate, incorporates information about the curvature of the loss landscape, and reduces dependence on the starting point. All of these lead to significantly faster convergence, especially in a limited iteration regime.

We introduce the Natural GaLore algorithm, a matrix-free algorithm for efficiently applying the inverse FIM to the low-rank gradients, using Woodbury Identity, Cholesky Decomposition, and Matrix-Vector Products, all of which can be efficiently implemented on the GPU. Further, our approach does not require any explicit layer-wise information or significant computational overhead, as is seen in existing approaches like K-Fac, INGD, and SINGD.

We validate the effectiveness of Natural GaLore through extensive empirical evaluations. Pre-training experiments on LLaMA models with 60M, 300M, and 1.1B parameters using the C4 dataset demonstrate that Natural GaLore achieves significantly lower perplexity than GaLore without additional memory overhead, indicating faster convergence within the same computational budget.

Furthermore, we showcase the practical benefits of Natural GaLore in fine-tuning tasks. 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 surpasses LoRA and exceeds that of GPT-4-turbo, which achieves **79.08%** accuracy—all while using **30%** less memory.

2 Accelerating GaLore with Natural Gradients

2.1 Next Token Prediction Problem

Generative LLMs are trained to predict the next token in a sequence based solely on the tokens that have come before it. This ”causal” approach 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 objective is to maximize the likelihood of a 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 and $\text{Prob}_\theta(x_t \mid x_{<t})$ is the probability of the next token given all previous tokens and the parameter $\theta \in \mathbb{R}^{n\times m}$.

This is equivalent to minimizing the *negative log-likelihood (NLL)* of the observed sequences, which is 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, for LLMs the dataset is also massive, making the optimization problem very challenging.

2.2 Low Rank Gradient Descent

Stochastic gradient descent algorithms are iterative, where each step aims 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 defined using the Taylor series expansion:

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

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, precisely under the condition that the loss function can be represented in terms of KL divergence between the actual and approximated distributions 3, then $\mathbf{H_k}$ can be approximated by the FIM. The FIM is defined as the expectation of the Hessian of the negative log-likelihood w.r.t. the data distribution:

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

The FIM captures the curvature of the loss landscape and provides a natural metric for the optimization process. Hence, it can 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 it 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:

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

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), 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:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\mathbf{g}_k \tag{6}$$
$$v_k = \beta_2 v_{k-1} + (1-\beta_2)\mathbf{g}_k^2 \tag{7}$$
$$\mathbf{u_k^*} = m_k/\sqrt{v_k+\epsilon} \tag{8}$$

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 now argue, this leads to a much more favorable dependence on the starting point, which means 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 significant constant factor (Martens, 2020).

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

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 satisfying 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. The loss function is non-convex, so the property can not be stated to hold for the global optimum.

The result also relies on the computation of the exact FIM $\mathbf{F}_k(\theta_k)$ using the entire data distribution, which is not practical. The Fisher efficiency guarantee is, however, only approximately satisfied when using the empirical FIM $\hat{\mathbf{F}}_{\mathbf{k}}$ instead. However, we still get a variance reduction in the gradient estimates, leading to faster convergence and better optimization performance in the early stages of training large-scale models, making it especially valuable for training with a limited iteration budget.

Further, incorporating second-order information through the empirical FIM allows the optimizer to account for the curvature of the loss landscape, enabling natural gradient descent to take more informed steps than 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. If momentum is applied to the gradient estimator, the first term becomes independent of the choice of FIM estimator, thereby not leading to any asymptotic improvements. However, regularizing with the empirical FIM estimate can significantly reduce the constant factor associated with the starting-point-dependent second term. This leads to practical performance gains in finite iteration regimes (although negligible for large k).

Finally, the Fisher efficiency result also assumes that the model can perfectly capture the data distribution, a condition known as *realizability*. However, with the growing size of LLMs, this assumption is likely to hold, thereby satisfying the conditions for the guarantee. Therefore, especially in low-resource settings, Natural GaLore can be 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 FIM to low-rank gradients using the Woodbury Identity. Most of the steps in the algorithm are similar to GaLore, with the critical difference being the incorporation of the natural gradient transform.

In order to implement the natural gradient transform, we compute the inverse of the empirical FIM and apply it to the gradient $\mathbf{g_k}$ 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 $\hat{\mathbf{F}}_k = \lambda I + GG^T$, $A = \lambda I$, $U = G$, and $B = I$, where $G = [\mathrm{vec}(\mathbf{g}_k), \mathrm{vec}(\mathbf{g}_{k-1}), \ldots, \mathrm{vec}(\mathbf{g}_{k-s})]$ is the stacked gradient matrix over the past s 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 calculated as:

$$\tilde{\mathbf{g}}_k = \frac{1}{\lambda}\mathbf{g_k} - \frac{1}{\lambda}G\left(\lambda I + G^T G\right)^{-1} G^T \mathbf{g_k} \tag{10}$$

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

$$Sz = y \tag{11}$$

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:

$$\tilde{\mathbf{g}}_k = \frac{1}{\lambda}\mathbf{g_k} - \frac{1}{\lambda^2}Gz \tag{12}$$

This natural gradient estimate can then be sent to the Adam optimizer 8, and the model parameters the same way as in GaLore.

Table 1: Comparison of Natural GaLore with other low-rank algorithms on pre-training various sizes of LLaMA models on the C4 dataset. Validation perplexity is reported, along with a memory estimate (in gigabytes) of the total parameters and optimizer states based on BF16 format. The actual memory footprint of GaLore is reported in Figure **??**.

	60M	**130M**	**350M**	**1.1B**
Full-Rank	34.06 (0.36G)	25.08 (0.76G)	18.80 (2.06G)	15.56 (7.80G)
Natural GaLore	**34.20** (0.24G)	**25.12** (0.52G)	**18.85** (1.22G)	**15.58** (4.38G)
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)
Rank r/d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

3 EXPERIMENTS

We evaluate **Natural GaLore** on both pre-training and fine-tuning tasks for large language models (LLMs). All experiments are conducted on a single node with NVIDIA A100 GPUs to leverage high-performance computing capabilities.

3.1 PRE-TRAINING ON THE C4 DATASET

To assess the effectiveness of Natural GaLore, we apply it to pre-train LLaMA-based language models of varying sizes on the C4 dataset. The C4 dataset is a colossal, cleaned version of the Common Crawl corpus, primarily intended for pre-training language models and word representations (**?**). It provides a diverse and extensive corpus, making it suitable for evaluating pre-training methods in realistic scenarios.

In order to simulate practical pre-training conditions, we train models without data repetition over a sufficiently large amount of data, covering model sizes ranging from 60 million to 1.1 billion parameters. This approach ensures that our evaluation reflects the challenges encountered in large-scale language model training.

We adopt the experimental setup from Lialin & Schatz (2023), utilizing a LLaMA-based[3] architecture with RMSNorm and SwiGLU activations (**?**Shazeer, 2020; Touvron et al., 2023). For each model size, we maintain the same set of hyperparameters across all methods, except for the learning rate, which is tuned individually to ensure optimal performance. All experiments are conducted using the BF16 (Brain Floating Point) format to reduce memory usage without compromising computational efficiency.

To ensure a fair comparison, we allocate the same computational budget to each method and report the best validation perplexity achieved after hyperparameter tuning. Detailed descriptions of the task setups and hyperparameters are provided in the appendix.

Table 1 presents the validation perplexity and memory consumption for models trained with different methods. Our proposed **Natural GaLore** consistently outperforms GaLore across all model sizes, achieving validation perplexities closer to the Full-Rank baseline while maintaining significant memory savings. For example, on the 1.1B parameter model, Natural GaLore achieves a perplexity of **15.58** compared to GaLore's 15.64 and the Full-Rank model's 15.56, all while using substantially less memory (4.38G vs. 7.80G).

Furthermore, Natural GaLore demonstrates superior performance compared to other low-rank adaptation methods like LoRA and ReLoRA, particularly as model size increases. LoRA and ReLoRA exhibit higher perplexities and greater memory consumption due to their less efficient use of low-rank structures and the need for additional optimizer states.

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

These results highlight the effectiveness of incorporating second-order information through Natural GaLore. By efficiently applying the inverse Empirical Fisher Information Matrix to low-rank gradients, Natural GaLore improves convergence rates and achieves better performance without additional memory overhead. This makes it a compelling choice for large-scale pre-training scenarios where both memory efficiency and model performance are critical.

3.2 Fine-Tuning TinyLlama 1.1B for Function Calling in Advanced Agentic Systems

Advanced Agentic Systems (AAS) require language models that can understand and generate code snippets to integrate various tools and APIs, fulfilling user queries through function calling. We utilize the TinyAgent framework, which provides an end-to-end pipeline for training and deploying task-specific LLM agents capable of function calling (**?**), to drive agentic systems at the edge.

The primary objective is to teach language models to perform function calling effectively. Given a natural language query, the LLM agent must generate a sequence of function calls that accomplish the desired tasks. The set of possible function calls is predefined and known to the agent. The challenge lies in determining which functions to call, the appropriate arguments to pass, and the correct order of function calls, all while respecting interdependencies among the functions.

? introduced LLMCompiler, a framework that enables language models to perform function calling by first generating a *function calling plan*. This plan includes the set of functions the model needs to call along with the required arguments. The LLMCompiler then compiles this plan into an executable sequence of function calls. The critical aspect is training the model to produce a function calling plan with the correct syntax and dependencies.

In preliminary experiments reported by **?**, the off-the-shelf TinyLlama 1.1B (instruct-32k) model performed poorly on this task. The model generated incorrect sets of functions, hallucinated function names, failed to respect dependencies, and incorrectly passed arguments. This underperformance is expected, as the model was initially trained on datasets like SlimPajama and StarCoder, which are not specific to function calling tasks. To address this, we follow the TinyAgent framework and fine-tune the TinyLlama 1.1B model on a high-quality, curated dataset designed for function calling.

TinyAgent Dataset The TinyAgent dataset is a meticulously curated collection aimed at building a local agentic system for function calling on devices like Apple MacBooks for day-to-day tasks. It contains 40,000 examples of natural language queries paired with their corresponding function calling plans. The dataset is divided into 38,000 training examples, 1,000 validation examples, and 1,000 test examples. It encompasses 16 tasks, including Email, Contacts, SMS, Calendar, Notes, Reminders, File Management, Zoom Meetings, and more. Each task has predefined scripts that the model needs to generate. The dataset is intentionally challenging, requiring the model to understand dependencies between function calls and the arguments to be passed.

Fine-Tuning Procedure We fine-tune the TinyLlama 1.1B model on the TinyAgent dataset for three epochs using a batch size of 32. The learning rate is set to 7×10^{-5}. After each epoch, the model is evaluated on the validation set, and the best-performing model is selected based on validation performance to be evaluated on the test set. The training objective is to maximize the accuracy of the generated function calling plans.

Success is defined as the model generating the correct plan with the right set of function calls, correct arguments, and the appropriate order of function calls. Verifying the selection of the correct set of functions involves straightforward set comparison. However, ensuring the correctness of arguments and the order of function calls is more complex and requires constructing the associated Directed Acyclic Graph (DAG) to check for equality.

During fine-tuning, the prompt includes descriptions of the ground truth functions as well as irrelevant functions serving as negative samples. This strategy encourages the model to learn to select the correct functions rather than merely memorizing the ground truth. Additionally, several in-context examples demonstrate how queries are translated into function calling plans. These examples are selected using a Retrieval-Augmented Generation (RAG) process based on the user's query from the training data.

Results and Discussion After fine-tuning, the TinyLlama 1.1B model's success rate on the test set improved significantly. Using **Natural GaLore**, the success rate increased from 12.71% (baseline) to **83.09%**, outperforming GPT-4-Turbo by approximately 4%. In contrast, fine-tuning with the LoRA method resulted in a success rate of 78.89%. These results demonstrate the effectiveness of Natural GaLore in enhancing the performance of language models on function calling tasks within advanced agentic systems.

The substantial improvement underscores the benefit of incorporating second-order information through Natural GaLore during fine-tuning. By efficiently applying the inverse Empirical Fisher Information Matrix to low-rank gradients, Natural GaLore facilitates better convergence and optimization, leading to superior performance without additional memory overhead.

4 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 diffusion 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.

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, 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.

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.

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

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.

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.

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*, 2023. 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.