APOLLO: SGD-LIKE MEMORY, ADAMW-LEVEL PERFORMANCE

Hanqing Zhu [*1,2] **Zhenyu Zhang** [*1] **Wenyan Cong** [1] **Xi Liu** [2] **Sem Park** [2] **Vikas Chandra** [2] **Bo Long** [2]
David Z. Pan [†1] **Zhangyang Wang** [†1] **Jinwon Lee** [†2]

ABSTRACT

Large language models (LLMs) demonstrate remarkable capabilities but are notoriously memory-intensive during training, particularly with the popular AdamW optimizer. This memory burden often necessitates using more or higher-end GPUs or reducing batch sizes, limiting training scalability and throughput, respectively. To address this, various memory-efficient optimizers have been proposed to reduce optimizer memory usage. However, they face key challenges: (i) reliance on costly SVD operations (e.g., GaLore, Fira); (ii) significant performance trade-offs compared to AdamW (e.g., Flora); and (iii) still substantial memory overhead of optimization states in order to maintain competitive performance (e.g., 1/4 rank in GaLore, and full-rank first momentum in Adam-mini).

In this work, we investigate the redundancy in AdamW's learning rate adaption rule and identify that it can be coarsened as a structured learning rate update (channel-wise or tensor-wise). Based on this insight, we propose a novel approach, *Approximated Gradient Scaling for Memory Efficient LLM Optimization* (**APOLLO**), which approximate the channel-wise learning rate scaling with an auxiliary low-rank optimizer state based on pure *random projection*. The structured learning rate update rule makes APOLLO highly tolerant to further memory reduction with lower rank, halving the rank while delivering similar pre-training performance. We further propose an extreme memory-efficient version, APOLLO-Mini, which utilizes tensor-wise scaling with only a rank-1 auxiliary sub-space, achieving **SGD-level memory cost** but superior pre-training performance than Adam(W).

We conduct extensive experiments across different model architectures and tasks, showing that APOLLO series performs **generally on-par with, or even better than Adam(W)**. Meanwhile, APOLLO achieves **even greater memory savings than GaLore**, by almost eliminating the optimization states in AdamW. These savings translate into significant system benefits: (1) ***Enhanced Throughput:*** APOLLO and APOLLO-Mini achieve around 3× throughput on an 8×A100-80GB setup compared to AdamW by fully utilizing memory to support 4× larger batch sizes. (2) ***Improved Model Scalability:*** APOLLO-Mini *for the first time* enables pre-training LLaMA-13B model with naive DDP on A100-80G without requiring other system-level optimizations. (3) ***Low-End GPU Friendly Pre-training:*** Combined with quantization, the APOLLO series *for the first time* enables the training of LLaMA-7B from scratch on a single GPU using less than 12 GB of memory. Check the project page at APOLLO.

1 INTRODUCTION

Large Language Models (LLMs) have achieved remarkable progress across various domains (Brown et al., 2020; Kocoń et al., 2023; Dubey et al., 2024), largely due to substantial increases in model size, now reaching billions of parameters. Training these high-dimensional models demands robust optimization techniques, with the Adam(W) optimizer (Kingma & Ba, 2014; Loshchilov, 2017) emerging as the de-facto standard for stabilizing LLM training (Zhang et al., 2024a) by tracking both first-order and second-order moments. Despite its effectiveness, Adam(W) incurs significant memory overhead, as maintaining both moments effectively triples the memory required relative to the model's parameter size. This results in excessive memory consumption for the optimizer, even with a single batch. For instance, training a LLaMA-7B model from scratch requires at least 58 GB of memory, with 28 GB devoted to AdamW's optimizer states (Zhao et al., 2024). For larger models like GPT-3, with 175 billion parameters, memory demands reach 700 GB for the model alone, leading to a staggering 1.4 TB requirement for AdamW's optimizer states.

This excessive optimizer memory usage poses significant challenges in training large-scale LLMs. It compels the community to either use more and higher-end GPUs, or to

[*]Equal contribution [†]Co-advising to this work [1]Department of Electrical and Computer Engineering, The University of Texas at Austin [2]AI at Meta; Work was done during Hanqing's internship at Meta. Correspondence to: David Z, Pan <dpan@ece.utexas.edu>, Zhangyang Wang <atlaswang@utexas.edu>, Jinwon Lee <jinwonl@meta.com>.

Preprint.

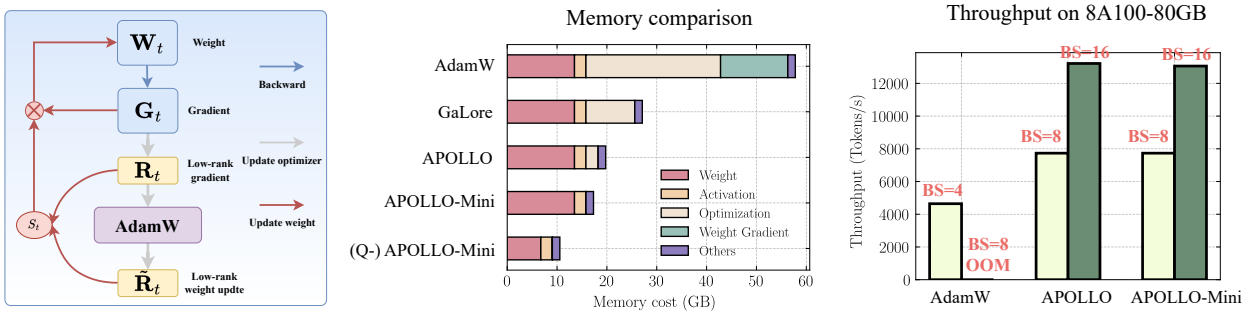

Figure 1. (Left) Overview of our APOLLO optimizer; (Middle) Memory breakdown comparison for a single batch size, where both GaLore and our method employ the layer-wise gradient update strategy (Lv et al., 2023). The (Q-) prefix indicates the integration of INT8 weight quantization, as utilized in (Zhang et al., 2024c); (Right) End-to-end training throughput on 8 A100-80GB GPUs.

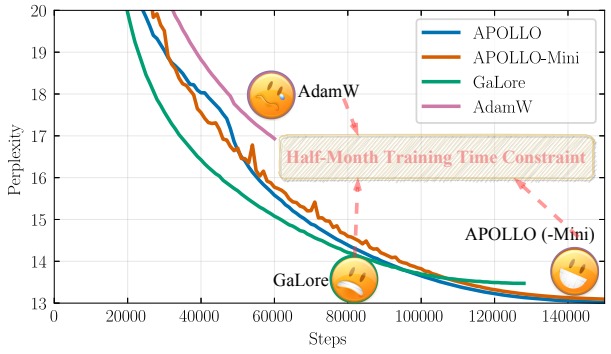

Figure 2. Comparison of Validation perplexity on LLaMA-7B.

reduce batch sizes. However, scaling training clusters introduce highly non-trivial communication and infrastructure overheads (Jiang et al., 2024); smaller batch sizes come at the cost of training throughput; and high-end GPUs are often inaccessible to researchers with limited resources.

Significant research efforts have focused on addressing the high memory costs of training LLMs. One approach involves reducing the parameter volume through methods such as designing smaller-scale LLMs (Liu et al., 2024b; Tang et al., 2024), employing sparse model training (Liu et al., 2022; Thangarasa et al., 2023), and leveraging low-rank adaptation (Hu et al., 2021). Although these techniques effectively reduce memory usage, they restrict the optimization space of model parameters and often result in performance trade-offs (Biderman et al., 2024), particularly during pretraining (Lialin et al., 2023).

Another avenue of research focuses on designing memory-efficient optimizers that reduce memory consumption while achieving performance on par with Adam(W). This includes exploring redundancy in optimizer states (Zhang et al., 2024b) and leveraging low-rank properties (Zhao et al., 2024; Chen et al., 2024). Among low-rank-based methods, GaLore (Zhao et al., 2024) stands out, enabling full-parameter training of LLMs by performing low-rank gradient updates through Singular Value Decomposition (SVD). Fira (Chen et al., 2024) enhances GaLore by incorporating the error residual between the full-rank gradient and its low-rank approximation, effectively simulating full-rank updates. LDAdam (Robert et al., 2024) also integrates a generalized error feedback mechanism to explicitly account for the compression of gradient and optimizer states.

However, the periodic updates to the gradient subspace via SVD (e.g., every 200 iterations) incurs a computational cost of $O(mn^2)$, prohibitive when the matrix dimensions, m and n, are large, as is the case with LLMs. For instance, in the case of the LLaMA-7B model, a single subspace update can take approximately 10 minutes, whereas inference only takes seconds. This substantial overhead significantly reduces training throughput, as demonstrated in Fig. 7. Several GaLore variants, therefore, explore replacing the SVD projection with online PCA (Liang et al., 2024) or random projections (He et al., 2024), yielding provable convergence.

In contrast, the recently proposed Adam-mini (Zhang et al., 2024b) demonstrates that a block-wise second moment $\mathbf{V}$ suffices for learning rate adjustments, offering an orthogonal and more efficient alternative. However, achieving performance on par with AdamW requires careful handling of different model components to maintain compatibility with its optimization dynamics.

In this paper, we effectively integrate the two idea streams of *low-rank approximation* and *optimizer state redundancy*, introducing a unified framework that achieves significant memory savings (below GaLore and its variants & close to SGD) while maintaining or surpassing the performance of Adam(W). Our **key observation** is that AdamW's element-wise learning rate update rule can be effectively restructured into a channel-wise or even tensor-wise format, where each channel or tensor shares the same gradient scaling factor. To enable this structured gradient scaling, we introduce a memory-efficient approximation for the scaling factors using an auxiliary optimizer state, requiring only lower-dimensional gradient information as input. This signifi-

cantly reduces memory usage by leveraging a compressed representation of the gradient information. Additionally, we eliminate the need for costly SVD-based low-rank projections by adopting an **SVD-free** method based on random projections. We show that a much lower rank, or even a **rank-1 approximation**, is sufficient to capture the structured gradient scaling factors effectively. This innovation allows for a simpler and more efficient training process without compromising performance. Our new memory-efficient optimizer for LLM training, named *Approximated Gradient Scaling for Memory Efficient LLM Optimization* (**APOLLO**), not only achieves better performance than AdamW but also delivers greater memory savings compared to GaLore.

Our key contributions are as follows:

- **Structured Learning Rate Update for LLM Training**: We show that structured learning rate updates, such as channel-wise or tensor-wise scaling, are sufficient for LLM training. This addresses redundancy in AdamW's element-wise learning rate update rule and reduces computational overhead.
- **Approximated Channel-wise Gradient Scaling in a Low-Rank Auxiliary Space (APOLLO)**: We propose a practical and memory-efficient method to approximate channel-wise gradient scaling factors in an auxiliary low-rank space using pure random projections. APOLLO achieves superior performance to AdamW, even with lower-rank approximations, while maintaining excellent memory efficiency.
- **Minimal-Rank Tensor-wise Gradient Scaling (APOLLO-Mini)**: For extreme memory efficiency, we introduce APOLLO-Mini, which applies tensor-wise gradient scaling using only a rank-1 auxiliary sub-space. APOLLO-Mini achieves SGD-level memory costs while outperforming AdamW, demonstrating the effectiveness of our approach.

We demonstrate the efficacy of the APOLLO series in both pre-training and fine-tuning scenarios. In pre-training, across a range of LLaMA model sizes from 60M to 7B parameters, APOLLO and APOLLO-Mini consistently outperform AdamW, achieving up to a 2.8 reduction in validation perplexity while significantly reducing memory overhead by eliminating nearly all optimizer states. In fine-tuning, APOLLO and APOLLO-Mini achieve performance on par with full fine-tuning. Beyond these performance gains, the APOLLO series offers practical **system-level** advantages, including: (i) **3× better throughput** on pre-training LLaMA 7B (Fig.1 (right) and the 7B experiments in Fig.2); (2) **Extreme training memory savings**. By combining APOLLO-Mini with weight quantization, we **set a new record for memory efficiency**: pre-training a LLaMA 7B model requires only **12GB** of memory (Fig. 1 (middle)). More information can be found in Section 5.3.

These results establish APOLLO and APOLLO-Mini as highly efficient and scalable solutions for both pre-training and fine-tuning of LLMs, offering compelling improvements in performance, memory usage, and throughput.

2 Related Work

2.1 Algorithm-Level Memory-Efficient Training

Numerous algorithmic improvements have been introduced to tackle the substantial memory overhead in training LLMs. One category focuses on reducing the number of trainable parameters to save memory costs. This includes approaches such as developing high-quality, small-scale models (Liu et al., 2024b; Tang et al., 2024), introducing sparsity during training (Liu et al., 2022; Thangarasa et al., 2023), and implementing low-rank adaptation (Hu et al., 2021). While these methods are effective at reducing memory usage, they often fall short in achieving comparable performance, especially in pre-training scenarios for large models.

Another avenue of research targets advancements in optimizers, as exemplified by works such as GaLore (Zhao et al., 2024), Fira (Chen et al., 2024), Flora (Hao et al., 2024), Adam-mini (Zhang et al., 2024b), GaLore-mini (Huang et al.), LDAdam (Robert et al., 2024), GoLore (He et al., 2024), and LoQT (Loeschcke et al.). These approaches have made notable progress but still face significant challenges. Some methods rely on computationally expensive SVD operations (e.g., GaLore and Fira), although recent research shows that random projections can effectively compress gradients during later training stages while still requiring SVD early on (He et al., 2024). Others either exhibit noticeable performance gaps compared to AdamW, or demand substantial memory overhead to maintain competitive performance, as seen in GaLore's 1/4 rank requirement and Adam-mini's reliance on full-rank first momentum.

In contrast, APOLLO achieves efficient memory usage entirely without relying on SVD while delivering performance that matches or even surpasses AdamW. Moreover, our extreme variant, APOLLO-Mini, drives memory costs down to SGD levels while maintaining or exceeding the performance of AdamW, setting a new benchmark for memory-efficient optimization.

2.2 System-Level Memory Efficiency Optimization

Several system-level techniques have been developed to reduce memory usage in LLM training (Chen et al., 2016; Ren et al., 2021). Activation checkpointing (Chen et al., 2016) recomputes activations during backward instead of storing them for the whole training iteration, reducing memory requirements. Quantization (Dettmers et al., 2024) reduces memory requirements by utilizing lower bit data formats. Memory offloading (Zhang et al., 2023; Ren et al., 2021) reduces GPU memory consumption by leveraging non-GPU

memory. Our method, APOLLO, is orthogonal to these system-level optimizations and can be seamlessly integrated to achieve greater memory efficiency. Furthermore, by eliminating the need for SVD, APOLLO is more system-friendly, requiring only a cheap general matrix multiplication to complete the projection step.

3 Coarsened Learning Rate Update Rule Is Enough for LLMs

In this section, we first revisit the Adam(W) (Kingma & Ba, 2014; Loshchilov, 2017) and reformulate it as an adaptive learning rate algorithm without explicit momentum term (Section 3.1). Then, we propose that the element-wise learning rate update rule can be coarsened with a structured channel-wise learning rate adaptation strategy, with even slightly better model performance by empirical verification.

3.1 Reformulating AdamW as a Pure Adaptive Learning Rate Algorithm

Vanilla AdamW update rule. AdamW has established itself as the go-to optimizer for Transformer training, leveraging both **first moment** (the mean of past gradients) and **second moment** (the variance of past gradients) to adjust updates. This dual momentum-based approach has proven superior to purely first-order optimizers like SGD (Zhang et al., 2024a). Disregarding weight decay, the vanilla AdamW update rule is as follows:

At time step t, given a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ $(m \leq n)$ with gradient $\mathbf{G}_t = -\nabla_{\mathbf{W}}\phi_t(\mathbf{W}_t)$, the standard AdamW update rule is defined as:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \cdot \tilde{\mathbf{G}}_t, \quad \tilde{\mathbf{G}}_t = \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t} + \epsilon} \tag{1}$$

Here, η is the learning rate and ϵ is a small constant for numerical stability. The first and second moment, $\mathbf{M}_t$ and $\mathbf{V}_t$, are computed as exponentially weighted averages:

$$\mathbf{M}_t = \beta_1 \mathbf{M}_{t-1} + (1-\beta_1)\mathbf{G}_t$$

$$\mathbf{V}_t = \beta_2 \mathbf{V}_{t-1} + (1-\beta_2)\mathbf{G}_t^2$$

where $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates.

Viewing AdamW as an adaptive learning rate algorithm without momentum. The above update rule in equation 1 can then be reformulate as an element-wise **gradient scaling rule** with a gradient scaling factor $\mathbf{S} = \frac{\tilde{\mathbf{G}}_t}{\mathbf{G}_t} \in \mathbb{R}^{m\times n}$ over the raw gradient $\mathbf{G}_t$, *i.e.*,

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \cdot \frac{\tilde{\mathbf{G}}_t}{\mathbf{G}_t} \cdot \mathbf{G}_t \tag{2}$$

In other words, the effectiveness of AdamW can be viewed as the result of a **variance-aware learning rate schedule** per element in raw gradient $\mathbf{G}_t$ using the corresponding element in $\mathbf{S}$, where elements with higher variance in $\mathbf{V}_t$ are scaled down to reduce unstable updates. While this reformulation is very straightforward, it paves the way for subsequent analysis. It also provides a convenient strategy to analyze other momentum algorithms through "SGD-like" lens (e.g., all reduced to adaptive SGD learning rates).

3.2 Coarsening Element-wise Learning Rate Adjustment in a Structured Manner

While the element-wise learning rate update rule in AdamW is effective, it can be **overly sensitive to noisy gradients** in specific parameters, especially in high-dimensional models like large language models (LLMs). Recent work, such as Adam-mini (Zhang et al., 2024b), proposes to grouping parameters into blocks and applying a **block-wise learning rate adjustment** to reduce memory usage while maintaining on-par performance as Adam. However, the block-wise approach in Adam-mini (Zhang et al., 2024b) requires carefully chosen block sizes for different modules in Transformers and only achieves memory savings for the second moments, leaving the first moment memory unaffected.

A more structured learning rate update rule. Inspired by these findings, we propose an effective simplification by coarsening the element-wise adaptive learning rate rule in equation 2 into a **structured channel-wise adaptation**. We group parameters based on the larger dimension of the weight tensors. The element-wise scaling factor $\mathbf{S} = \frac{\tilde{\mathbf{G}}_t}{\mathbf{G}_t}$ is then simplified into a **channel-wise** format, $s \in \mathbb{R}^{1\times n}$, where each element s_j for each channel j is:

$$s_j = \frac{\|\tilde{\mathbf{G}}_t[:, j]\|_2}{\|\mathbf{G}_t[:, j]\|_2} \tag{3}$$

where $\|\cdot\|_2$ denotes the ℓ_2 norm. Then, the final gradient scaling rule becomes $\tilde{\mathbf{G}}_t = \mathbf{S} \cdot \mathbf{G}_t = \mathbf{G}_t \cdot \mathrm{diag}(s)$.

Empirical validation. We first empirically explore the effectiveness of the proposed update rule where we compared the training loss of the original element-wise learning rate adaptation with our proposed channel-wise adaptation on a LLaMA-130M model. As shown in Figure 3, both approaches achieve similar loss reduction over training steps, demonstrating that the structured adaptation effectively maintains performance. In fact, the channel-wise adaptation achieves slightly better perplexity 24.43 (AdamW: 25.08), further supporting the effectiveness of our approach. However, we notice that our channel-wise learning rate adaption (orange curve) shows a significant spike at the early stage, which is due to the unstable gradient at the early stage. Instead of applying the vanilla gradient clipping method, we

use the Norm-growth Limiter (NL) in (Chen et al., 2024) to limit the consecutive gradient growth, as it is shown slightly more effective than gradient clipping:

$$\text{if } \frac{||\tilde{\mathbf{G}}_t||}{||\tilde{\mathbf{G}}_{t-1}||} > \gamma \text{ then } \tilde{\mathbf{G}}_t \leftarrow \frac{\tilde{\mathbf{G}}_t}{||\tilde{\mathbf{G}}_t||} \cdot \gamma ||\tilde{\mathbf{G}}_{t-1}|| \quad (4)$$

where γ is a threshold to ensure that the rate of gradient growth remains controlled. This approach limits the magnitude of gradient norm increases, particularly for the unstable gradients in the early stages, thereby preventing loss spikes (green curve), leading to further better perplexity 24.11. We, by default, use the NL in our method and set $\gamma = 1.01$.

Takeaways ①: A structured learning rate update is sufficient for LLM training.

This observation suggests that effective optimization can be achieved by applying adaptive learning rates at a coarser granularity, such as channel-wise, rather than at the element-wise level. This insight forms the basis for the memory-efficient methods we propose in the next section.

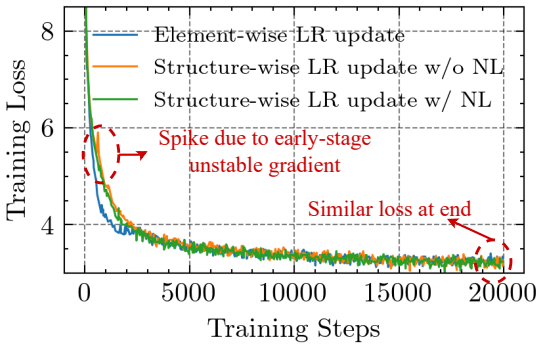

Figure 3. Training loss comparison between Element-wise and Channel-wise Learning Rate (LR) Adaptations with or without norm limiter (NL) on the LLaMA-130M model.

4 APOLLO: Approximated Gradient Scaling for Memory Efficient LLM Optimization

From observation to practical Benefit. While coarsening gradient scaling factors is effective, it does not inherently reduce optimizer memory usage. Computing structured gradient scaling factors still requires access to the full optimizer states $\mathbf{M}_t$ and $\mathbf{V}_t$, which consume significant memory. This brings us to a critical question:

Question ①: Can structured learning rate adaptation be converted into practical, memory-efficient optimization?

4.1 APOLLO: Approximate Structural Gradient Scaling for LLM Optimization

Algorithm 1 AdamW with APOLLO/APOLLO-Mini

Input: A weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ with $m \leq n$. Step size η, scale factor α, decay rates $\{\beta_1, \beta_2\}$, weight decay λ, rank r, subspace update frequency T.
Initialize: $t \leftarrow 0$
repeat
  # Step 1: Calculate gradient into low rank space.
  $\mathbf{G}_t \in \mathbb{R}^{m \times n} \leftarrow -\nabla_{\mathbf{W}} \phi_t(\mathbf{W}_t)$
  if $t \bmod T = 0$ **then**
    $\mathbf{P}_t \leftarrow \mathcal{N}_{seed}(0, 1/r)$
    seed $\leftarrow$ an independent new random seed
  end if
  $\mathbf{R}_t \leftarrow \mathbf{P}_t \mathbf{G}_t$
  # Step 2: Obtain low rank optimization states, $\mathbf{M}_t, \mathbf{V}_t$.
  $\mathbf{M}_t^R, \mathbf{V}_t^R \leftarrow \text{AdamW}(\mathbf{R}_t, \beta_1, \beta_2, \lambda)$
  $\tilde{\mathbf{R}}_t \leftarrow \mathbf{M}_t^R / (\sqrt{\mathbf{V}_t^R} + \epsilon)$
  # Step 3: Update weight in original space.
  if APOLLO **then**
    $\mathbf{S} \leftarrow \text{diag}(s_0^R, s_1^R, ..., s_m^R)$ $\{s_i^R = \frac{\|\tilde{\mathbf{R}}_t[:,j]\|_2}{\|\mathbf{R}_t[:,j]\|_2}\}$
  else if APOLLO-Mini **then**
    $\mathbf{S} \leftarrow s^R$ $\{s^R = \frac{\|\tilde{\mathbf{R}}_t\|_2}{\|\mathbf{R}_t\|_2}\}$
  end if
  $\mathbf{W}_t \leftarrow \mathbf{W}_{t-1} + \eta \cdot \alpha \cdot \mathbf{G}_t \mathbf{S}$
  $t \leftarrow t + 1$
until convergence criteria met
return $\mathbf{W}_T$

4.1.1 Approximating Gradient Scaling with an Auxiliary Low-Rank Space

To address this challenge, we propose APOLLO, which approximates the channel-wise gradient scaling in a compressed low-rank space rather than the original full-rank one, showing in Algorithm 1. Specifically, an auxiliary low-rank optimizer state is stored by taking the low-rank gradient $\mathbf{R}_t$ as input, which is computed as $\mathbf{R}_t = \mathbf{P}_t \mathbf{G}_t \in \mathbb{R}^{r \times n}$, using a pre-defined projection matrix $\mathbf{P}_t \in \mathbb{R}^{r \times m}$.

The auxiliary optimizer state will only maintain the low-rank version of the first and second moments as:

$$\mathbf{M}_t^R = \beta_1 \mathbf{M}_{t-1}^R + (1 - \beta_1) \mathbf{R}_t$$

$$\mathbf{V}_t^R = \beta_2 \mathbf{V}_{t-1}^R + (1 - \beta_2) \mathbf{R}_t^2$$

These low-rank moments, $\mathbf{M}_t^R$ and $\mathbf{V}_t^R$, are then converted into a lightweight, channel-wise scaling factors:

$$s_j^R = \frac{\|\tilde{\mathbf{R}}_t[:,j]\|_2}{\|\mathbf{R}_t[:,j]\|_2}, \text{where } \tilde{\mathbf{R}}_t = \frac{\mathbf{M}_t^R}{\sqrt{\mathbf{V}_t^R} + \epsilon} \quad (5)$$

In this way, APOLLO estimates the channel-wise gradient scaling factor s with the auxiliary low-rank optimizer state, saving memory from $2mn$ to $2nr$. We will show later that

the coarse-grained channel-wise scaling makes APOLLO insensitive to low rank, unlike GaLore, which needs relatively high rank to retain performances (typically, one-quarter of the original dimension), leading to great memory saving. Details can be found at Section 5.4, *A3*.

However, since APOLLO operates in a compressed domain (*i.e.* low-rank space), a key question remains:

Question ②: Can the adaptive learning rate in the compressed space effectively approximate its behavior in the original space?

Moreover, what type of low-rank projection method is ideal for this purpose? The default choice might be Singular Value Decomposition (SVD), as it captures the most informative components of the gradient. In fact, most existing low-rank optimizers for LLMs rely on SVD-based approximations to maintain accuracy, especially during pre-training. However, SVD is computationally expensive for large models and cannot be efficiently parallelized on GPUs, hindering the training process. Therefore, we pose the following question:

Question ③: Do we still need costly SVD to construct our compressed space?

4.1.2 APOLLO Performs Well with Random Projection: SVD is Not Necessary.

To answer the above questions, we first analyze the norm difference between the first moment $\mathbf{M}_t^R$ in the *compressed space* and the *original space*, as well as similar results for the second state $\mathbf{V}_t^R$ and $\mathbf{V}_t$.

We demonstrate that *random projection can effectively bound the difference between the gradient scaling factor* in the *compact* and *original* space in equation 6:

$$\begin{aligned} \textit{Original space: } s_j &= \frac{\|\tilde{\mathbf{G}}_t[:,j]\|}{\|\mathbf{G}_t[:,j]\|},\ \tilde{\mathbf{G}}_t = \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}} \\ \textit{Compact space: } s_j^R &= \frac{\|\tilde{\mathbf{R}}_t^R[:,j]\|}{\|\mathbf{R}_t[:,j]\|},\ \tilde{\mathbf{R}}_t^R = \frac{\mathbf{M}_t^R}{\sqrt{\mathbf{V}_t^R}} \end{aligned} \tag{6}$$

with all small ϵ in the denominators removed for simplicity.

Generating random projection matrix. We generate the random projection matrix $\mathbf{P}$ by sampling each element from a standard Gaussian distribution. With high probability, projection using a random Gaussian matrix largely preserves the scaled norm from the original space based on the Johnson–Lindenstrauss lemma (JLT) (Freksen, 2021).

First-order moment ratio bounding. We expand the computation formula of the first moment recursively, as:

$$\begin{aligned} \mathbf{M}_t &= \beta_1 \mathbf{M}_{t-1} + (1-\beta_1)\mathbf{G}_t \\ &= \beta_1^t \mathbf{M}_0 + (1-\beta_1)\sum_{k=0}^{t-1} \beta_1^k \mathbf{G}_{t-k} \end{aligned} \tag{7}$$

$$\begin{aligned} \mathbf{M}_t^R &= \beta_1 \mathbf{M}_{t-1}^R + (1-\beta_1)\mathbf{R}_t \\ &= \beta_1^t \mathbf{M}_0^R + (1-\beta_1)\sum_{k=0}^{t-1} \beta_1^k \mathbf{R}_{t-k} \end{aligned} \tag{8}$$

where $\beta_1 < 1$.

We quantify the approximation error in the following theorem.

Theorem 4.1. ***Approximated Channel-wise Momentum with a bound for its ℓ_2 norm**: $\boldsymbol{G}_t \in \mathbb{R}^{m\times n}$ is the full-rank gradient ($m \le n$). Let $\boldsymbol{P}$ be a matrix of shape $\mathbb{R}^{r\times m}$ where each element is independently sampled from a standard Gaussian distribution in the variance of $1/r$. With the projected gradient $\boldsymbol{R}_t = \boldsymbol{P}\boldsymbol{G}_t$, we have the projected gradient with a **bounded channel-wise first order moment**. For any channel j, with probability at least $1 - 2\exp\left(-\frac{r\epsilon^2}{8}\right)$:*

$$(1-\epsilon)\|\mathbf{M}_t[:,j]\|^2 \le \|\mathbf{M}_t^R[:,j]\|^2 \le (1+\epsilon)\|\mathbf{M}_t[:,j]\|^2,$$

.

Proof: Please refer to Appendix A.1.3.

Second-order moment ratio bounding. Similarly, the second-order moment state can be formulated as:

$$\mathbf{V}_t = (1-\beta_2)\sum_{k=0}^{t-1} \beta_2^k \mathbf{G}_{t-k}^2$$

$$\mathbf{V}_t^R = (1-\beta_2)\sum_{k=0}^{t-1} \beta_2^k \mathbf{R}_{t-k}^2$$

where we assume $\mathbf{V}_0 = 0$ (common in most initialization).

Theorem 4.2. ***Approximated channel-wise variance with a bound for its ℓ_1 norm**: For any channel j and time t, if*

$$r \ge \frac{8}{\epsilon^2}\log\left(\frac{2t}{\delta}\right),$$

then with probability at least $1 - \delta/2$:

$$(1-\epsilon)\|\mathbf{V}_t[:,j]\|_1 \le \|\mathbf{V}_t^R[:,j]\|_1 \le (1+\epsilon)\|\mathbf{V}_t[:,j]\|_1$$

, where $\mathbf{V}_t[:,j]$ and $\mathbf{V}_t^R[:,j]$ are the second moments in the original and projected spaces, respectively.

Proof: Please refer to Appendix A.1.4.

Bounded update ratio s^R/s Now, we can bound the difference between the gradient scaling factor in the compact original space based on the theorem 4.1 and theorem 4.2:

$$s_j^R/s_j = \frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\mathbf{R}_t[:,j]\|} \cdot \frac{\|\mathbf{G}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|} = \frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|} \cdot \frac{\|\mathbf{G}_t[:,j]\|}{\|\mathbf{R}_t[:,j]\|}$$

For any channel j, with probability $\geq 1-\delta$:

$$\frac{\sqrt{1-\epsilon}}{1+\epsilon} \leq \sqrt{\frac{n}{r}}\frac{s_j^R}{s_j} \leq \frac{\sqrt{1+\epsilon}}{1-\epsilon}. \tag{9}$$

Proof: Please refer to Appendix A.1.5.

Therefore, our APOLLO method is theoretically sound with random projection and validated by the empirical results presented in the following section. Additionally, APOLLO with SVD also performs well within our framework, yet incurs significant computational cost. We default to use random projection in APOLLO.

The theorem suggests we should scale the gradient with the factor $\sqrt{\frac{n}{r}}$ to ensure consistent behavior as AdamW with structured learning rate update. Hence, we add the gradient scale factor α in Algorithm 1. However, this gradient scale factor can be combined with the learning rate, therefore set to 1 by default for APOLLO. When the r is too small compared to n, as in our APOLLO-Mini case, which uses rank-1 space, we specifically assign the scaling factor by using $\sqrt{128}$.

Moreover, as suggested in GaLore, fixing the projection matrix is not ideal for high-dimensional LLM training; we also re-sample the projection matrix every T step, which is effortlessly done by re-generating a new seed, which is set to 200 by default.

Take-away ②: APOLLO can approximate the structured learning rate adaption with only random projection.

4.2 APOLLO-Mini: Achieve Extreme Memory Efficiency with Rank-1 Space

The rank r plays a crucial role in balancing memory efficiency and the quality of the approximated gradient scaling factor. Although the coarsening learning rate update rule provides high tolerance to relatively low rank, we show our APOLLO can reduce rank by half compared to GaLore with a slight impact on perplexity. We still need to pay $n \times r$ memory cost for the optimizer state. If we can relax the rank requirement to 1, then the optimizer state cost is totally negligible. However, simply setting rank to 1 in APOLLO using channel-wise gradient scaling doesn't work well due to rank-1 space sacrificing too much information. Details at Section 5.4 *A2*. This leads to our next question:

Question ④: "Can we further compress the optimizer state to SGD-level memory cost while matching or surpassing AdamW's performance?

To address this, we introduce an extremely memory-efficient APOLLO variant, called APOLLO-Mini, which coarsens the scaling factor into a *tensor-wise scaling factor* to reduce variance during gradient scaling estimation in a rank-1 space. The scaling factor is computed as: $s = \frac{\|\tilde{R}_t\|_2}{\|R_t\|_2}$. Moreover, we find the *tensor-wise scaling factor* estimated by rank-1 space is typically smaller than the one estimated with a larger rank, which can be theoretically justified by the theorem in equation 9. Hence, we heuristically set a gradient scale factor α like GaLore to avoid performance degradation, default to set as $\sqrt{128}$.

4.3 Savings and cost analysis

Table 1 provides a detailed comparison of memory and computational trade-offs among various memory-efficient training methods, including APOLLO-Mini, APOLLO, Fira (Chen et al., 2024), GaLore (Zhao et al., 2024), and Flora (Hao et al., 2024). Notably, APOLLO can purely implemented with random projection, thereby avoiding the costly SVD operations required by Fira and GaLore.

Table 1. Detailed comparison between Fira (Chen et al., 2024), GaLore (Zhao et al., 2024), Flora (Hao et al., 2024), APOLLO, and APOLLO-Mini. Denote $\mathbf{W}_t \in \mathbb{R}^{m\times n}$ ($m \leq n$), rank r. APOLLO series has a constant 2 due to storing random seed and gradient norm used for norm-worth limiter.

	APOLLO-Mini	APOLLO	Fira	GaLore	Flora
Weights	mn	mn	mn	mn	mn
Optimizer States	$2n+2$	$2nr+2$	$mr+2nr+1$	$mr+2nr$	$2nr+1$
Full-Rank Gradients	✔	✔	✔	✘	✘
Full-Rank Weights	✔	✔	✔	✔	✔
Pre-Training	✔	✔	✔	✔	✘
Fine-Tuning	✔	✔	✔	✔	✔
w.o. SVD	✔	✔	✘	✘	✔

In terms of memory efficiency, random projection allows APOLLO to eliminate the need to store a projection matrix, achieving significant memory efficiency. Additionally, APOLLO is robust to rank reduction; halving the rank has minimal impact on pre-training performance (see Table 2). Our extreme variant, APOLLO-Mini, further maximizes memory efficiency by reducing optimizer state costs to a const number $2n+2$, making it comparable to the memory footprint of SGD, yet it retains or even surpass AdamW-level performance, as confirmed in the following experiments.

5 Experiments

In Section 5.1 and 5.2, we systematically evaluate APOLLO on various pre-training and downstream tasks, respectively. Section 5.3 compares the memory overhead and throughput

Table 2. Comparison of pretraining perplexity across various memory-efficient training approaches. We pretrain the LLaMA models with model size ranging from 60M to 1B on the C4 (Raffel et al., 2020) dataset and report the validation perplexity. The memory overhead focus solely on weights and optimization states. Results marked with ♠ are collected from (Zhao et al., 2024). By default, we set the rank to one-quarter of the original dimension for all low-rank-based training approaches, while results marked with † indicate the use of a halved rank, *i.e.*, one-eighth of the original dimension.

Methods	60M		130M		350M		1B	
	Perplexity	Memory	Perplexity	Memory	Perplexity	Memory	Perplexity	Memory
AdamW♠	34.06	0.36G	25.08	0.76G	18.80	2.06G	15.56	7.80G
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
GaLore♠	34.88	0.24G	25.36	0.52G	18.95	1.22G	15.64	4.38G
Fira	**31.06**	0.24G	**22.73**	0.52G	17.03	1.22G	14.31	4.38G
APOLLO *w.* SVD	31.26	0.24G	22.84	0.52G	**16.67**	1.22G	**14.10**	4.38G
APOLLO	31.55	0.24G	22.94	0.52G	**16.85**	1.22G	**14.20**	4.38G
APOLLO†	31.26	0.18G	23.18	0.39G	**16.98**	0.95G	**14.25**	3.49G
APOLLO-Mini	31.93	0.12G	23.84	0.25G	17.18	0.69G	**14.17**	2.60G

of different approaches, while Section 5.4 presents extensive ablation studies that analyze the impact of low-rank projection methods, rank quantity, scaling factor granularity, and provide a detailed comparison of training curves.

5.1 Memory-Efficient Pre-training with APOLLO

We demonstrate that APOLLO achieves **superior pre-training performance** across various sizes of LLaMA models (60M to 7B) on the C4 dataset (Raffel et al., 2020), with up to a 2.80 reduction in validation perplexity. Additionally, APOLLO-Mini uses negligible memory budget for optimization states while outperforming both AdamW (Loshchilov, 2017) and GaLore (Zhao et al., 2024).

Setup. We consider the LLaMA series models for pre-training, with sizes ranging from 60M to 7B. Following the training configurations used in prior works (Zhao et al., 2024; Lialin et al., 2023), we pre-train each model from scratch, with a detailed description in Appendix.A.4. The C4 dataset (Raffel et al., 2020), a comprehensive corpus derived from Common Crawl data and meticulously filtered and cleaned, is used for pre-training. All experiments are conducted in BF16 data format without other quantization.

Baselines. For comparative analysis, we include the following baselines in our evaluation: (i) AdamW: We pre-train the models using the original AdamW optimizer (Loshchilov & Hutter, 2019), maintaining the weights, gradients, and optimizer states in their full-rank format. (ii) Low-Rank: This approach decomposes the model weights into two low-rank matrices ($W = UV$), with both U and V optimized using AdamW. (iii) LoRA: LoRA (Hu et al., 2021) employs low-rank adapters for memory-efficient training by decomposing the original weights as $W = W_0 + UV$. During training, only U and V are optimized with AdamW, while the backbone weights W_0 remain frozen. (iv) ReLoRA: ReLoRA (Lialin et al., 2023) is an enhanced version of LoRA specifically designed for pre-training, where the low-rank adapters UV are periodically merged back into the original weights W during training. (v) GaLore: GaLore (Zhao et al., 2024) projects gradients, rather than weights, into a low-rank space, effectively reducing memory consumption for optimizer states. (vi) Fira: Fira (Chen et al., 2024) further improves GaLore by incorporating the error residual of low-rank gradient into training.

Main Results. We evaluate APOLLO and its two variants: APOLLO *w.* SVD, which replaces the original random projection with SVD; and APOLLO-Mini, which uses a rank of 1 and computes the scaling factor in a tensor-wise manner. The memory cost of optimization states in APOLLO-Mini is negligible, as the rank used is substantially smaller than the original dimension, *e.g.*, 1 *vs.* 512 for LLaMA-60M and 1 *vs.* 2048 for LLaMA-1B. Results are reported in Table 2, from which several observations can be made: (i) **Performance under the same memory budget**: When the rank is set to one-quarter of the original dimension, APOLLO consistently outperforms GaLore, achieving up to a 3.33 reduction in perplexity; (ii) **Comparison with full-rank AdamW**: APOLLO demonstrates superior performance while using significantly less memory. Notably, APOLLO-Mini incurs a memory cost similar to that of SGD while significantly outperforming AdamW and vanilla SGD is known to fail on training transformer-based models (Zhang et al., 2024a); (iii) **Robustness across projection methods and rank sizes**: APOLLO exhibits robust performance under various subspace projection methods and rank sizes. For instance, with LLaMA-350M, enabling SVD

Table 3. Pre-training LLaMA 7B on C4 dataset for 150K steps. Validation perplexity and memory estimate (optimization states only) are reported. Results marked with ♠ are collected from (Zhao et al., 2024). APOLLO uses the rank of 256, and APOLLO-Mini uses the rank of 1.

	Optimizer Memory	40K	80K	120K	150K
8-bit Adam♠	13G	18.09	15.47	14.83	14.61
8-bit GaLore♠	4.9G	17.94	15.39	14.95	14.65
APOLLO	**3.2G**	17.55	14.39	13.23	**13.02**
APOLLO-Mini	**0.0G**	18.03	14.60	13.32	**13.09**
Tokens (B)		5.2	10.5	15.7	19.7

projection results in only a 0.18 improvement of perplexity, while SVD is known for its time-consuming nature (Zhang et al., 2024c). This indicates that APOLLO can maintain efficiency even without SVD, improving end-to-end throughput. Additionally, halving the rank has negligible impact on APOLLO 's performance, further demonstrating its effectiveness across different rank budgets. Section 5.4 provide a more thorough analysis of the impact of rank quantity and projection methods; (iv) **Comparison with Fira**: APOLLO is more favorable when dealing with larger models and more training tokens. For smaller models like 60M and 130M, Fira shows slightly better performance, but APOLLO consistently surpasses Fira as model size and training tokens increase. An in-depth comparison of training performance across model sizes and training tokens is provided in Section 5.4, *A4*. The above results validate the effectiveness of APOLLO on pre-training tasks, demonstrating that it achieves superior performance while requiring negligible memory costs for optimization states compared to AdamW.

Scaling up to Pre-training LLaMA-7B. We evaluate the pre-training of a LLaMA-7B model using AdamW, GaLore, and our APOLLO series (APOLLO ($r = 256$) and APOLLO-Mini ($r = 1$)) on an 8× A100 80GB setup. To ensure performance consistency, a total batch size of 512 per epoch is maintained, while micro-batch sizes are adjusted based on the memory consumption of each method. AdamW is limited to a micro-batch size of 4 due to memory constraints, whereas GaLore matches the memory usage of our methods with a micro-batch size of 8. Considering the extended training duration typically required for fully training a LLaMA-7B model with AdamW, we allocate a fixed training time budget of around two weeks (**15 Days**) for a fair and practical comparison. The training curve, showing validation perplexity recorded every 1000 steps, is presented in Fig. 2.

Our experiments reveal two key benefits of the APOLLO series on the large 7B model: (i) **Accelerated training through saved memory, enabling larger batch sizes.** The significant memory efficiency of the APOLLO series allows for larger batch sizes, resulting in ∼3× faster training throughput compared to AdamW and ∼2× faster throughput compared to GaLore. Notably, our methods are the only ones to complete the pre-training within the half a month timeframe. (ii) **Superior model performance with best perplexity and reduced optimizer overhead.** Despite its efficiency, APOLLO delivers better perplexity results than GaLore, even when GaLore employs a high rank of 1024. Midway through training, APOLLO surpasses GaLore in performance, marking a critical intersection point where our approach demonstrates clear superiority. Furthermore, as shown in Tab. 3, the APOLLO series achieves >1.5 perplexity improvement compared to the 8-bit versions of Adam and GaLore, all while maintaining significantly lower memory usage for optimizer states.

These findings underscore that the APOLLO series accelerates training by optimizing memory usage, supports larger batch sizes for improved throughput, enhances model quality, and provides a highly practical solution for pre-training large language models efficiently.

5.2 Memory-Efficient Fine-tuning with APOLLO

Pre-training large foundation models typically demands thousands of GPUs and months of training, making it feasible only for large organizations. As a result, fine-tuning these models has become a more practical approach among engineers and researchers. Here, we thoroughly evaluate the performance of APOLLO in fine-tuning scenarios.

Setup. We employ three open-source pre-trained models in the fine-tuning experiments, including LLaMA-3.2-1B, LLaMA-3-8B (AI@Meta, 2024), Gemma-7B and Mistral-7B (Jiang et al., 2023). The downstream tasks are divided into two categories: (i) Eight common-sense reasoning tasks: Winogrande (WG)(Sakaguchi et al., 2021), PIQA(Bisk et al., 2020), SIQA (Sap et al., 2019), OpenBookQA (OBQA)(Mihaylov et al., 2018), HellaSwag (HS)(Zellers et al., 2019), BoolQ (Clark et al., 2019), and ARC (ARC-Easy and ARC-Challenge) (Clark et al., 2018); (ii) MMLU (Hendrycks et al., 2020) tasks across various domains: STEM, Social Sciences, Humanities and others.

Baseline. We compare APOLLO against several baselines used during the pre-training experiments, including Full-rank AdamW (Loshchilov, 2017), LoRA (Hu et al., 2021), GaLore (Zhao et al., 2024), and Fira (Chen et al., 2024). Details of these baselines are summarized in Section 5.1. Additionally, we include DoRA (Liu et al., 2024a), an effective fine-tuning approach. We set the rank to 32 and 8 for common-sense reasoning and MMLU tasks, respectively. For MMLU, due to the rank being already small, like 8, we don't run APOLLO-Mini.

Table 4. Comparison of various finetuning approaches on common-sense reasoning tasks. Experiments are conducted with Llama-3.2-1B based on the implementation from (Liu et al., 2024a).

Methods	WG	PIQA	SIQA	OBQA	HS	BoolQ	Arc-E	Arc-C	Average
AdamW	68.19	76.12	72.36	69.00	69.19	64.34	72.22	55.12	68.07
LoRA	67.56	63.28	71.65	68.20	19.13	63.58	67.30	52.99	59.21
DoRA	68.98	74.70	72.47	64.80	63.93	64.01	69.32	52.82	66.38
GaLore	62.75	72.63	68.17	62.20	47.81	58.99	68.94	47.61	61.14
Fira	71.82	77.20	73.08	69.00	68.21	64.31	73.40	54.78	68.98
APOLLO *w.* SVD	70.88	77.69	72.52	70.60	68.19	63.00	74.03	55.72	**69.08**
APOLLO	70.40	76.93	72.72	70.60	63.75	62.69	73.40	55.20	**68.21**
APOLLO-Mini	67.64	76.50	72.88	69.60	66.54	64.22	72.98	55.46	**68.23**

Table 5. Comparison results of various memory-efficient fine-tuning algorithms on MMLU tasks. For Galore, Fira, and APOLLO, we report the best accuracy obtained by sweeping the learning rate within the range [1e-5, 2.5e-5, 5e-5, 1e-4,1.5e-4, 2e-4].

Model	Methods	STEM	Social Sciences	Humanities	Other	Average
LLaMA-3-8B	Full	54.27	75.66	59.08	72.80	64.85
	LoRA	53.00	74.85	58.97	72.34	64.25
	GaLore	54.20	75.37	58.03	72.34	64.31
	Fira	53.53	75.46	58.59	72.09	64.32
	APOLLO *w.* SVD	54.73	75.46	58.72	72.68	**64.76**
	APOLLO	54.07	75.36	58.08	71.93	64.25
Gemma-7B	Full	30.03	37.16	34.08	35.47	34.21
	LoRA	26.23	34.94	30.88	36.96	32.18
	GaLore	25.47	33.21	31.07	33.71	30.95
	Fira	28.07	35.30	32.63	35.97	33.01
	APOLLO *w.* SVD	29.20	40.42	32.40	38.94	**34.98**
	APOLLO	27.53	36.97	33.99	36.40	33.81
Mistral-7B	Full	52.40	72.95	55.16	69.05	61.67
	LoRA	52.13	72.46	55.05	68.77	61.41
	GaLore	51.27	72.99	55.07	69.30	61.47
	Fira	52.80	72.85	55.07	69.11	61.72
	APOLLO *w.* SVD	52.43	73.28	55.05	69.24	**61.76**
	APOLLO	51.23	72.95	55.14	69.58	61.54

Main Results. As shown in Table 4 and Table 5, APOLLO consistently matches or outperforms other baselines, achieving up to an 1.01 average accuracy improvement over full-rank AdamW on common-sense reasoning tasks. Notably, compared to AdamW, APOLLO requires only one-quarter of the memory overhead for optimization states, while APOLLO-Mini uses a rank of 1, resulting in negligible memory costs for optimization states. Despite this, both approaches deliver a clear margin of accuracy improvement over AdamW on commonsense tasks while maintaining comparable performance on the MMLU tasks.

5.3 End-to-End System-level Benefits

We further validate the system-level benefits, end-to-end throughput, and memory overhead by running LLaMA-7B on 8× A100-80GB GPUs, comparing APOLLO and APOLLO-Mini against AdamW in Fig. 1.

Higher throughput: APOLLO achieves significantly higher throughput than AdamW, particularly under limited hardware resources, as AdamW struggles with large batch sizes due to memory constraints. With APOLLO, we can scale the batch size up to 4× that of AdamW, resulting in up to a 3× improvement in throughput. Additionally, the APOLLO series avoids the extra computational overhead associated with SVD that are known to be expensive.

Much lower memory usage: With a batch size of 4, Adam already reaches the memory limit with a memory usage of ∼79 GB, while APOLLO and APOLLO-Mini require only ∼70 GB and ∼68 GB, respectively, for a batch size of 16. This study highlights that using Adam not only results in high memory costs but also reduces training efficiency, as models become memory-bound, preventing full utilization of computational resources. By using APOLLO, we can effectively increase batch size, accelerating large-scale training with even better performance.

APOLLO-Mini unlocks pre-training LLaMA-13B on A100 80GB without system-Level optimization: Leveraging the exceptional memory efficiency of APOLLO-Mini, we are the first to enable the pre-training of a LLaMA-13B model A100 80GB GPU with naive DDP, without requiring other system-level optimizations, such as model sharding. This breakthrough not only simplifies deployment by reducing engineering complexity but also empowers researchers to scale up model sizes effortlessly with APOLLO-Mini.

Combination with weight quantization unlocks pre-training LLaMA-7B under 12 GB: With our methods significantly reducing optimizer memory costs, the memory consumed by model weights becomes the next dominant factor. To further address this challenge, we integrate our approach with the Int8 weight quantization method proposed in Q-GaLore (Zhang et al., 2024c), enabling even greater memory savings. The results, detailed in Table 6, demonstrate that our Q-APOLLO series effectively mini-

Table 6. Validation of pretraining perplexity of APOLLO-series combined with int-8 weight quantization strategy (Zhang et al., 2024c). APOLLO-series uses a quantization group size of 128. ♠ are collected from (Zhao et al., 2024) and (Zhang et al., 2024c).

Methods	60M		130M		350M	
	Perplexity	Memory	Perplexity	Memory	Perplexity	Memory
AdamW♠	34.06	0.36G	25.08	0.76G	18.80	2.06G
GaLore♠	34.88	0.24G	25.36	0.52G	18.95	1.22G
Q-GaLore♠	34.88	0.18G	25.53	0.39G	19.79	0.88G
APOLLO	31.55	0.24G	22.94	0.52G	16.85	1.22G
Q-APOLLO	31.97	0.18G	24.16	0.39G	18.79	0.88G
APOLLO-Mini	31.93	0.12G	23.84	0.25G	17.18	0.69G
Q-APOLLO-Mini	33.05	0.06G	24.70	0.12G	18.90	0.35G

mizes memory usage for both optimizer and weight components while maintaining on-par or better pre-training perplexity compared to full-precision AdamW. Moreover, Q-APOLLO achieves a clear performance margin over Q-GaLore, underscoring its superiority in both memory efficiency and model quality.

By successfully combining APOLLO-series with quantization, we enable—**for the first time**—the pre-training of a LLaMA-7B model using just 12 GB of memory (Q-APOLLO-Mini), assuming a layer-wise gradient update strategy (Lv et al., 2023) is employed. This represents a significant breakthrough in making LLM pre-training feasible on low-end GPUs, eliminating the dependency on high-end hardware traditionally required for LLM training. Our approach democratizes access to LLM pre-training, making it accessible to a broader range of researchers and organizations with limited resources.

5.4 Extra Investigation and Ablation Study

This section presents multiple experimental investigations, focusing on four key research questions: *Q1:* How can the scaling factor subspace be identified? *Q2:* Is APOLLO sensitive to the number of rank? *Q3:* What is an appropriate granularity for scaling factors? *Q4:* How do detailed comparisons evolve throughout the training process? *Q5:* How APOLLO performs in long-context training setting?

***A1*: Similar performance between Random Projection (RP) and Singular Value Decomposition (SVD).** Previous low-rank gradient-based approaches (Zhao et al., 2024) rely on SVD to identify the gradient subspace, frequently updated during training. This process is time-consuming, thereby affecting training throughput. For a LLaMA-7B model, each SVD operation takes approximately ten minutes, resulting in an additional 25 hours of training time over 150K steps when the subspace is updated every 1,000 steps. To alleviate this overhead, (Zhang et al., 2024c) employs a lazy subspace updating strategy, though it still incurs substantial SVD costs. In this section, we demonstrate that APOLLO performs effectively with random projection, significantly reducing the heavy SVD costs present in previous memory-efficient training algorithms. We pre-train LLaMA models ranging from 60M to 350M on the C4 dataset using GaLore, APOLLO, and APOLLO-Mini, reporting results for both SVD and random projection in each method. As shown in Figure 4 (a-c), GaLore is significantly impacted by random projection, failing to match the performance of AdamW (red dashed line). In contrast, both APOLLO and APOLLO-Mini demonstrate strong robustness with random projection, even slightly outperforming SVD in certain cases, such as APOLLO-Mini on LLaMA-350M. These results confirm the effectiveness of APOLLO under random projection, thereby addressing the throughput challenges present in previous low-rank gradient methods.

***A3*: APOLLO-Mini remains effective even with a rank of 1.** We carry out an ablation study on pre-training LLaMA-60M with different rank budgets, as shown in Figure 4 (d). The results demonstrate that GaLore's performance degrades significantly as the rank decreases, matching full-rank AdamW only when the rank is set to 128 (one-quarter of the original dimension), which limits its effectiveness in extreme low-rank scenarios. In contrast, APOLLO exhibits much better robustness with smaller rank settings compared to both GaLore and Fira, achieving performance comparable to full-rank AdamW even with lower ranks.

Interestingly, APOLLO-Mini shows the best rank efficiency, remaining effective even with a rank of 1, clearly outperforming AdamW. By averaging the gradient scaling factor across different channels, APOLLO-Mini seems to effectively mitigates the errors introduced by low-rank projection. This capability allows APOLLO-Mini to achieve SGD level memory cost while reaching superior performance than AdamW.

***A3*: The gradient scaling factor can even be calculated at the tensor level.** In Table 7, we compare the pre-training

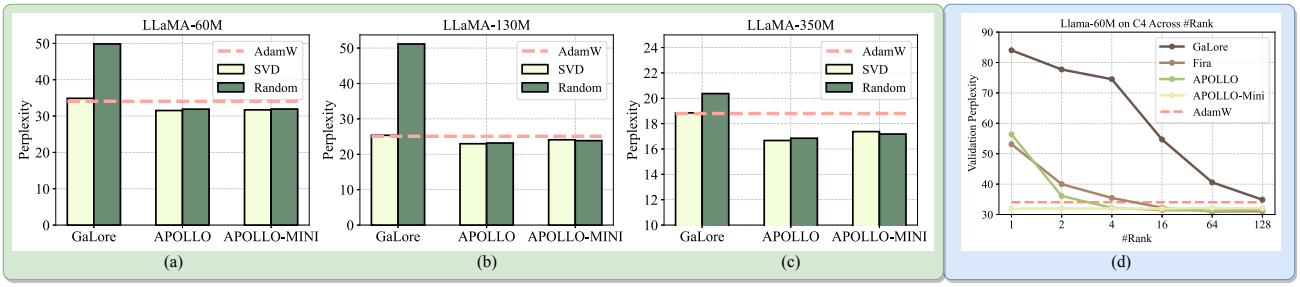

Figure 4. (a-c) Comparison results of various optimization methods using singular value decomposition or random projection. The experiments were conducted on LLaMA-60M/130M/350M models for C4 pretraining tasks. (d) Validation perplexity with varying rank sizes, where 128 is one-quarter of the original model dimension. The red dashed line indicates the performance of full-rank AdamW.

Table 7. Ablation study on the granularity of gradient scaling factors. Perplexity on the validation set is reported.

Methods	Granularity	60M	130M	350M
AdamW		34.06	25.08	18.80
GaLore		34.88	25.36	18.95
APOLLO *w.* SVD	*Channel*	31.26	22.84	16.67
	Tensor	31.77	23.86	16.90
APOLLO	*Channel*	31.55	22.94	16.85
	Tensor	32.10	23.82	17.00

perplexity of our method using different scaling factor granularities. Here, *Channel* indicates that the gradient scaling factor is calculated along the channels with the smaller dimension of each layer, while *Tensor* denotes that a single gradient scaling factor is used for each layer. We keep one-quarter of the original model dimension as the rank. Across model sizes ranging from 60M to 350M, the perplexity difference between these granularities is minimal and both configurations outperform AdamW and GaLore. These results demonstrate that using a tensor-wise scaling factor is sufficient for modest rank training (one-quarter of the original dimension). However, in extreme low-rank scenarios, tensor-wise scaling factor (APOLLO-Mini) outperforms channel-wise ones (APOLLO), as shown in Figure 4 (d).

***A4*: APOLLO performs better with larger model sizes and more training tokens.** Figure 5 illustrates the validation perplexity across the training process for LLaMA-350M models. In the early training stages, Fira shows faster convergence and lower perplexity. However, APOLLO gradually catches up, achieving improved performance in the later stages. This observation suggests that AdamW optimization states play a more crucial role in the initial phase (as Fira maintains the low-rank format of these states), while compressing the optimization states into gradient scaling factors (as done in APOLLO) becomes more effective in later stages. Additionally, Figure 5 indicates that APOLLO seem to benefit from increased training tokens. To quantify

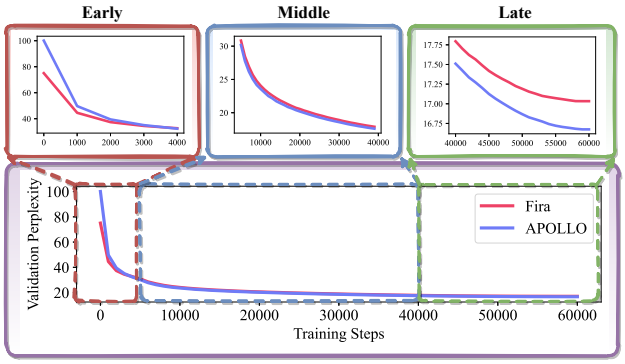

Figure 5. Validation perplexity of pretraining LLaMA-350M on the C4 dataset, with zoomed-in figures showing early, middle, and late stages of training at top, with full training period at bottom.

this effect, we pre-trained LLaMA-130M models for {20k, 30k} steps, with final perplexities for Fira and APOLLO reaching {22.73, 21.69} and {22.84, 21.71}, respectively, further confirming that APOLLO can gradually catch up Fira with more training tokens. Furthermore, Table 2 shows that as model size increases, APOLLO demonstrates better scaling capabilities than Fira: validation perplexity decreases from 31.55 to 14.20 when scaling model sizes from 60M to 1B, whereas Fira only improves from 31.06 to 14.31. Overall, APOLLO exhibits superior performance with both larger model sizes and additional training tokens.

***A5*: APOLLO performs on par with or even better than AdamW in the long-context setting.** Training LLM with long context windows is computationally expensive, but it is critical to enhance LLM performance by involving more contexts to reason. Here, we further validate the effectiveness of the APOLLO series on pre-training a LLaMA-350m with a long context window of 1024, four times over original GaLore uses. To establish a strong baseline, we vary AdamW's learning rate across a range of [1e-4, 2.5e-4, 5e-4, 7.5e-4, 1e-3]. We also lazily tune the scale factor of APOLLO-series by varying APOLLO's in $[\sqrt{1}, \sqrt{2}, \sqrt{3}]$ and APOLLO-Mini's in $[\sqrt{128}, \sqrt{256}, \sqrt{384}]$.

As shown in Fig. 6, both APOLLO and APOLLO-Mini demonstrate better performance than AdamW while achieving drastic reductions in optimizer memory costs—1/8 or even 1/1024 of AdamW's memory usage. Note that our methods generally exhibit even better performance in the later stage with more training tokens involved, marking it a promising candidate in partial LLM pre-training settings, i.e., long-context window and trillions of training tokens.

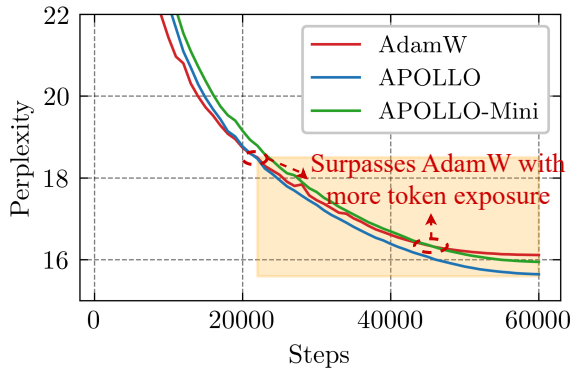

Figure 6. Perplexity curves of the LLaMA-350M model trained in a long-context window setting. APOLLO and APOLLO-Mini outperform AdamW with a grid-searched learning rate, demonstrating the effectiveness of the APOLLO series in industrial LLM pre-training settings(long sequences and extensive training tokens).

6 Conclusion

In this paper, we introduced a novel approach for training large language models (LLMs) that strikes an effective balance between memory efficiency and performance. Motivated by the limitations of existing methods like GaLore, which rely on memory-intensive SVD, and inspired by techniques like Fira and Adam-mini, we proposed two methods to achieve structured-wise gradient scaling. Our approach leverages low-rank optimizer states, using random projection only to preserve gradient norms, enabling efficient training without storing the full optimizer state. Extensive experiments across both pre-training and fine-tuning tasks demonstrate the effectiveness of our APOLLO, consistently surpassing the Adam baseline with greater memory saving than GaLore. APOLLO-Mini further squeeze the memory cost by using a rank-1 sub-space, achieving better performance than Adam at the cost of SGD. Overall, our method offers a promising solution to the memory bottlenecks in LLM training, providing an efficient alternative that maintains high performance while drastically reducing memory consumption.

References

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Biderman, D., Portes, J., Ortiz, J. J. G., Paul, M., Greengard, P., Jennings, C., King, D., Havens, S., Chiley, V., Frankle, J., et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chen, X., Feng, K., Li, C., Lai, X., Yue, X., Yuan, Y., and Wang, G. Fira: Can we achieve full-rank training of llms under low-rank constraint? *arXiv preprint arXiv:2410.01623*, 2024.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Freksen, C. B. An introduction to johnson-lindenstrauss transforms. *arXiv preprint arXiv:2103.00564*, 2021.

Hao, Y., Cao, Y., and Mou, L. Flora: Low-rank adapters are secretly gradient compressors. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uubBZKM99Y.

He, Y., Li, P., Hu, Y., Chen, C., and Yuan, K. Subspace optimization for large language models with convergence guarantees. *arXiv preprint arXiv:2410.11289*, 2024.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Huang, W., Zhang, Z., Zhang, Y., Luo, Z.-Q., Sun, R., and Wang, Z. Galore-mini: Low rank gradient learning with fewer learning rates. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Jiang, Z., Lin, H., Zhong, Y., Huang, Q., Chen, Y., Zhang, Z., Peng, Y., Li, X., Xie, C., Nong, S., et al. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 745–760, 2024.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kocoń, J., Cichecki, I., Kaszyca, O., Kochanek, M., Szydło, D., Baran, J., Bielaniewicz, J., Gruza, M., Janz, A., Kanclerz, K., et al. Chatgpt: Jack of all trades, master of none. *Information Fusion*, 99:101861, 2023.

Lialin, V., Muckatira, S., Shivagunde, N., and Rumshisky, A. Relora: High-rank training through low-rank updates. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*, 2023.

Liang, K., Liu, B., Chen, L., and Liu, Q. Memory-efficient llm training with online subspace descent. *arXiv preprint arXiv:2408.12857*, 2024.

Liu, S., Chen, T., Chen, X., Chen, X., Xiao, Q., Wu, B., Kärkkäinen, T., Pechenizkiy, M., Mocanu, D., and Wang, Z. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022.

Liu, S., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. DoRA: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=3d5CIRG1n2.

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*, 2024b.

Loeschcke, S. B., Toftrup, M., Kastoryano, M., Belongie, S., and Snæbjarnarson, V. Loqt: Low-rank adapters for quantized pretraining. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Loshchilov, I. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Lv, K., Yang, Y., Liu, T., Gao, Q., Guo, Q., and Qiu, X. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*, 2023.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Robert, T., Safaryan, M., Modoranu, I.-V., and Alistarh, D. Ldadam: Adaptive optimization from low-dimensional gradient statistics. *arXiv preprint arXiv:2410.16103*, 2024.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sap, M., Rashkin, H., Chen, D., LeBras, R., and Choi, Y. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

Tang, Y., Liu, F., Ni, Y., Tian, Y., Bai, Z., Hu, Y.-Q., Liu, S., Jui, S., Han, K., and Wang, Y. Rethinking optimization and architecture for tiny language models. *arXiv preprint arXiv:2402.02791*, 2024.

Thangarasa, V., Gupta, A., Marshall, W., Li, T., Leong, K., DeCoste, D., Lie, S., and Saxena, S. Spdf: Sparse pre-training and dense fine-tuning for large language models. In *Uncertainty in Artificial Intelligence*, pp. 2134–2146. PMLR, 2023.

Wainwright, M. J. Chapter 2: Tail bounds, 2015. URL https://www.stat.berkeley.edu/~mjwain/stat210b/Chap2_TailBounds_Jan22_2015.pdf.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, H., Zhou, Y., Xue, Y., Liu, Y., and Huang, J. G10: Enabling an efficient unified gpu memory and storage architecture with smart tensor migrations. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 395–410, 2023.

Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R., and Luo, Z.-Q. Why transformers need adam: A hessian perspective. *arXiv preprint arXiv:2402.16788*, 2024a.

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

Zhang, Z., Jaiswal, A., Yin, L., Liu, S., Zhao, J., Tian, Y., and Wang, Z. Q-galore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. *arXiv preprint arXiv:2407.08296*, 2024c.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient LLM training by gradient low-rank projection. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=hYHsrKDiX7.

Zheng, Y., Zhang, R., Zhang, J., Ye, Y., Luo, Z., Feng, Z., and Ma, Y. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

A APPENDIX

A.1 Proof of Gradient Scaling Approximation in Random Projected Low-rank Space

A.1.1 Problem Statement

Notations and Definitions: We first introduce the notations and definitions used in the proof:

- Let $\mathbf{G}_t \in \mathbb{R}^{m\times n}$ denote the gradient matrix at iteration t ($m \leq n$).
- Let $\mathbf{P} \in \mathbb{R}^{r\times m}$ denote the random projection matrix where $P_{ij} \sim N(0, 1/r)$ i.i.d.
- Define $\mathbf{R}_t = \mathbf{P}\mathbf{G}_t$ as the projected gradient.
- Let $\beta_1, \beta_2 \in (0, 1)$ be exponential decay rates.
- Define $\mathbf{M}_t$, $\mathbf{V}_t$ as first and second moments in the original space.
- Define $\mathbf{M}_t^R$, $\mathbf{V}_t^R$ as first and second moments in projected space.
- Let T denote the total number of iterations.
- Let n denote the number of channels.
- Assume zero initialization: $\mathbf{M}_0 = \mathbf{M}_0^R = 0$, $\mathbf{V}_0 = \mathbf{V}_0^R = 0$.
- $\|\|$ indicates ℓ_2 norm of a vector.
- $\|\|_1$ indicates ℓ_1 norm of a vector.

Problem: We aim to prove that gradient scaling factors s_j and s_j^R in the original and low-rank projected space have a bound for their ratio s_j^R/s_j,

$$s_j^R/s_j = \frac{\|\mathbf{G}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|}\frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\mathbf{R}_t[:,j]\|} = \frac{\|\mathbf{G}_t[:,j]\|}{\|\mathbf{R}_t[:,j]\|}\frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|}$$

where:

$$\tilde{\mathbf{R}}_t = \frac{\mathbf{M}_t^R}{\sqrt{\mathbf{V}_t^R}}$$

and

$$\tilde{\mathbf{G}}_t = \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}}$$

A.1.2 Norm Preservation Under Random Projection

Theorem A.1 (Norm Preservation)**.** *For any fixed vector $x \in \mathbb{R}^m$ and random matrix $\mathbf{P} \in \mathbb{R}^{r\times m}$ where $P_{ij} \sim \mathcal{N}(0, 1/r)$ i.i.d., the following holds with high probability:*

$$\Pr[(1-\epsilon)\|x\|^2 \leq \|\mathbf{P}x\|^2 \leq (1+\epsilon)\|x\|^2] \geq 1-2\exp\left(-\frac{r\epsilon^2}{8}\right). \qquad \Pr\left(\left|\frac{Z}{r}-1\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{r\epsilon^2}{8}\right).$$

Theorem A.1 is proven by leveraging the properties of Gaussian random projections and the concentration inequality for the chi-squared distribution.

Proof. The projected norm $\|\mathbf{P}x\|^2$ can be expressed as:

$$\|\mathbf{P}x\|^2 = \sum_{j=1}^{r}\left(\sum_{i=1}^{m}\mathbf{P}_{ji}x_i\right)^2.$$

Rewriting this using the quadratic form, we have:

$$\|\mathbf{P}x\|^2 = x^\top\mathbf{P}^\top\mathbf{P}x,$$

where $\mathbf{P}^\top\mathbf{P}$ is a symmetric $m \times m$ matrix. To analyze $\|\mathbf{P}x\|^2$, consider the distribution of $\mathbf{P}^\top\mathbf{P}$.

Each entry of $\mathbf{P}^\top\mathbf{P}$ is given by:

$$(\mathbf{P}^\top\mathbf{P})_{ij} = \sum_{k=1}^{r}\mathbf{P}_{ki}\mathbf{P}_{kj}.$$

For $i = j$ (diagonal entries), we have:

$$(\mathbf{P}^\top\mathbf{P})_{ii} = \sum_{k=1}^{r}\mathbf{P}_{ki}^2,$$

and since $\mathbf{P}_{ki} \sim \mathcal{N}(0, 1/r)$, $\mathbf{P}_{ki}^2 \sim$ Exponential$(1/r)$.

Therefore, $(\mathbf{P}^\top\mathbf{P})_{ii} \sim \frac{1}{r}\chi_r^2$, where χ_r^2 is the chi-squared distribution with r degrees of freedom. For $i \neq j$ (off-diagonal entries), the expectation is zero:

$$\mathbb{E}[(\mathbf{P}^\top\mathbf{P})_{ij}] = 0,$$

due to the independence of $\mathbf{P}_{ki}$ and $\mathbf{P}_{kj}$.

The expected value of $\mathbf{P}^\top\mathbf{P}$ is therefore:

$$\mathbb{E}[\mathbf{P}^\top\mathbf{P}] = I_m,$$

where I_m is the identity matrix.

The expectation of $\|\mathbf{P}x\|^2$ is:

$$\mathbb{E}[\|\mathbf{P}x\|^2] = x^\top\mathbb{E}[P^\top\mathbf{P}]x = x^\top I_m x = \|x\|^2.$$

Now consider the concentration of $\|\mathbf{P}x\|^2$ around its expectation. Define the random variable:

$$Z = \frac{r\|\mathbf{P}x\|^2}{\|x\|^2}.$$

Since $\mathbf{P}_{ij}$ entries are i.i.d., $Z \sim \chi_r^2$. Using the moment generating function of χ_r^2, the following concentration bounds can be derived using standard tail inequalities for sub-exponential random variables (Wainwright, 2015):

Returning to $\|\mathbf{P}x\|^2$, we scale Z back:

$$\|\mathbf{P}x\|^2 = \frac{Z\|x\|^2}{r}.$$

Thus, with high probability:

$$(1-\epsilon)\|x\|^2 \leq \|\mathbf{P}x\|^2 \leq (1+\epsilon)\|x\|^2,$$

and the probability of this event is at least:

$$1 - 2\exp\left(-\frac{r\epsilon^2}{8}\right).$$

□

A.1.3 First Moment Analysis

Theorem A.2 (First Moment Preservation)**.** *For any channel j, with probability at least $1 - 2\exp\left(-\frac{r\epsilon^2}{8}\right)$:*

$$(1-\epsilon)\|\mathbf{M}_t[:,j]\|^2 \leq \|\mathbf{M}_t^R[:,j]\|^2 \leq (1+\epsilon)\|\mathbf{M}_t[:,j]\|^2,$$

using a fixed projection matrix $\mathbb{R}^{r\times m}$ over t.

Proof. Our goal is to bound $\|\mathbf{M}_t^R[:,j]\|$ in terms of $\|\mathbf{M}_t[:,j]\|$.

Step 1: Recursive Definitions of $\mathbf{M}_t[:,j]$ and $\mathbf{M}_t^R[:,j]$. The first moment $M_t[:,j]$ in the original space is recursively defined as:

$$\mathbf{M}_t[:,j] = (1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k \mathbf{G}_{t-k}[:,j],$$

where $\mathbf{G}_{t-k}[:,j] \in \mathbb{R}^m$ is the gradient at timestep $t-k$.

The projected first moment $\mathbf{M}_t^R[:,j]$ is similarly defined as:

$$\mathbf{M}_t^R[:,j] = (1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k \mathbf{R}_{t-k}[:,j],$$

where $\mathbf{R}_{t-k}[:,j] = \mathbf{P}\mathbf{G}_{t-k}[:,j] \in \mathbb{R}^r$.

Step 2: Projected First Moment in a Lower Dimension. With a random matrix $\mathbf{P} \in \mathbb{R}^{r\times m}$ where $P_{ij} \sim \mathcal{N}(0, 1/r)$ i.i.d., we have the projected first moment in the low-rank space,

$$\begin{aligned}\mathbf{M}_t^R[:,j] &= (1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k \mathbf{R}_{t-k}[:,j] \\ &= (1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k \mathbf{P}\mathbf{G}_{t-k}[:,j] \\ &= \mathbf{P}\left((1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k \mathbf{G}_{t-k}[:,j]\right) \\ &= \mathbf{P}\mathbf{M}_t[:,j]\end{aligned},$$

by factoring $\mathbf{P}$ out of the summation.

This implies the $\mathbf{M}_t^R[:,j]$ can be viewed as a projected version of $\mathbf{M}_t[:,j]$ into a lower dimension with a fixed $\mathbf{P}$ over time t.

Step 3: Properties of Random Projection By Theorem A.1, we have the norm of $\mathbf{M}_t[:,j]$ is preserved in a high probability,

$$\begin{aligned}\Pr\Big((1-\epsilon)\|\mathbf{M}_t[:,j]\|^2 &\leq \|\mathbf{M}_t^R[:,j]\|^2 \\ &\leq (1+\epsilon)\|\mathbf{M}_t[:,j]\|^2\Big) \\ &\geq 1 - 2\exp\left(-\frac{r\epsilon^2}{8}\right).\end{aligned} \tag{10}$$

Remark: Here, we assume the projection matrix is fixed over time step t. GaLore (Zhao et al., 2024) also derives their theorem with the same assumption. However, as acknowledged in GaLore, using the same projection matrix for the entire training may limit the directions in which the weights can grow. Therefore, empirically, as in GaLore, we can periodically resample $\mathbf{P}$ over T iterations to introduce new directions. Unlike GaLore, which uses time-consuming SVD-based updates, we can simply re-sample $\mathbf{P}$ from the Gaussian distribution by changing the random seed. □

A.1.4 Second Moment Analysis

Theorem A.3 (Second Moment Preservation)**.** *For any channel j and time t, if*

$$r \geq \frac{8}{\epsilon^2}\log\left(\frac{2t}{\delta}\right),$$

then with probability at least $1 - \delta/2$:

$$(1-\epsilon)\|\mathbf{V}_t[:,j]\|_1 \leq \|\mathbf{V}_t^R[:,j]\|_1 \leq (1+\epsilon)\|\mathbf{V}_t[:,j]\|_1$$

, where $\mathbf{V}_t[:,j]$ and $\mathbf{V}_t^R[:,j]$ are the second moments in the original and projected spaces, respectively.

Proof. Our goal is to show that the norm of the second moment $\mathbf{V}_t$ in the original space is preserved under projection to the lower-dimensional space. We proceed by analyzing the recursive definition of $\mathbf{V}_t$ and applying the results of Theorem A.2 on norm preservation.

Step 1: Recursive Formulation of $\mathbf{V}_t$ The second moment $\mathbf{V}_t[:,j]$ for channel j at iteration t is defined recursively as:

$$\mathbf{V}_t[:,j] = \beta_2 \mathbf{V}_{t-1}[:,j] + (1-\beta_2)(\mathbf{G}_t[:,j])^2$$

By expanding recursively, we can write $\mathbf{V}_t[:, j]$ as a weighted sum of the squared gradients from all past iterations:

$$\mathbf{V}_t[:, j] = (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k (\mathbf{G}_{t-k}[:, j])^2$$

Step 2: Projected Second Moment in Lower Dimension Similarly, in the projected space, the second moment $\mathbf{V}_t^R[:, j]$ for channel j at iteration t is given by:

$$\mathbf{V}_t^R[:, j] = \beta_2 \mathbf{V}_{t-1}^R[:, j] + (1 - \beta_2)(\mathbf{R}_t[:, j])^2$$

Expanding recursively, we have:

$$\mathbf{V}_t^R[:, j] = (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k (\mathbf{R}_{t-k}[:, j])^2$$

Step 3: ℓ_1 Norm of Channel-wise Second Moment. Then, we can obtain the ℓ_1 norm of the second-moment term $\mathbf{V}_t^R[:, j]\|_1$

$$\|\mathbf{V}_t^R[:, j]\|_1 = \sum_{i=1}^{r} (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k (\mathbf{R}_{t-k}[i, j])^2,$$

We can swap the summation order and have,

$$\begin{aligned}\|\mathbf{V}_t^R[:, j]\|_1 &= (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k \sum_{i=1}^{r} (\mathbf{R}_{t-k}[i, j])^2 \\ &= (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k ||\mathbf{R}_{t-k}[:, j]||^2\end{aligned}$$

Similarly, we can have

$$\begin{aligned}\|\mathbf{V}_t[:, j]\|_1 &= (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k \sum_{i=1}^{n} (\mathbf{G}_{t-k}[i, j])^2 \\ &= (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k ||\mathbf{G}_{t-k}[:, j]||^2\end{aligned}$$

Step 4: Constructing the Bounds for $\mathbf{V}_t^R[:, j]$ By Theorem A.1, we know that for each k, the ℓ_2 norm of the projected gradient $\|\mathbf{R}_{t-k}[:, j]\|$ satisfies:

$$(1-\epsilon)\|\mathbf{G}_{t-k}[:, j]\|^2 \leq \|\mathbf{R}_{t-k}[:, j]\|^2 \leq 1+\epsilon\|\mathbf{G}_{t-k}[:, j]\|^2,$$

with probability $\geq 1 - 2\exp(-r\epsilon^2/8)$.

Therefore,

$$\begin{aligned}&\|\mathbf{V}_t^R[:, j]\|_1 = (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k ||\mathbf{R}_{t-k}[:, j]||^2 \\ &\leq (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k (1 + \epsilon) ||G_{t-k}[:, j]||^2 = (1 + \epsilon)\|V_t[:, j]\|_1\end{aligned}$$

Similarly, we can obtain the lower bound,

$$\begin{aligned}&\|\mathbf{V}_t^R[:, j]\|_1 = (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k ||\mathbf{R}_{t-k}[:, j]||^2 \\ &\geq (1 - \beta_2) \sum_{k=0}^{t-1} \beta_2^k (1 - \epsilon) ||G_{t-k}[:, j]||^2 = (1 - \epsilon)\|V_t[:, j]\|_1\end{aligned}$$

We obtain the following bounds for the ℓ_1 norm of full projected second moment $\mathbf{V}_t^R[:, j]$:

$$\|(1 - \epsilon)\mathbf{V}_t[:, j]\|_1 \leq \|\mathbf{V}_t^R[:, j]\|_1 \leq \|(1 + \epsilon)\mathbf{V}_t[:, j]\|_1$$

.

Step 5: Probability of Success. To ensure the bound holds across all t timesteps, we apply the union bound. For each k, the failure probability is $2\exp(-r\epsilon^2/8)$. Across t timesteps, the total failure probability is:

$$2t \exp\left(-\frac{r\epsilon^2}{8}\right).$$

Set this total failure probability to $\delta/2$, giving the condition:

$$r \geq \frac{8}{\epsilon^2} \log\left(\frac{2t}{\delta}\right).$$

Remark: Here, the requirement that r grows sublinearly as $\log(t)$ ensures that even for large t, the rank r does not grow excessively. However, empirically, we find our method is not sensitive to rank selection; even a rank of 256 is sufficient to train LLaMA 7B with 150k steps. This can be explained by recent Adam-mini (Zhang et al., 2024b) that the variance doesn't need to be precise, and a block-wise approximation is enough, showing that the variance approximation error can be tolerated well.

□

A.1.5 Main Result: Gradient Scaling Approximation

Theorem A.4 (Main Result)**.** *For any channel j, with probability $\geq 1 - \delta$:*

$$\frac{\sqrt{1-\epsilon}}{1+\epsilon} \leq \sqrt{\frac{n}{r}} \frac{s_j^R}{s_j} \leq \frac{\sqrt{1+\epsilon}}{1-\epsilon}$$

Proof. Express ratio:

$$\frac{s_j^R}{s_j} = \frac{\|\mathbf{G}_t[:, j]\|}{\|\mathbf{R}_t[:, j]\|} \frac{\|\tilde{\mathbf{R}}_t[:, j]\|}{\|\tilde{\mathbf{G}}_t[:, j]\|}$$

Apply Theorem A.2 for the first part, we can obtain the error bound for the first part:

$$\frac{\|\mathbf{G}_t[:, j]\|}{\|\mathbf{R}_t[:, j]\|} \in [\sqrt{\frac{1}{1+\epsilon}}, \sqrt{\frac{1}{1-\epsilon}}]$$

For the second part, it is equal to

$$\begin{aligned}\frac{\|\tilde{\mathbf{R}}_t[:,j]\|^2}{\|\tilde{\mathbf{G}}_t[:,j]\|^2} &= \frac{\|(\frac{\mathbf{M}_t^R}{\sqrt{\mathbf{V}_t^R}})[:,j]\|^2}{\|(\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}})[:,j]\|^2} \\ &= \frac{\sum_{i=1}^r (\frac{\mathbf{M}_t^R}{\sqrt{\mathbf{V}_t^R}})^2[i,j]}{\sum_{i=1}^n (\frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}})^2[i,j]}\end{aligned} \tag{11}$$

SGD with Momentum only If we handle SGD with Momentum only, where variance term above is non-existent, and can be simplified as

$$\frac{\|\tilde{\mathbf{R}}_t[:,j]\|^2}{\|\tilde{\mathbf{G}}_t[:,j]\|^2} = \frac{\|\mathbf{M}_t^R[:,j]\|^2}{\|\mathbf{M}_t[:,j]\|^2}$$

We can easily apply Theorem A.3 for the first-moment term:

$$\sqrt{1-\epsilon} \leq \frac{\|\mathbf{M}_t^R[:,j]\|}{\|\mathbf{M}_t[:,j]\|} \leq \sqrt{1+\epsilon}$$

where the final scaling factor is bounded,

$$\frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|} \in [\sqrt{1-\epsilon}, \sqrt{1+\epsilon}]$$

AdamW AdamW's case is more tricky, as equation 11 involves the element-wise division and cannot easily separate the momentum and variance. However, recent works such as Adam-mini (Zhang et al., 2024b) and GaLore-mini (Huang et al.) find out that the variance term can be approximated as an average of a block-wise (original full-rank space) or channel-wise (projected low-rank space). Given the ℓ_1 norm of the variance term is bounded based on Theorem A.4, we take this assumption by replacing the variance term as the average of variance vector, i.e., $\frac{||\mathbf{V}_t[:,j]||_1}{n}$ and $\frac{||\mathbf{V}_t^R[:,j]||_1}{r}$ in equation 11. Then it is approximated as,

$$\begin{aligned}\frac{\|\tilde{\mathbf{R}}_t[:,j]\|^2}{\|\tilde{\mathbf{G}}_t[:,j]\|^2} &= \frac{\sum_{i=1}^r (\frac{\mathbf{M}_t^R[i,j]^2}{\frac{||\mathbf{V}_t^R[:,j]||_1}{r}})}{\sum_{i=1}^n (\frac{\mathbf{M}_t[i,j]^2}{\frac{||\mathbf{V}_t[:,j]||_1}{n}})} \\ &= (\frac{r}{n}) \frac{||\mathbf{V}_t[:,j]||_1}{||\mathbf{V}_t^R[:,j]||_1} \frac{\|\mathbf{M}_t^R[:,j]\|^2}{\|\mathbf{M}_t[:,j]\|^2}\end{aligned}$$

Multiply inequalities from theorem A.3 and theorem A.4 with union bound probability $\geq 1-\delta$, we have the above term

$$\sqrt{\frac{n}{r}} \frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|} \in [\sqrt{\frac{1-\epsilon}{1+\epsilon}}, \sqrt{\frac{1+\epsilon}{1-\epsilon}}]$$

Then, we have the bounded ratio,

$$\sqrt{\frac{n}{r}} \frac{s_j^R}{s_j} = \sqrt{\frac{n}{r}} \frac{\|\mathbf{G}_t[:,j]\|}{\|\mathbf{R}_t[:,j]\|} \frac{\|\tilde{\mathbf{R}}_t[:,j]\|}{\|\tilde{\mathbf{G}}_t[:,j]\|} \in [\frac{\sqrt{1-\epsilon}}{1+\epsilon}, \frac{\sqrt{1+\epsilon}}{1-\epsilon}]$$

Remark: This contains the constant factor $\sqrt{\frac{n}{r}}$, suggesting we should scale the gradient to make sure it has consistent behavior as AdamW with structured learning rate update. This gradient scale factor can be combined with the learning rate. When the r is too small compared to n, as in our `APOLLO-Mini` case, which uses rank-1 space, we specifically assign the scaling factor by using $\sqrt{128}$.

Probability of Success: We now establish the probability of success. Both Theorem A.3 and Theorem A.4 rely on the same random projection matrix P are derived from Theorem A.2 (norm preservation for random projections). Therefore, the probability of failure for both bounds is governed by the failure of Theorem A.2.

For a single timestep t, the failure probability of Theorem A.2 is:

$$\Pr(\text{Theorem A.2 fails at timestep } t) \leq 2\exp\left(-\frac{r\epsilon^2}{8}\right).$$

Across all t timesteps, the total failure probability (union bound) is:

$$\Pr(\text{Theorem A.2 fails for any timestep}) \leq 2t\exp\left(-\frac{r\epsilon^2}{8}\right).$$

Set this total failure probability to δ:

$$2t\exp\left(-\frac{r\epsilon^2}{8}\right) \leq \delta.$$

Solving for r, we require:

$$r \geq \frac{8}{\epsilon^2}\log\left(\frac{2t}{\delta}\right).$$

This ensures that both Theorem A.3 and Theorem A.4 hold simultaneously with probability $\geq 1-\delta$, which together make Theorem A.5 hold.

□

A.2 Training throughput of GaLore-type Optimizer on LLaMA-1B

We further show the training throughput for Galore-type low-rank optimizer (Galore, Fira) in Fig. 7. At every 200 update step, they need to call SVD to update the projection matrix, leading to a drastic drop in training throughput. Although Galore tries to amortize the cost by relaxing the update gap, the significantly high cost is hard to amortize fully as we still keep a short update gap to keep performance; for example, to update the projection matrix for a LLaMA 7B model needs 10 mins, while inference takes seconds.

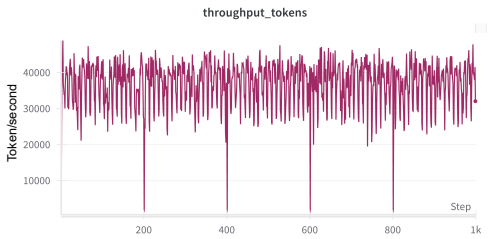

Figure 7. The training throughput of Galore-type low-rank optimizer with many spikes due to the expensive SVD operation every 200 steps.

A.3 Detailed Pre-Training Setting

This section provides an overview of the LLaMA architectures and the hyperparameters used during pre-training. To ensure a fair comparison, we adopt the same settings as Zhao et al. (2024). Table 8 outlines the hyperparameters for the various LLaMA model sizes. Across all architectures, we use a maximum sequence length of 256 and a batch size of 131K tokens. Additionally, we apply a learning rate warm-up over the first 10% of training steps, followed by a cosine annealing schedule that gradually reduces the learning rate to 10% of its initial value.

APOLLO runs using the same learning rate 0.01 and a subspace change frequency T of 200 without tuning, following the Galore open-sourced settings. The scale factor α is considered a fractional learning rate, which is set to 1 by default in APOLLO for models with a size of less than 1B, showing our method doesn't need too much tuning like Galore and Fira. On 1B-model, we set the high-rank APOLLO with a $\alpha = \sqrt{1/2}$ and the high-rank APOLLO w SVD with a $\alpha = 0.4$. As we find the scaling factor increases with the rank r, therefore we scale the gradient factor in APOLLO-Mini with setting α to $\sqrt{128}$.

A.4 Detailed Fine-Tuning Setting

A.4.1 Commonsense reasoning fine-tuning

We use the implementation from (Liu et al., 2024a) with the chosen hyperparameters detailed in Table 9.

A.4.2 MMLU fine-tuning

We use the implementation from (Zheng et al., 2024). We use the rank as 8 and sweep the learning rate within the range [1e-5, 2.5e-5, 5e-5, 1e-4,1.5e-4, 2e-4] for GaLore, Fira, APOLLO to ensure a faithful comparison. The full and LoRA results are from (Zhang et al., 2024c).

Table 8. Hyper-parameters of LLaMA architectures for pre-training.

Params	Hidden	Intermediate	Heads	Layers	Steps	Data Amount (Tokens)
60M	512	1376	8	8	10K	1.3 B
130M	768	2048	12	12	20K	2.6 B
350M	1024	2736	16	24	60K	7.8 B
1 B	2048	5461	24	32	100K	13.1 B
7 B	4096	11008	32	32	150K	19.7 B

Table 9. Hyperparameter of Llama-3.2-1B on the commonsense reasoning tasks.

Hyperparameters	AdamW	LoRA	DoRA	Galore	Fira	APOLLO w.SVD	APOLLO	APOLLO-Mini
Rank r	-	32	32	32	32	32	32	1
α	-	64	64	-	-	-	-	-
scale	-	-	-	0.25	0.25	1.0	5.0	128
Dropout	0.05							
LR	[2e-5, 5e-5]	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4
LR Scheduler	Linear							
Batch size	32							
Warmup Steps	100							
Epochs	3							
Where	Q,K,V,Up,Down							