


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]

Website: https://zhuhanqing.github.io/APOLLO
Code: https://github.com/zhuhanqing/APOLLO

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 adaptation 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 **substantially 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.

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 triples the memory required relative to the model's parameter size. For instance, training an LLaMA-7B model with a single batch 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 175B parameters, memory demands reach 700GB for the model alone and 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

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

Proceedings of the 8[th] MLSys Conference, Santa Clara, CA, USA, 2025.

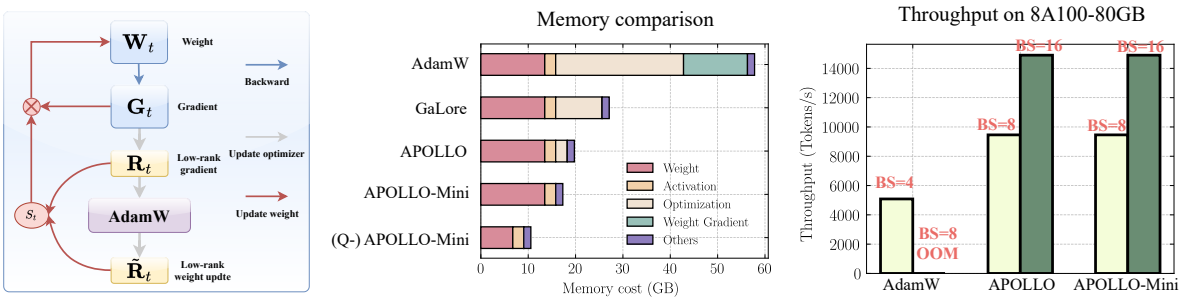

Figure 1. (Left) Overview of our APOLLO optimizer; (Middle) Memory breakdown comparison for a single batch size, where both GaLore and our method employ the layer-wise gradient update strategy (Lv et al., 2023). The (Q-) prefix indicates the integration of INT8 weight quantization, as utilized in (Zhang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

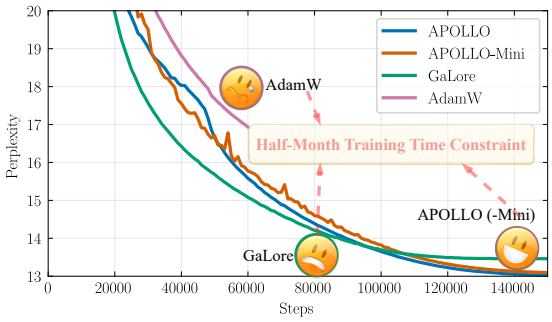

Figure 2. Comparison of Validation perplexity on LLaMA-7B.

community to either use more and higher-end GPUs, or to 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 efforts have focused on solving the high memory costs of training LLMs. One approach reduces the parameter volume by 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). While these techniques effectively reduce memory usage, they restrict the optimization space, resulting in performance trade-offs (Biderman et al., 2024), particularly in pretraining (Lialin et al., 2023).

Another avenue of research focuses on designing memory-efficient optimizers that reduce memory usage 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). GaLore (Zhao et al., 2024) stands out among low-rank methods by enabling full-parameter training of LLMs through low-rank gradient updates via 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 account for the compression of gradient and optimizer states.

However, the periodic updates to the gradient subspace via SVD (e.g., every 200 iterations) incur a computational cost of $O(mn^2)$, prohibitive when the matrix dimensions, m and n, are large. For example, a single subspace update can take $\sim$10 minutes for the LLaMA-7B model, whereas inference only takes seconds. This substantial overhead significantly reduces training throughput, as demonstrated in Fig. 9.

The recent Adam-mini (Zhang et al., 2024b) finds that a block-wise second moment $\mathbf{V}$ suffices for learning rate adjustments, offering an orthogonal, more efficient alternative. However, achieving performance on par with AdamW requires careful handling of different model components to preserve 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 (much below GaLore and its variants & close to SGD) while matching 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. We introduce a memory-efficient approximation for the scaling factors using an auxiliary optimizer state, requiring only lower-dimensional gradient information as input. This significantly reduces memory usage by leveraging compressed gradient representation. Moreover, we prove that an **SVD-free** low-rank projection via random projections is sufficient, eliminating the need for costly SVD. Notably, we show that a much lower rank, or even a **rank-1 approximation**, is sufficient to capture the structured gradient scaling factors. 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 than GaLore at SGD-like costs.

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.
- **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 record-low SGD-level memory costs while still outperforming AdamW.

We demonstrate the efficacy of the `APOLLO` series in both pre-training and fine-tuning scenarios. In pre-training, across LLaMA models ranging from 60M to 7B, `APOLLO` and `APOLLO-Mini` consistently outperform AdamW, achieving up to a 2.8 $\downarrow$ in validation perplexity while significantly reducing memory overhead by eliminating nearly all optimizer states. In fine-tuning, `APOLLO`-series achieves 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 Fig.2); (2) **Extreme training memory savings**. By combining `APOLLO-Mini` with weight quantization, we **set a new record for memory efficiency**: pre-training LLaMA 7B requires only **12GB** of memory (Fig. 1 (middle)). More system-level evaluations are in Section 5.3. These results establish `APOLLO` and `APOLLO-Mini` as highly efficient and scalable solutions for LLM pre-training and fine-tuning, 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 reduces trainable parameters to save memory costs, such as designing 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 with Adam, especially in pre-training scenarios. 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`-series achieves ultra-efficient memory usage without SVD while matching or even surpassing the performance of AdamW. Notably, our extreme variant, `APOLLO-Mini`, drives memory costs down to SGD levels, setting a new record 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, 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. `APOLLO` is orthogonal to these system-level optimizations and can be seamlessly integrated to achieve greater memory efficiency. Furthermore, SVD-free `APOLLO` is more system-friendly.

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 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 LLMs. Recent work, such as Adam-mini (Zhang et al., 2024b), proposes grouping parameters into blocks and applying a **block-wise learning rate adjustment** to reduce memory usage while maintaining Adam(W) performance. However, the block-wise approach in Adam-mini 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 findings of optimizer redundancy, 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 $\tilde{\mathbf{G}}_t = \mathbf{S} \cdot \mathbf{G}_t = \mathbf{G}_t \cdot \mathrm{diag}(s)$.

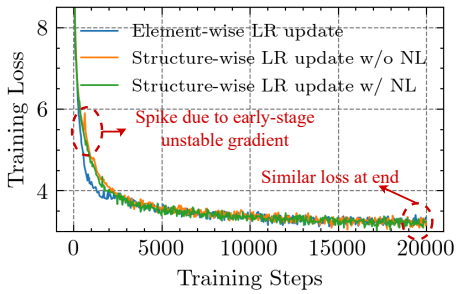

Figure 3. Training loss comparison between Element-wise and Channel-wise Learning Rate (LR) Adaptations with or without norm limiter (NL) on the LLaMA-130M model.

Empirical validation. We first empirically explore the effectiveness of the proposed update rule where we compare the training loss of the original element-wise learning rate adaptation with our proposed channel-wise one on a LLaMA-130M model. As shown in Fig. 3, both approaches achieve similar final training loss, 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 our effectiveness. 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:

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

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

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, $\mathbf{M}_t, \mathbf{V}_t$.
  $\mathbf{M}_t^R, \mathbf{V}_t^R \leftarrow \mathrm{AdamW}(\mathbf{R}_t, \beta_1, \beta_2, \lambda = 0)$
  $\tilde{\mathbf{R}}_t \leftarrow \mathbf{M}_t^R/(\sqrt{\mathbf{V}_t^R} + \epsilon)$
  # Step 3: Obtain approximated gradient scaling factor.
  if APOLLO **then**
    $\mathbf{S} \leftarrow \mathrm{diag}(s_0^R, s_1^R, ..., s_m^R)$ $\{s_i^R = \frac{\|\tilde{\mathbf{R}}_t[:,j]\|_2}{\|\mathbf{R}_t[:,j]\|_2}\}$
  else if APOLLO-Mini **then**
    $\mathbf{S} \leftarrow s^R$ $\{s^R = \frac{\|\tilde{\mathbf{R}}_t\|_2}{\|\mathbf{R}_t\|_2}\}$
  end if
  # Step 4: Update weight in original space.
  $\mathbf{W}_t \leftarrow \mathbf{W}_{t-1} + \eta \cdot \alpha \cdot \mathbf{G}_t\mathbf{S} - \eta \cdot \lambda\mathbf{W}_{t-1}$
  $t \leftarrow t + 1$
until convergence criteria met
return $\mathbf{W}_T$

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.

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, since the full states $\mathbf{M}_t$ and $\mathbf{V}_t$ are still required. 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

4.1.1 Approximating Gradient Scaling with an Auxiliary Low-Rank Space

To address this question, 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, computed as $\mathbf{R}_t = \mathbf{P}_t\mathbf{G}_t \in \mathbb{R}^{r\times n}$ with a projection matrix $\mathbf{P}_t \in \mathbb{R}^{r\times m}$. It 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:

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

In this way, APOLLO only stores auxiliary low-rank optimizer state, saving memory from $2mn$ to $2nr$. We will show later that the structured scaling renders APOLLO insensitive to the rank, unlocking substantial memory savings. In contrast, GaLore requires a relatively high rank to retain performance (see Sec. 5.4 and Appendix *A3* for details).

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 SVD, as it captures the most informative components of the gradient. In fact, most existing low-rank optimizers for LLMs rely on SVD to maintain pre-training performance. 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.

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

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

with all small ϵ in the denominators removed for simplicity.

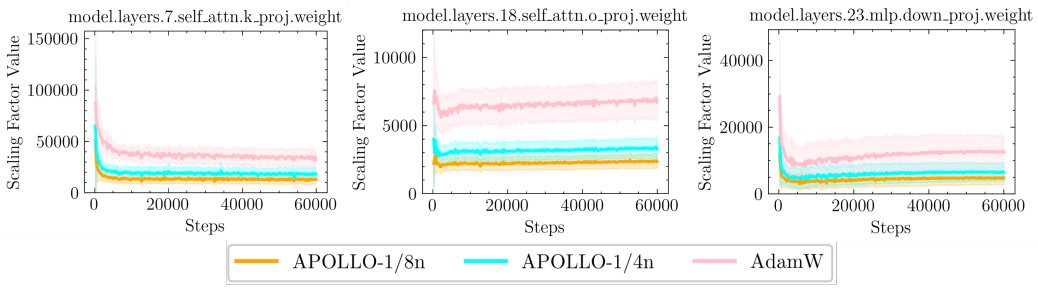

Figure 4. Visualization of the channel-wise scaling factor ratio for APOLLO with rank $1/8n$ and $1/4n$, compared with AdamW (full rank n). The empirical data aligns well with the theoretical ratios $1 : \sqrt{2} : 2\sqrt{2}$, validating the bounds across various layer types and stages on the LLaMA-350M model. More visualization can be found at Fig. 5.

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:

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

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

We quantify the approximation error in the following.

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 We now bound the difference between the gradient scaling factor in the compressed and original space based on the theorems 4.1 and 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 $\ge 1-\delta$:

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

Proof: Please refer to Appendix A.1.5.

The theorem suggests we should scale the gradient by a factor $\sqrt{\frac{n}{r}}$ to ensure consistent behavior with AdamW under structured learning rate update. Accordingly, we include a gradient scale factor α in Algorithm 1. However, since this factor can be absorbed into the learning rate, we set $\alpha = 1$ by default in `APOLLO`. When the r is too small compared to n, as in our `APOLLO-Mini` case with rank-1 space, we explicitly set $\alpha = \sqrt{128}$ to compensate.

Empiric evidence of the ratio $\sqrt{n/r}$: We empirically validate the scaling factor ratio $\sqrt{n/r}$ (Eq. 9) by comparing the original rank-n space with the compressed rank-r space using LLaMA-350M. Specifically, we evaluate `APOLLO` with r to $1/8n$ and $1/4n$ against the full-rank AdamW baseline. As shown in Fig. 4, the observed ratios align closely

with the theoretical predictions ($\sim \sqrt{1/8}$ and $\sqrt{1/4}$ respectively), confirming the bound and effectiveness of random projection. Check more details in Appendix A.2.

So far, APOLLO has been shown to be theoretically sound with random projection, which we adopt as the default. APOLLO with SVD also performs well, yet incurs significant computational overhead with only marginal improvement. We re-sample the projection matrix every T steps (200 by default) by effortlessly generating a new random seed, as fixing the matrix has been shown to be suboptimal for high-dimensional LLM training (Zhao et al., 2024).

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 approximation quality. The coarsened learning rate update rule is highly tolerant to low-rank approximations. So, APOLLO can operate effectively with a rank that is half of what GaLore requires. However, we still need $n \times r$ memory cost for the optimizer state. If we can relax the rank to 1, then the optimizer state cost is totally negligible. However, simply setting rank to 1 in APOLLO doesn't work well due to rank-1 space sacrificing too much information (details at Sec. 5.4 and Appendix *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, 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 in a rank-1 space is typically smaller than that obtained with a higher rank, which the theorem can theoretically justify in equation 9. Hence, we heuristically set a gradient scale factor α (defaulting to $\sqrt{128}$) to compensate for this difference.

4.3 Savings and Cost Analysis

Tab. 1 compares various memory-efficient training methods, including APOLLO-series, Fira (Chen et al., 2024), GaLore (Zhao et al., 2024), and Flora (Hao et al., 2024).

Notably, APOLLO purely relies on random projection, avoiding the costly SVD in Fira and GaLore. In terms of memory efficiency, APOLLO exhibits greater robustness to low rank settings compared to other methods (see Tab. 2). Additionally, it eliminates the need to store the projection matrix, requiring only a random seed. Our variant, APOLLO-Mini, achieves extreme memory efficiency by reducing optimizer states to a constant $2n + 2$, making it comparable to SGD cost, yet it retains or even surpasses AdamW performance.

Table 1. Detailed comparison between Fira, GaLore, Flora, 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	$2n$ $+2$	$2nr$ $+2$	$2nr$ $+mr+1$	$2nr$ $+mr$	$2nr$ $+1$
Full-Rank Gradients	✔	✔	✔	✘	✘
Full-Rank Weights	✔	✔	✔	✔	✔
Pre-Training	✔	✔	✔	✔	✘
Fine-Tuning	✔	✔	✔	✔	✔
w.o. SVD	✔	✔	✘	✘	✔

5 EXPERIMENTS

We evaluate the effectiveness of APOLLO through a comprehensive set of experiments. In Sec. 5.1 and 5.2, we assess APOLLO on various pre-training and fine-tuning tasks, respectively. Sec.5.3 highlights the system-level advantages of APOLLO in terms of memory usage and throughput. Sec.5.4 presents extensive ablation studies analyzing the impact of low-rank projection methods, rank selection, and scaling factor granularity, along with detailed comparisons of training dynamics. Finally, Sec. 5.5 offers preliminary insights into why a stateless APOLLO can outperform AdamW.

5.1 Memory-Efficient Pre-training with APOLLO

We show that the APOLLO-series achieves **superior pre-training performance** across various sizes of LLaMA models, with up to a 2.80↓ in validation perplexity on C4 dataset. Notably, APOLLO-Mini uses a negligible memory budget for optimization states, yet still outperforms AdamW.

Setup. We evaluate LLaMA models of various sizes, ranging from 60M to 7B. Following the training setting used in prior works (Zhao et al., 2024), we pre-train each model from scratch, with a detailed description in Appendix A.6. 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. We include the following baselines: (i) AdamW: We pre-train the models using the standard AdamW optimizer (Loshchilov & Hutter, 2019). (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) uses low-rank adapters for memory-efficient training by decompos-

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. **For a fair comparison, we keep the same training settings as GaLore/Fira, which is not tuned on the APOLLO series. We can further tune the hyperparameters, e.g., the learning rates, for optimal performance. Here, we report the APOLLO-Mini with a learning rate of 0.02 (not 0.01 in GaLore), achieving stronger results, marked with ‡.**

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	23.53	0.25G	17.18	0.69G	**14.17**	2.60G
APOLLO-Mini ‡	30.95	0.12G	22.85	0.25G	16.63	0.69G	13.95	2.60G

ing the weights as $W = W_0 + UV$, where W_0 remains frozen and only U and V are optimized with AdamW. (iv) ReLoRA: ReLoRA (Lialin et al., 2023) enhanced LoRA, specifically for pre-training, which periodically merges the low-rank adapters UV back into the original weights W. (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 adding the error residual of low-rank gradient back.

Main Results. We evaluate APOLLO and its two variants: APOLLO *w.* SVD, which uses SVD instead of random projection; and APOLLO-Mini, which uses rank-1 space and computes scaling factors in a tensor-wise manner, with negligible optimizer memory cost. Results are reported in Tab. 2, from which several observations can be made: (i) **Performance under the same memory budget**: With rank set to one-quarter of the original dimension, APOLLO consistently outperforms GaLore, achieving up to a 3.62 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. This is impressive given that vanilla SGD is known to fail in training Transformer models (Zhang et al., 2024a); (iii) **Robustness across projection methods and rank sizes**: APOLLO performs robustly regardless of projection type or rank. For exmaple, on LLaMA-350M, switching to SVD improves perplexity by only a 0.18. This indicates that APOLLO can maintain performance even without SVD, which is known for its time-consuming nature (Zhang et al., 2024c). Further halving the rank leads to negligible performance degradation, highlighting APOLLO 's efficiency even under aggressive memory constraints. A detailed analysis of the effects of projection methods and rank is in Sec. 5.4; (iv) **Comparison with Fira**: APOLLO scales better with larger models and more training tokens, consistently outperforming Fira as model size and training tokens increase, though it slightly outperforms us on smaller models (60M, 130M). An in-depth comparison of training performance across model sizes and training tokens is in Sec. 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.

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	**1.6G**	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

Scaling up to LLaMA-7B. We evaluate the pre-training of a LLaMA-7B model using AdamW, GaLore, APOLLO ($r = 256$) and APOLLO-Mini ($r = 1$) on an 8× A100 80GB setup. To ensure consistency, we maintain a total batch size of 512 per epoch across all methods, adjusting the micro-batch size based on each method's memory footprint. AdamW is limited to a micro-batch size of 4. GaLore is configured to match our memory usage with a micro-

batch size of 8. Since AdamW requires extended training time to fully train a 7B model, we allocate a fixed training time budget of half a month(**15 Days**) for a fair and practical comparison. The training curve, showing validation perplexity recorded every 1000 steps, is in Fig. 2.

Our 7B-scale experiments highlight two key benefits of the APOLLO series: (i) **Accelerated training memory savings enabling larger batch sizes.** The APOLLO series achieves substantial memory efficiency, allowing for larger batch sizes and resulting in ∼3× faster training throughput compared to AdamW, ∼2× faster than GaLore. Notably, APOLLO and APOLLO-Mini are the only methods able to complete pre-training within the half-month timeframe. (ii) **Superior model performance with best perplexity.** Despite significantly reducing optimizer overhead, APOLLO delivers the best perplexity results, even when GaLore uses a high rank of 1024. Midway through training, APOLLO surpasses GaLore in performance, marking a key crossover point that demonstrates its clear advantage. As shown in Tab.3, the APOLLO series delivers >1.5 perplexity improvement compared to the 8-bit Adam and GaLore, all while maintaining significantly lower optimizer memory usage.

Downstream Task Performance. As perplexity may not precisely assess model quality (Jaiswal et al., 2023), we further evaluate performance on a suite of commonsense and math reasoning tasks. We use LLaMA-350M models pretrained with sequence lengths of 256 and 1024, selecting the best AdamW checkpoint via learning rate sweeps. Tab.4 presents zero-shot results across multiple tasks, including BoolQ (Clark et al., 2019), RTE (Wang et al., 2018), HellaSwag (HS) (Zellers et al., 2019), Winogrande (WG) (Sakaguchi et al., 2021), OpenBookQA (OBQA) (Mihaylov et al., 2018), ARC (ARC-Easy (ARC-E), ARC-Challenge (ARC-C)) (Clark et al., 2018), PIQA (Bisk et al., 2020), SciQ (Johannes Welbl, 2017), and MathQA (Amini et al., 2019). We confirm that models pretrained with the APOLLO series not only achieve **lower perplexity** but also **outperform AdamW-trained models on downstream tasks**.

*Conclusion ①: The APOLLO series **optimizes memory usage to a SGD-like level, enhances model quality over AdamW**, and **accelerates pre-training** by enabling larger batch sizes. This makes APOLLO a highly practical and efficient solution for large-scale LLM pre-training.*

5.2 Memory-Efficient Fine-tuning with APOLLO

Pre-training large foundation models typically demands thousands of GPUs and months of training. Hence, 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 use four open-source pre-trained models for our fine-tuning evaluation: LLaMA-3.2-1B, LLaMA-3-8B, Gemma-7B, and Mistral-7B. The downstream tasks are divided into two categories: (i) Eight common-sense reasoning tasks: WG, PIQA, SIQA (Sap et al., 2019), OBQA, HS, BoolQ, and ARC-E and ARC-C; (ii) MMLU (Hendrycks et al., 2020) tasks across various domains: STEM, Social Sciences, Humanities and others. Details at Appendix A.7.

Baseline. We compare APOLLO against several baselines, including full-rank AdamW, LoRA, GaLore, and Fira. 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. APOLLO-Mini uses a rank of 1.

Main Results. As shown in Tab. 5 and Tab. 6, APOLLO consistently matches or outperforms other baselines, achieving up to a 1.01 average accuracy improvement over full-rank AdamW on commonsense reasoning tasks, while maintaining comparable performance on MMLU tasks. Notably, APOLLO requires only a rank-32 space for optimizer states, and APOLLO-Mini uses a rank of 1—resulting in negligible optimizer memory cost.

*Conclusion ②: The APOLLO series establishes itself as a compelling **memory-efficient full-parameter** fine-tuning method, delivering **on-par or better performance compared to AdamW**.*

5.3 System-level Benefits

End-to-end system-level benefits: We evaluate the end-to-end training throughput and memory usage by running LLaMA-7B on 8× A100 80GB GPUs. Fig. 1 compares APOLLO and APOLLO-Mini against AdamW, highlighting two key benefits: **(1) Much lower memory usage:** With a batch size of 4, AdamW already reaches the memory limit, consuming approximately 79 GB. In contrast, APOLLO and APOLLO-Mini require only 70 GB and 68 GB, respectively, even with a batch size of 16; **(2) Higher throughput:** The APOLLO series achieves significantly higher throughput than AdamW, with up to 3× improvement in training throughput. This is enabled by the substantial memory savings, which allow the batch size to scale up to 4× larger than that of AdamW.

These results demonstrate that AdamW not only incurs high memory costs but also limits training efficiency by becoming memory-bound, preventing full utilization of available compute. In contrast, APOLLO enables larger batch sizes and better hardware utilization, accelerating large-scale training with even better performance.

Negligible optimizer overhead. We further evaluate the optimizer step time of AdamW, the APOLLO series, GaLore,

Table 4. Zero-shot performance of LLaMA-350M models pretrained with AdamW and APOLLO-series on commonsense and math reasoning tasks.

Sequence Length: 256													
Method	Memory	Perplexity	BoolQ	RTE	HS	WG	OBQA	ARC-E	ARC-C	PIQA	SciQ	MathQA	Average
AdamW	1.37G	18.80	0.5881	0.4729	0.3286	0.5335	0.304	0.3615	0.2167	0.6387	0.591	0.2047	0.3554
APOLLO	0.34G	16.85	0.5165	0.4729	0.3528	0.5146	0.318	0.3792	0.2517	0.6632	0.592	0.2188	**0.3681**
APOLLO-Mini	0.00G	17.18	0.5434	0.4729	0.3481	0.5162	0.320	0.3653	0.2474	0.6469	0.591	0.2178	**0.3654**
Sequence Length: 1024													
Method	Memory	Perplexity	BoolQ	RTE	HS	WG	OBQA	ARC-E	ARC-C	PIQA	SciQ	MathQA	Average
AdamW	1.37G	16.30	0.4917	0.4693	0.3688	0.5233	0.332	0.3729	0.2449	0.6534	0.609	0.2064	0.3712
APOLLO	0.34G	15.64	0.5373	0.4693	0.3850	0.4925	0.322	0.3788	0.2483	0.6681	0.624	0.2127	**0.3840**
APOLLO-Mini	0.00G	16.12	0.5376	0.4693	0.3707	0.5217	0.324	0.3758	0.2312	0.6638	0.619	0.2224	**0.3785**

Table 5. 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**

and Fira, in Tab. 7. We benchmark LLaMA-1B and 7B with a sequence length of 1024, using batch sizes of 16 and 4, respectively, the largest batch sizes that AdamW supports.

Unlike GaLore and Fira, which suffer from expensive SVD updates (taking $\sim$10 minutes for the 7B model), **the APOLLO series incurs minimal overhead** by relying only on cheap random projections. While APOLLO introduces an additional step to compute the scaling factor and applies NL, *projecting gradients into a low-rank space reduces the overhead of maintaining first-order and second-order moments.* Notably, on the 7B model, the APOLLO series achieves even faster optimizer step times than AdamW. Overall, the optimizer step overhead remains negligible compared to the backward pass, particularly for larger batch sizes, ensuring that APOLLO enables superior memory efficiency while being a lightweight solution.

APOLLO-Mini enables LLaMA-13B pre-training 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 on A100 80GB GPU with naive DDP, without requiring other system-level optimizations (e.g. 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.

Pre-training LLaMA-7B under 12 GB with weight quantization. As our methods significantly reduce optimizer memory costs, model weights become the next major memory bottleneck. To further address this, we integrate our approach with the Int8 weight quantization method proposed in Q-GaLore (Zhang et al., 2024c), enabling even greater memory savings. As shown in Table 8, our Q-APOLLO series minimizes memory consumption across both optimizer and weight components while maintaining pre-training perplexity on par with—or better than—full-precision AdamW. Notably, Q-APOLLO also achieves a clear performance advantage over Q-GaLore, underscoring its superiority in both memory efficiency and model quality. By 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 marks a major breakthrough, making LLM pre-training feasible on low-end GPUs and democratizing access to LLM training for a broader audience.

*Conclusion ③: The APOLLO series **significantly reduces optimizer memory usage** with **minimal compute overhead**, enabling **higher throughput**, **improved model scalability**, and **more friendly low-end GPU training** for LLMs. APOLLO takes a pivotal step toward **democratizing LLM training**, making large-scale model training more accessible and efficient.*

5.4 Extra Investigation and Ablation Study

This section presents several experimental investigations focused on five key research questions:

Table 6. Comparison results of various memory-efficient fine-tuning algorithms on MMLU tasks. For Galore, Fira, APOLLO, and APOLLO-Mini, we report the best accuracy obtained by sweeping the learning rate within the range [5e-6, 7.5e-6, 1e-5, 2.5e-5, 5e-5, 7.5e-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	54.50	75.11	58.59	72.03	64.43
	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	54.37	75.86	58.18	71.69	64.35
	APOLLO-Mini	54.40	75.37	58.72	71.59	64.41
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	29.03	35.27	32.40	36.52	33.26
	APOLLO *w. SVD*	29.20	40.42	32.40	38.94	**34.98**
	APOLLO	27.53	36.97	33.99	36.40	33.81
	APOLLO-Mini	27.30	33.83	31.61	33.77	31.67
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	51.87	72.82	54.94	69.49	61.56
	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	51.63	73.12	54.90	69.58	61.58
	APOLLO-Mini	51.97	72.89	54.43	69.18	61.35

Table 7. Optimizer step time (in seconds) across LLaMA-1B and LLaMA-7B with a sequence length of 1024 on a single A100 GPU. Results are averaged over 400 steps, with 100 warm-up steps and low-rank projection matrices updated every 200 steps. Batch sizes are set to 16 (1B) and 4 (7B), the maximum batch size supported by AdamW. Lower values indicate less overhead.

Model Size	Bwd Time(s)	Optimizer Step Time (s)				
		AdamW	APOLLO	APOLLO-Mini	GaLore	Fira
1B	1.069	**0.036**	0.051	0.048	0.371	0.421
7B	0.712	0.173	0.159	**0.142**	2.874	3.086

Q1: How can the scaling factor subspace be identified? **Short answer:** Random Projection performs well with no need for SVD. *More details in Appendix A.3.1.*

Q2: Is APOLLO sensitive to the rank? **Short answer:** APOLLO is largely insensitive to low rank settings, and APOLLO-Mini remains effective even with rank 1. *More details in Appendix A.3.2.*

Q3: What is the appropriate granularity for scaling factors? **Short answer:** Channel-wise granularity is preferred for higher ranks (APOLLO), while rank-1 settings prefers tensor-level scaling (APOLLO-Mini). *More details in Appendix A.3.3.*

Q4: How does the performance of different methods evolve during training? **Short answer:** The APOLLO series outperforms others as model size and training tokens increase. *More details in Appendix A.3.4.*

Q5: How does APOLLO perform in long-context training settings? **Short answer:** The APOLLO series performs on par with, or even better than, AdamW in long-context scenarios. *More details in Appendix A.3.5.*

Table 8. 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

5.5 Extra Insights on Why a Stateless Optimizer Can Beat AdamW in Pre-training

We provide preliminary insights in Appendix A.4 on why a stateless APOLLO can surpass AdamW, and we leave a formal one as future work.

6 Conclusion

In this paper, we introduced a novel approach for training LLMs that strikes an effective balance between memory efficiency and performance. Motivated by the limitations of existing methods like GaLore, which rely on 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 AdamW baseline with greater memory saving than GaLore. APOLLO-Mini further squeezes the memory cost by using a rank-1 sub-space, achieving better performance than AdamW 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.

Acknowledgements

Meta supported H. Zhu during his internship. D. Pan is in part supported by the NSF AI Institute TILOS (award number 2112665) and an equipment donation from Nvidia. Z. Wang is in part supported by NSF Awards 2145346 (CAREER), 2133861 (DMS), 2113904 (CCSS), and the NSF AI Institute for Foundations of Machine Learning (IFML) We extend our heartfelt gratitude to Yuandong Tian for critical discussions.

REFERENCES

Amini, A., Gabriel, S., Lin, P., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. Mathqa: Towards interpretable math word problem solving with operation-based formalisms, 2019.

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*.

Jaiswal, A., Gan, Z., Du, X., Zhang, B., Wang, Z., and Yang, Y. Compressing llms: The truth is rarely pure and never simple. *arXiv preprint arXiv:2310.01382*, 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.

Johannes Welbl, Nelson F. Liu, M. G. Crowdsourcing multiple choice science questions. 2017.

Keskar, N. S. and Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

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.

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.

Pan, Y. and Li, Y. Toward understanding why adam converges faster than sgd for transformers. *arXiv preprint arXiv:2306.00204*, 2023.

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.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

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.

Zhou, P., Feng, J., Ma, C., Xiong, C., Hoi, S. C. H., et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296, 2020.

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 Empirical validation of the derived bound in Theorem A.4

In this part, we present a visualization of the scaling factor ratio $\sqrt{n/r}$ derived in Theorem A.4. The plot demonstrates how the ratio adheres to the theoretical bounds under various rank settings, providing empirical support for the theorem.

Here, we consider the following variants:

- **AdamW with the same structured channel-wise learning rate adaptation rule:** This variant uses a full rank n and serves as the golden standard for estimating s_j, the scaling factor.

- **APOLLO with rank r:** This variant computes a low-rank approximated version of the scaling factor, s_j^R, which should theoretically be $\sqrt{n/r}$ times smaller than s_j.

We visualize the channel-wise scaling factor on the LLaMA-350M model [1], comparing APOLLO with ranks $1/8n$ and $1/4n$. These configurations should yield scaling factor ratios of approximately $\sqrt{1/8}$ ($\sim$ 0.35) and $1/2$, respectively, relative to the full-rank AdamW.

As shown in Fig. 5, the scaling factor ratio adheres closely to the theoretical predictions across different layer types (e.g., attention, MLP) and model stages (e.g., early, middle, or late layers).

A.3 Ablation Study

A.3.1 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 Fig. 6 (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.

[1] To ensure consistent optimization trajectories across the variants, we use the same learning rate as APOLLO with rank $1/4n$. Additionally, we scale the final gradient updates using the heuristic ratio derived from the rank settings relative to $1/4n$.

A.3.2 A2: 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 Fig. 6 (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 mitigate the errors introduced by low-rank projection. This capability allows APOLLO-Mini to achieve SGD level memory cost while reaching superior performance than AdamW.

Table 9. 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

A.3.3 A3: The gradient scaling factor can even be calculated at the tensor level.

In Tab.9, we compare the pre-training 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 Fig. 6 (d).

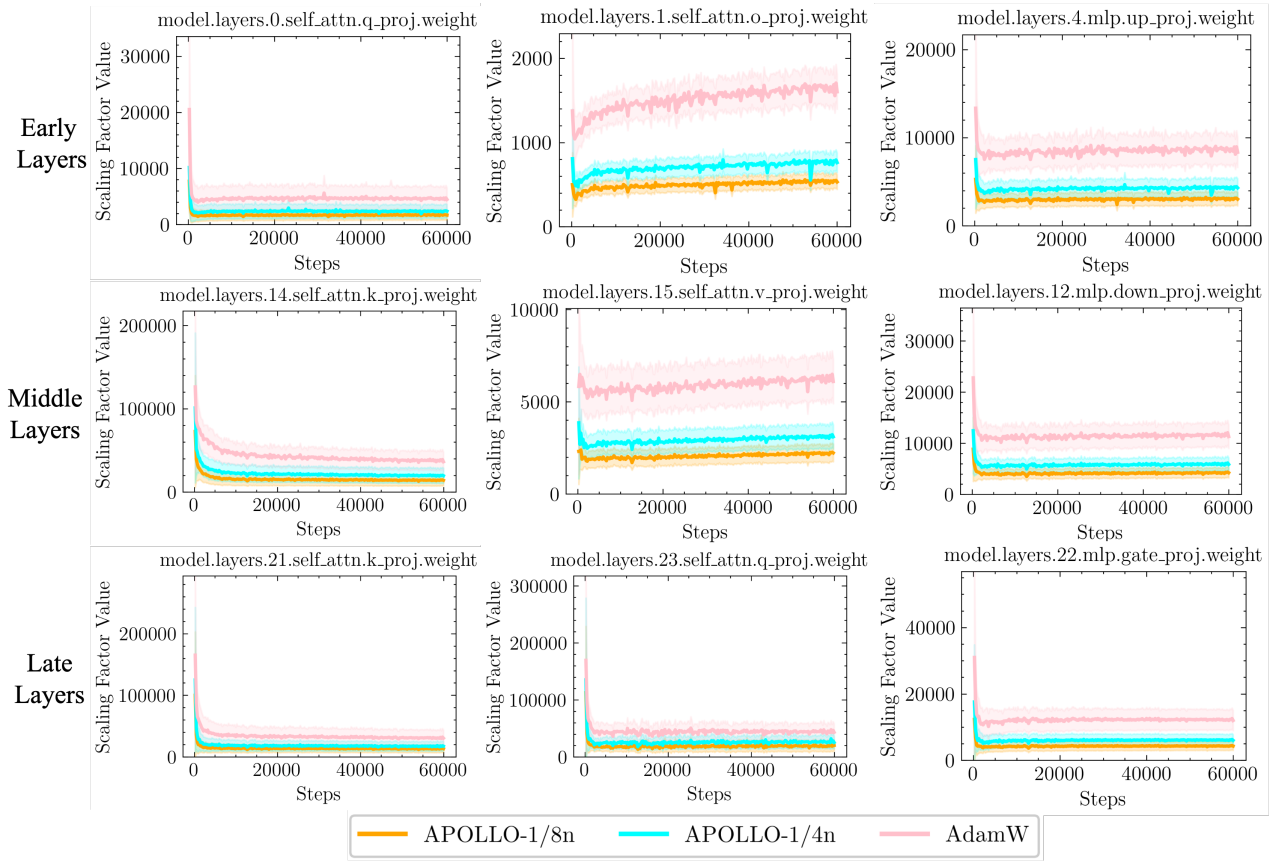

Figure 5. Visualization of the channel-wise scaling factor ratio for APOLLO with rank $1/8n$ and $1/4n$, compared with AdamW (full rank n). The empirical data aligns well with the theoretical ratios $1 : \sqrt{2} : 2\sqrt{2}$, validating the bounds across various layer types and stages on the LLaMA-350M model.

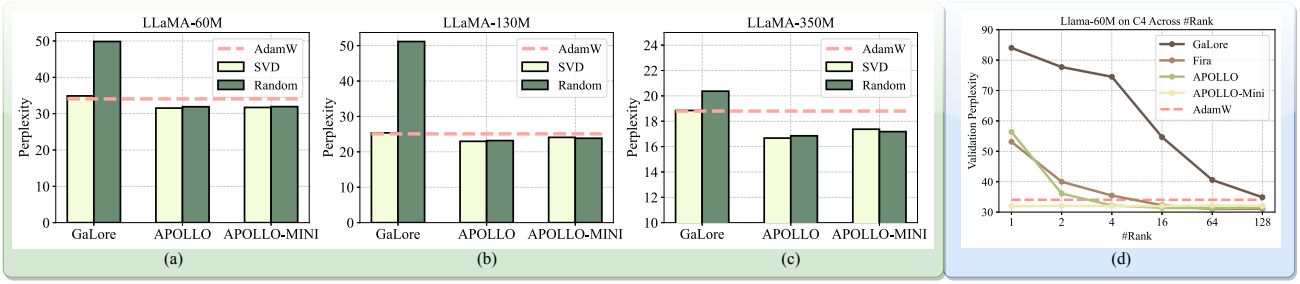

Figure 6. (a-c) Comparison results of various optimization methods using singular value decomposition or random projection. The experiments were conducted on LLaMA-60M/130M/350M models for C4 pretraining tasks. (d) Validation perplexity with varying rank sizes, where 128 is one-quarter of the original model dimension. The red dashed line indicates the performance of full-rank AdamW.

A.3.4 A4: `APOLLO` performs better with larger model sizes and more training tokens.

Fig. 7 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, Fig. 7 indicates that `APOLLO` seem to benefit from increased training tokens. To quantify this effect, we pre-trained LLaMA-130M models for $\{20\text{k}, 30\text{k}\}$ steps, with final perplexities

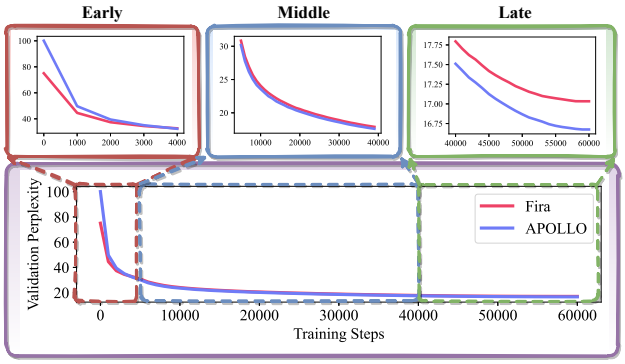

Figure 7. Validation perplexity of pretraining LLaMA-350M on the C4 dataset, with zoomed-in figures showing early, middle, and late stages of training at top, with full training period at bottom.

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, Tab.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.

A.3.5 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-3, 2.5e-3, 5e-3, 7.5e-3, 1e-2]. 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 $[\sqrt{128}, \sqrt{256}, \sqrt{384}]$, under a fixed learning rate 1e-2.

As shown in Fig. 8, 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.

A.4 Extra Insights on Why a Stateless Optimizer Can Beat AdamW in Pre-training

We provide preliminary insights on why a stateless APOLLO can surpass AdamW in certain scenarios, and we leave a formal one as the future work.

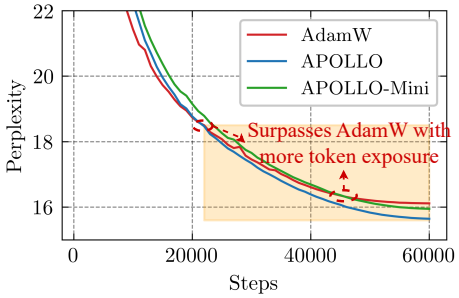

Figure 8. Perplexity curves of the LLaMA-350M model trained in a long-context window setting. APOLLO and APOLLO-Mini outperform AdamW with a grid-searched learning rate, demonstrating the effectiveness of the APOLLO series in industrial LLM pre-training settings(long sequences and extensive training tokens).

Adam(W) applies $\tilde{\mathbf{G}}_t = \frac{\mathbf{M}_t}{\sqrt{\mathbf{V}_t}+\epsilon}$, which can be viewed as scaling the raw gradient $\mathbf{G}_t$ by a scaling matrix $\mathbf{S} = \frac{\tilde{\mathbf{G}}_t}{\mathbf{G}_t}$. APOLLO observes that this fine-grained, parameter-wise scaling $\mathbf{S}$ can be approximated at the channel or tensor level, validated in Fig. 3. Although coarser, this scaling largely preserves the original gradient direction, (e.g., $\mathbf{G}_t s^R$ in APOLLO-Mini) and thus behaves similarly to SGD. Such an "SGD-like" update depends more on the current gradient and injects greater randomness during training, **enhancing the ability to escape local optima** and yielding **better generalization performance**(Zhou et al., 2020; Keskar & Socher, 2017). This explains why APOLLO series can surpass AdamW, especially at later training stages (when generalization becomes critical) and for larger models (with more complex loss landscapes). Key observations supporting this claim include:

- In Sec. 3.2, Fig. 3, the structured AdamW (APOLLO-style update rule) underperforms AdamW initially but eventually surpasses it.
- In Sec. 5.4 (Ablation A.3.4), APOLLO typically outperforms AdamW at later stages of training.

Why APOLLO Resembles SGD Yet Performs Well for LLM Training? While SGD is associated with stronger generalization, it often struggles with Transformer training (Pan & Li, 2023; Zhang et al., 2024a). APOLLO reconciles SGD's generalization benefits with AdamW's convergence speed, as illustrated by the following two hypotheses (Pan & Li, 2023; Zhang et al., 2024a).

Hypothesis 1: Directional Sharpness (Pan & Li, 2023) A key finding in (Pan & Li, 2023) is that Adam achieves lower *directional sharpness* than SGD, thereby improving Transformer training. The directional sharpness of f at x along direction v (with $\|v\|_2 = 1$) is $v^\top \nabla^2 f(x) v$. Lower

Table 10. Directional sharpness comparison across different optimizers.

Epoch	SGD	Adam	APOLLO	APOLLO-Mini
2	1.959722	0.009242	0.006024	0.004017
5	1.512521	0.000509	0.000249	0.000107
10	2.471792	0.000242	0.000163	0.000056
20	3.207535	0.000399	0.000261	0.000101

directional sharpness implies the possibility of taking larger effective steps, potentially yielding a greater local decrease in the objective. In contrast, if the directional sharpness is large, we have no choice but to take a tiny step, as otherwise the loss would blow up due to the second-order term.

Empirical tests on APOLLO/APOLLO-Mini (using a small T5 model for a machine translation task following (Pan & Li, 2023)) show significantly reduced sharpness relative to SGD and comparable to or better sharpness than Adam(W) (see Tab. 10). This provides a theoretical underpinning for APOLLO 's effectiveness in LLM training.

Hypothesis 2: Adaptive Learning Rates for Transformers (Zhang et al., 2024a) Transformer blocks display varying Hessian spectra, suggesting that block-wise adaptive learning rates are advantageous (Zhang et al., 2024a), which can render naive SGD less suitable. However, fully parameter-wise adaptive learning rates (as in AdamW) can be redundant, as shown in Adam-Mini (Zhang et al., 2024a), which replaces the second-order moment with group-wise averaging—thereby reducing optimizer memory usage by up to 50%.

APOLLO applies adaptive learning rates in a structured channel-/tensor-wise manner and goes beyond Adam-Mini by reducing memory usage for both first- and second-order moments, even eliminating optimizer memory in APOLLO-Mini.

A.5 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. 9. 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.

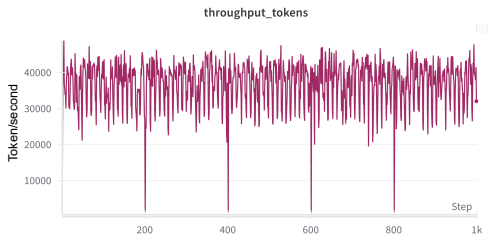

Figure 9. The training throughput of Galore-type low-rank optimizer with many spikes due to the expensive SVD operation every 200 steps.

A.6 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). Tab. 11 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.7 Detailed Fine-Tuning Setting

A.7.1 Commonsense reasoning fine-tuning

We use the implementation from (Liu et al., 2024a) with the chosen hyperparameters detailed in Table 12.

A.7.2 MMLU fine-tuning

We use the implementation from (Zheng et al., 2024). We adopt the implementation from (Zheng et al., 2024). For a fair and comprehensive comparison, we set the rank to 8 and sweep the learning rate across the range [5e-6, 7.5-6, 1e-5, 2.5e-5, 5e-5, 7.5e-5, 1e-4, 1.5e-4, 2e-4] for GaLore, Fira, APOLLO, and APOLLO-Mini. Specifically, APOLLO-Mini uses a scaling factor of $\sqrt{4}$ for fine-tuning LLaMA-3-8B and Mistral-7B, while a factor of 1 is applied

Table 11. 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

to Gemma-7B, as it exhibits higher sensitivity during fine-tuning. The full fine-tuning and LoRA results are taken from (Zhang et al., 2024c).

A.8 Discussion with Fira (Chen et al., 2024)

As suggested by the authors of concurrent work Fira (Chen et al., 2024), our channel-wise approximation of the element-wise gradient scaling rule $\frac{\tilde{\mathbf{G}}_t}{\mathbf{G}_t}$ shares a similar format the scaling factor in the Fira, which is used for normalizing the error residual between low-rank GaLore and full-rank gradients. While our approach shares a similar mathematical form, *as being a straightforward computation of ℓ_2-norm ratios*, it originates from a fundamentally distinct perspective. We argue that the element-wise gradient scaling rule in equation 2 is unnecessarily fine-grained and can be effectively replaced with structured *channel-wise* or *tensor-wise* adaptation. In contrast, Fira seeks to normalize the error residual between low-rank GaLore updates and full-rank updates based on the observation that channel-wise gradient norm ratios between low-rank and full-rank optimizers are inherently similar. Our method, however, establishes a different finding: the low-rank approximated channel-wise gradient scaling factor, $\frac{\tilde{\mathbf{G}}_t}{\mathbf{G}_t}$, follows a predictable ratio of $\sqrt{r/n}$ (see Theorem A.4) compared to full-rank optimization, which differs fundamentally from Fira's observations.

B Artifact Appendix

B.1 Abstract

APOLLO introduces a memory-efficient optimizer designed for large language model (LLM) pre-training and full-parameter fine-tuning, for the first time offering SGD-like memory cost with AdamW-level performance based on only cheap random projection. APOLLO-Mini is an extremely memory-efficient version of APOLLO, which uses a rank of 1 but uses tensor-wise scaling instead of channel-wise scaling in APOLLO.

Our artifact contains the complete source code for APOLLO and key experimental scripts to validate APOLLO's effectiveness on LLM pre-training and fine-tuning as well as system level benefits, i.e., throughput and memory saving.

The code and artifact are accessible at GitHub.

Our APOLLO has been integrated into Hugging Face Transformers and LLaMA-Factory. Welcome to try our APOLLO in their code framework as well following their instruction.

B.2 Artifact check-list (meta-information)

- **Algorithm:** APOLLO, a memory-efficient optimizer.
- **Program:** Python.
- **Data set:**
  - **Pre-training:** C4 dataset (Raffel et al., 2020) — a comprehensive corpus derived from Common Crawl data, meticulously filtered and cleaned.
  - **Finetuning:** MMLU (Hendrycks et al., 2020) task.
- **Run-time environment:**
  - Python, PyTorch, transformers, bitsandbytes.
  - Please refer to the minimal packages and minimal experimental packages for details.
- **Hardware:**
  - The minimal LLM pre-training example (LLaMA-60M) requires at least one Nvidia A6000 GPU (48GB) for 3 hours.
  - Our code has been tested on Nvidia A6000 (48GB) and A100 (80GB).
- **Experiments:** We prepared two main suites of experiments to evaluate that APOLLO is *functional* and *avaliable*:
  - **Memory-efficient LLM Pre-training:** Use the code base available on our GitHub.
  - **Memory-efficient full-parameter LLM Fine-tuning:** Use the code base using LLaMA-Factory, which support APOLLO natively.

  Additionally, scripts are provided to demonstrate extreme memory efficiency:
  - Pretraining a LLaMA-7B model within 12GB memory (runnable on Nvidia Titan GPU).
- **How much disk space required (approximately)?:** 100 GB.
- **How much time is needed to prepare workflow (approximately)?:** 1 hour.
- **How much time is needed to complete experiments (approximately)?:**

Table 12. 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	$\sqrt{5}$	$\sqrt{128}$
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							

  - One minimal pre-training example with LLaMA-60M: 3 hours on a single A6000, 1 hour in 4 A6000.
  - One fine-tuning example with LLaMA-8B: 3 hours.

- **Publicly available?:** Yes. Available at the GitHub Repo or via PyPI.
- **Code licenses (if publicly available)?:** CC-BY-NC.
- **Archived (provide DOI)?** https://doi.org/10.6084/m9.figshare.28558319.v1.

B.3 Description

B.3.1 How delivered

The artifact is delivered via the GitHub Repo or directly from PyPI. We provide the source codes and essential scripts to replicate our main results.

Alternatively, you can use `APOLLO` within the frameworks of Hugging Face Transformers and LLaMA-Factory, where `APOLLO` is natively integrated and supported.

B.3.2 Hardware dependencies

At least one GPU is required for minimal LLM training and fine-tuning examples (tested on NVIDIA A6000 and A100). Moreover, `APOLLO` enables training a 7B model on an NVIDIA Titan—demonstrating, for the first time, the capability to run large-scale models without any system-level optimizations such as offloading techniques.

B.3.3 Software dependencies

The artifact is implemented in Python and requires several packages. Please refer to the minimal packages and minimal experimental packages for details.

B.3.4 Data sets

You can use the streaming mode of the C4 dataset without the need to download it locally (the full dataset is large, 500GB). The finetuning dataset becomes available once you set up LLaMA-Factory.

B.4 Installation

Our code and scripts are available at the GitHub Repo, which includes detailed instructions for installation.

You can install the `APOLLO` optimizer either from the source:

```
git clone https://github.com/zhuhanqing/APOLLO.git
cd APOLLO
pip install -e .
```

or directly from pip:

```
pip install apollo-torch
```

Moreover, our `APOLLO` has been integrated into Hugging Face Transformers and LLaMA-Factory. You can directly try `APOLLO` within their frameworks by installing their up-to-date versions.

B.5 Experiment workflow

Please check the detailed usage for `APOLLO`-series (`APOLLO` and `APOLLO-Mini`) optimizer with hyperparameter setting.

We provide the following essential experiment scripts to replicate our method's results, which can be obtained by clone our GitHub Repo and install required packages following the repo guide.

Exp1: Pre-train LLaMA on C4 dataset

We provide the scripts in `scripts/benchmark_c4` for pre-training LLaMA models with sizes ranging from 60M to 7B on the C4 dataset.

You can also run LLM pre-training with a long context window by following the scripts in `scripts/benchmark_c4_long_context`, which compare Adam, `APOLLO`, and `APOLLO-Mini`.

Minimal example: The minimal example is provided in `scripts/pretrain_c4/llama_60m_apollo.sh` `scripts/pretrain_c4/llama_60m_apollo_mini.sh`,

which can be executed on a single GPU (e.g., A100 or A6000).

Expected outputs: The perplexity results should be similar to the reported results in Tab. 2, with possibly a slight variance.

Exp2: Pre-Train a LLaMA-7B on Nvidia Titan with 12GB memory!!!

We provide the script in `scripts/single_gpu` for pre-training a LLaMA-7B model on a single GPU with a batch size of 1. This configuration allows pre-training within 11GB of memory without any complicated system-level optimizations, such as sharding or offloading, marking the first demonstration of this capability.

Exp3: Memory-efficient full-parameter LLM finetuning

We provide the finetuning experiment examples directly under the widely-used LLaMA-Factory with their direct support.

Please first install LLaMA-Factory according to their Installation guide.

The fine-tuning experiments are done inside LLaMA-Factory repo by cloning their repo from Github, which contains the official test examples in the `examples/extras/apollo` directory.

We provide a demo to perform a comparative evaluation with GaLore by fine-tuning models and testing on the MMLU task.

Use `llamafactory-cli train examples/extras/galore/llama3_full_sft.yaml` to fine-tune llama3-8B with GaLore.

Use `llamafactory-cli train examples/extras/apollo/llama3_full_sft.yaml` to fine-tune llama3-8B with APOLLO.

Since LLaMA-Factory does not provide evaluation scripts directly, please copy the `eval_llama3_full_sft.yaml` file from our repository. And put them under corresponding directory, `examples/extras/METHOD/`. `METHOD` is `apollo` or `galore`. Then run

```
llamafactory-cli eval
examples/extras/METHOD/eval_llama3_full_sft.yaml
```

to get fine-tuned model performance.

Expected outputs:

GaLore Performance:

```
Average: 64.96
           STEM: 55.43
Social Sciences: 75.66
     Humanities: 59.72
          Other: 71.25
```

APOLLO Performance (Scaling Factor = 32):

```
Average: 65.03
           STEM: 55.47
Social Sciences: 76.15
     Humanities: 59.60
          Other: 71.28
```

Besides performance, you can observe that `APOLLO` is significantly faster than GaLore without stall issue, since `APOLLO` does not require Singular Value Decomposition (SVD), eliminating the SVD delays commonly encountered when using GaLore.

You will not observe significant memory saving between GaLore and `APOLLO` since they use the same rank during fine-tuning.

Exp4: 7B-scale throughput improvement via memory efficiency

Due to the requirement of high-end GPUs like the 8xA100 to run large-scale pre-training experiments (e.g., LLaMA-7B/13B), we provide a series of videos available at Videos, allowing you to inspect the memory cost and throughput (replicate Experiment in our Fig. 1 (right)).

You can also run the llama-7B experiment by yourself using the 7B scripts in `scripts/benchmark_c4` with `APOLLO` and `APOLLO-Mini` at a batch size of 16. However, AdamW can only run at a batch size of 4 due to excessive optimizer memory cost. (Need A100-80GB)

Exp5: `APOLLO` can use extreme low rank

One interesting customization is to safely reduce the `rank` in `APOLLO` by a certain ratio and compensate by adjusting the `apollo_scale`, which will yield similar pre-training performance. This demonstrates that `APOLLO` can operate at very low ranks without a performance penalty, achieving SGD-like memory efficiency—unlike previous methods (e.g., GaLore) that require a higher rank to maintain performance.

For example, you can set the `rank` in `scripts/pretrain_c4/llama_130m_apollo.sh` from 192 to 48, and set `apollo_scale` from 1 to 4. The model perplexity remains similar. For your reference, LLaMA-130M has a model dimension of 768; using a rank of 48 corresponds to using only $\frac{48}{768} = \frac{1}{16}$ of the full rank, leading to neligble optimizer memory cost.

This phenomenon is theoretically proved in Appendix A.1.5 and empirically observed in Appendix A.2.

B.6 Evaluation and expected result

The expected results should match those reported in the experimental section, including similar perplexity scores and performance metrics under comparable configurations.

B.7 Experiment customization

You can experiment with different configurations of `APOLLO` by following the detailed usage instructions.

B.8 Methodology

Submission, reviewing, and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging