# 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 $\ell_2$ norm of a vector.
- $\|\|_1$ indicates $\ell_1$ 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 $\chi_r^2$ 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 $\chi_r^2$, 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: $\ell_1$ Norm of Channel-wise Second Moment.** Then, we can obtain the $\ell_1$ 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 $\ell_2$ 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 $\ell_1$ 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 $\ell_1$ 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 $\delta$:

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

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).

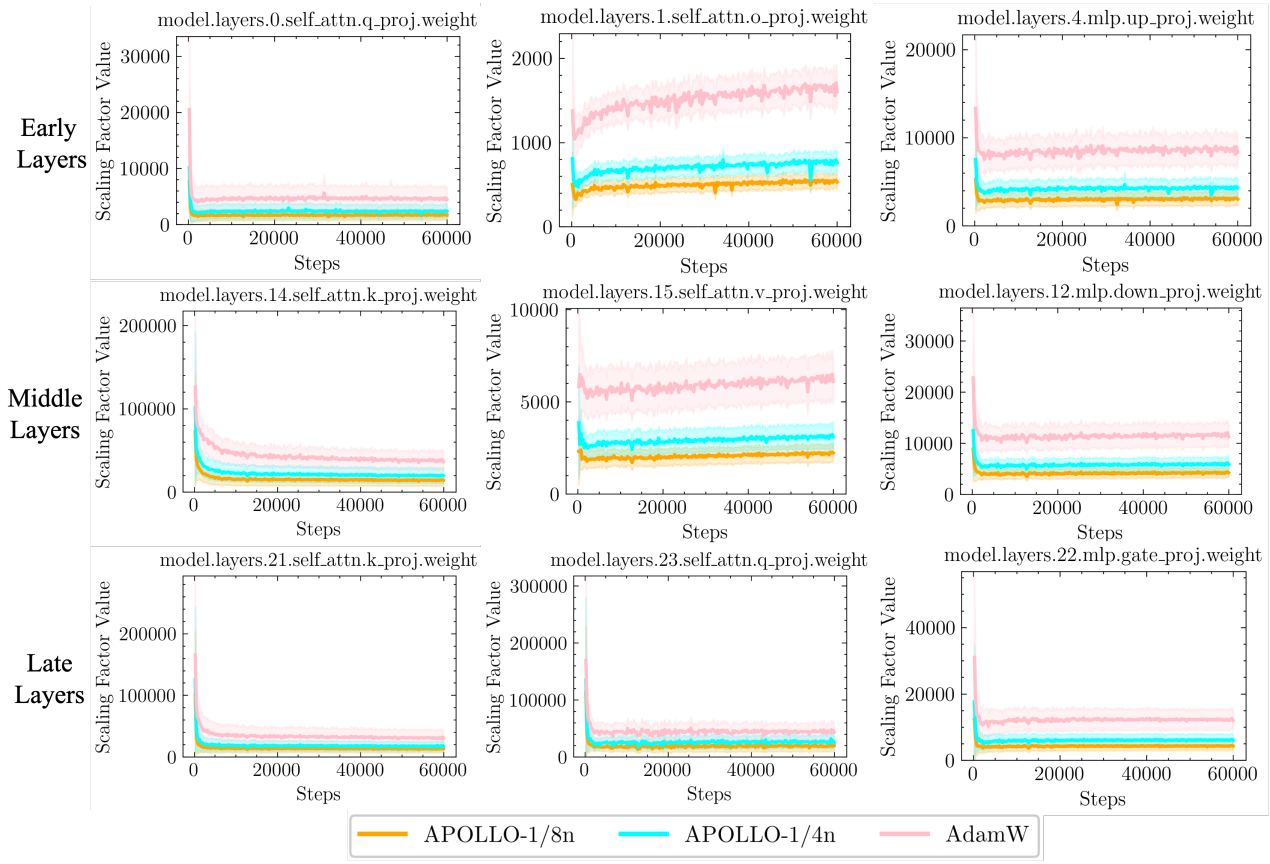


*Figure 5.* Visualization of the channel-wise scaling factor ratio for APOLLO with rank $1/8n$ and $1/4n$, compared with AdamW (full rank $n$). The empirical data aligns well with the theoretical ratios $1 : \sqrt{2} : 2\sqrt{2}$, validating the bounds across various layer types and stages on the LLaMA-350M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

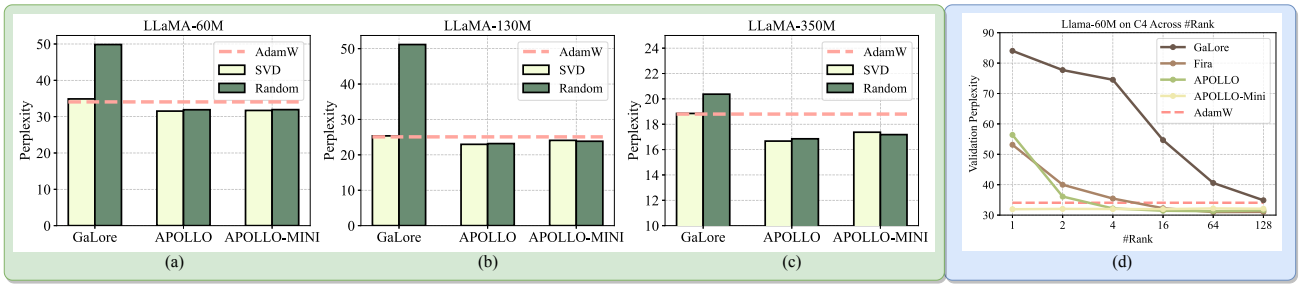


*Figure 6.* (a-c) Comparison results of various optimization methods using singular value decomposition or random projection. The experiments were conducted on LLaMA-60M/130M/350M models for C4 pretraining tasks. (d) Validation perplexity with varying rank sizes, where 128 is one-quarter of the original 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 {20k, 30k} steps, with final perplexities

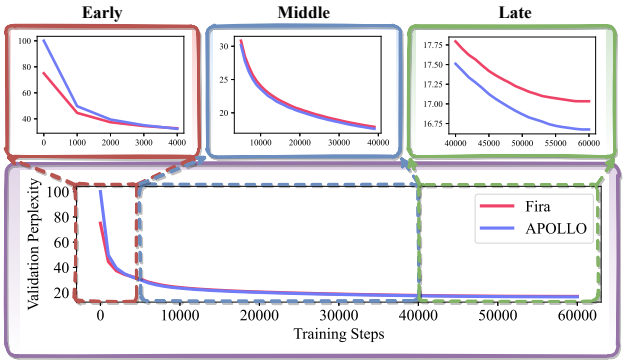


*Figure 7.* Validation perplexity of pretraining LLaMA-350M on the C4 dataset, with zoomed-in figures showing early, middle, and late stages of training at top, with full training period at bottom.

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.

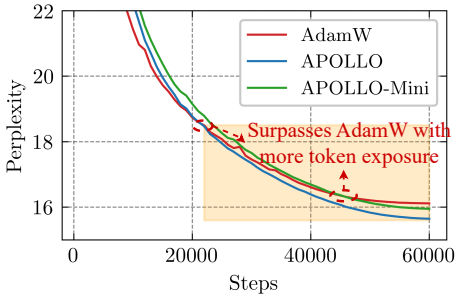


*Figure 8.* Perplexity curves of the LLaMA-350M model trained in a long-context window setting. APOLLO and APOLLO-Mini outperform AdamW with a grid-searched learning rate, demonstrating the effectiveness of the APOLLO series in industrial LLM pre-training settings(long sequences and extensive training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

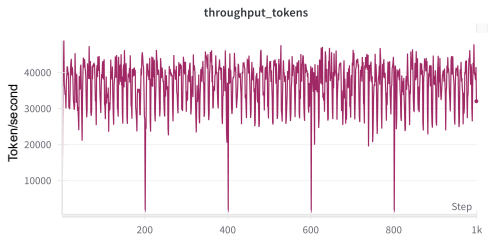


*Figure 9.* The training throughput of Galore-type low-rank optimizer with many spikes due to the expensive SVD operation every 200 steps.

### 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 $\alpha$ 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 $\alpha$ 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 $\ell_2$-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 |
| $\alpha$ | - | 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