PoLoRA: A Preconditioned Orthogonalized LoRA Optimizer

Anonymous authors
Paper under double-blind review

Abstract

Low-rank adaptation (LoRA) reduces the cost of finetuning large language models by training low-rank updates instead of the full weight matrices. Each update is the product of two trainable matrix factors. These factors are usually trained with Adam, which effectively treats them as a single flat vector of parameters and ignores the matrix and product structure of LoRA. Applying a matrix-aware optimizer such as Muon to each factor does not consistently improve over Adam, and neither do the recently proposed product-aware variants of Muon. We introduce PoLoRA (Preconditioned Orthogonalized LoRA), a product-aware spectral optimizer that adds curvature preconditioning and controls the size of each factor update. We evaluate PoLoRA on instruction-tuning datasets for code and math across models from 1B to 8B parameters, and find that it reaches the final held-out loss achieved by tuned Adam with a 1.2–1.7$\times$ speedup in training steps and at most 3% per-step overhead. Compared to Adam, PoLoRA is also less sensitive to the learning rate, and its optimal learning rate is stable across ranks.

1 Introduction

Deployment of large-scale pretrained models increasingly depends on adapting a general model to many specialized tasks [6]. Finetuning a separate full copy for each task is expensive, because training must hold gradients and optimizer state for the whole model, and deploying the finetuned models means storing and serving a full-sized model for every task. Low-rank adaptation (LoRA [20]) avoids both issues by freezing the pretrained *base weight* $\boldsymbol{W}_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and adding a small low-rank update $\Delta \boldsymbol{W}$,

$$\boldsymbol{W} = \boldsymbol{W}_0 + \Delta \boldsymbol{W}, \qquad \Delta \boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}, \qquad \boldsymbol{B} \in \mathbb{R}^{d_{\text{out}} \times r}, \quad \boldsymbol{A} \in \mathbb{R}^{r \times d_{\text{in}}}, \tag{1}$$

to each layer, where $r \in \mathbb{N}$ is the rank.[1]

The standard optimization recipe for LoRA applies Adam [29] to the two factors $\boldsymbol{A}$ and $\boldsymbol{B}$, treating all the parameters as a single flat vector and ignoring any matrix structure. In language-model pretraining, matrix-aware optimizers such as Muon [24], Scion [40], and Shampoo [16] treat each weight as a matrix and outperform Adam [16, 35, 41, 44, 49]. Muon in particular and variants such as NorMuon [33] have set records for training small language models [23, 27] and have been adopted in recent large-scale training runs [11, 14].

It is natural to ask whether similar gains can extend to LoRA. We found, however, that applying Muon to each factor does not improve over Adam. Part of the problem is structural: the layer output depends on $\boldsymbol{A}$ and $\boldsymbol{B}$ only through the *product* $\boldsymbol{B}\boldsymbol{A}$, while the optimizer acts on the *factors* $\boldsymbol{A}$ and $\boldsymbol{B}$ individually. Several recent works [9, 28, 32, 51] correct this with variants of a product-aware Muon step we refer to as *Product Muon* (Section 3). In our experiments, two such variants, iMuon [32] and LoRA-RITE [51], do not improve over Adam (Figure 1).

To obtain a consistent speedup over Adam, we introduce PoLoRA, an optimizer derived from a steepest descent step whose constraint is motivated by controlling the change in the per-sample loss, and whose solution is an orthogonalized update preconditioned by a curvature estimate. For LoRA, we make this update product-aware by imposing the constraint on the merged update rather than on

[1] LoRA [20, 36] scales the adapter $\boldsymbol{B}\boldsymbol{A}$ by α/r for some $\alpha > 0$. We set $\alpha = r$ [4, 42], so this ratio is 1.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

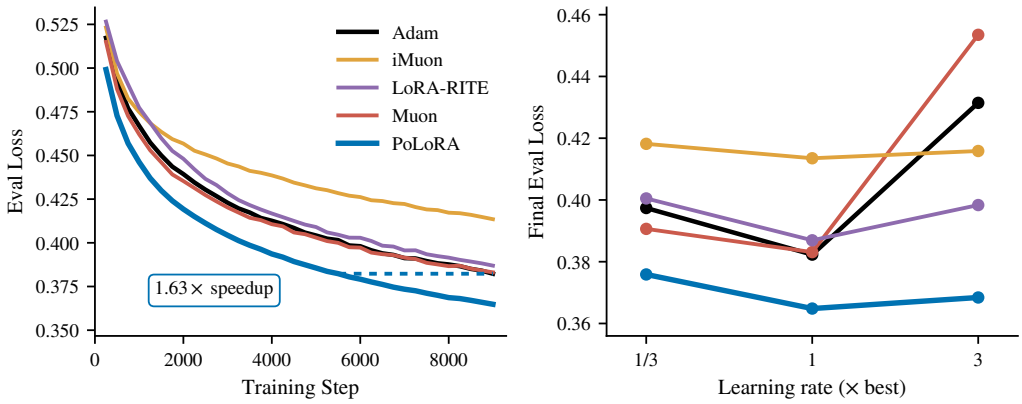

Figure 1: PoLoRA outperforms baselines during training. Llama-3.2-1B finetuned on the math dataset at rank $r = 256$. **Left:** evaluation loss across training steps; PoLoRA outperforms Adam, iMuon [32], Muon, and LoRA-RITE [51] and reaches Adam's final loss with a 1.63× speedup in steps. **Right:** final evaluation loss versus learning rate, in multiples of each optimizer's best learning rate. Every optimum is bracketed, and PoLoRA attains the lowest loss.

each factor. Like the Product Muon step, PoLoRA bounds the spectral norm of the merged update, but it also controls the size of the factor updates.

Across base models, datasets, and adapter ranks, PoLoRA consistently improves training efficiency over Adam (Tables 1 and 2), and it outperforms matrix-aware LoRA baselines on Llama-3.2-1B finetuned on math (Figure 1). It reaches the final held-out loss of tuned Adam with a 1.2–1.7× speedup in training steps, while adding at most 3% per-step overhead. Removing the curvature preconditioning and the magnitude rule (21) in turn shows that each accounts for about half of the speedup over Adam. Finally, we observe that PoLoRA is easier to tune than Adam: its optimal learning rate is stable across ranks, and performance is less sensitive to the learning rate.

The rest of the paper proceeds as follows. In Section 2 we derive a preconditioned Muon step from per-sample loss control. In Section 3 we make this step product-aware for LoRA and control the size of the factor updates, then add momentum and diagonal preconditioner estimates to obtain the PoLoRA optimizer (Algorithm 1). In Section 4 we evaluate PoLoRA against Adam and matrix-aware baselines and ablate its components, and in Section 5 we discuss related work. All proofs are in Appendix B.

2 Per-Sample Loss Control

We first consider a single linear layer $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and defer the LoRA product structure to Section 3. With the preceding layers fixed, let $\mathcal{D}$ denote the distribution of inputs x to this layer and $\ell(\boldsymbol{W}x)$ the loss on input x. The distribution $\mathcal{D}$ is induced by the distribution of the data and by the parameters of the preceding layers, but we omit this dependence for simplicity. The expected loss is $\mathcal{L}(\boldsymbol{W}) = \mathbb{E}_x\left[\ell(\boldsymbol{W}x)\right]$, with gradient $\boldsymbol{G} = \nabla_W \mathcal{L}(\boldsymbol{W}) = \mathbb{E}_x\left[\boldsymbol{G}(x)\right]$, where $\boldsymbol{G}(x) = \nabla_W \ell(\boldsymbol{W}x)$ is the per-sample gradient.

The Muon update is based on a linear minimization oracle (LMO) [3, 7, 8], which minimizes the local linearization of the loss subject to a spectral-norm ball constraint,

$$\begin{array}{ll} \underset{\Delta \boldsymbol{W}}{\text{minimize}} & \langle \boldsymbol{G}, \Delta \boldsymbol{W} \rangle \\ \text{subject to} & \|\Delta \boldsymbol{W}\|_2 \le \eta, \end{array} \qquad \text{where} \qquad \|\boldsymbol{W}\|_2 := \max_{\|x\|_2 = 1} \|\boldsymbol{W}x\|_2. \tag{2}$$

A solution to (2) is given by

$$\Delta \boldsymbol{W} = -\eta \, \operatorname{msign}(\boldsymbol{G}), \tag{3}$$

where if $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ is the reduced SVD, then the matrix sign [3] $\operatorname{msign}(\boldsymbol{G}) = \boldsymbol{U}\boldsymbol{V}^\top$ (see Lemma B.3). In practice, Muon replaces the gradient with an exponential moving average

(EMA) of past gradients, and approximates the matrix sign with a polynomial iteration [1, 52] (see Appendix E.2).

A justification for the spectral-norm constraint in (2) is that it is conservative, since $\|\Delta \boldsymbol{W}\|_2 \le \eta$ is equivalent to bounding the change in the layer's output,

$$\|\Delta \boldsymbol{W} x\|_2 \le \eta, \quad \text{for all } \|x\|_2 \le 1.$$

However, this viewpoint is not entirely satisfying, since it does not directly relate to the loss function ℓ that we are trying to minimize. For training, what we care about is that the loss is stable. We can enforce this by using the following loss sensitivity constraint

$$|\ell((\boldsymbol{W} + \Delta \boldsymbol{W})x) - \ell(\boldsymbol{W}x)| \le \tau, \qquad \text{for all } x \in \operatorname{support}(\mathcal{D}). \tag{4}$$

Since the loss is nonlinear and nonconvex, for tractability we replace it with a local linearization,

$$\ell((\boldsymbol{W} + \Delta \boldsymbol{W})x) - \ell(\boldsymbol{W}x) \approx \langle \boldsymbol{G}(x), \Delta \boldsymbol{W} \rangle.$$

Replacing the spectral constraint in (2) with the linearized loss sensitivity constraint gives

$$\underset{\Delta \boldsymbol{W}}{\text{minimize}} \quad \langle \boldsymbol{G}, \Delta \boldsymbol{W} \rangle \quad \text{subject to} \quad |\langle \boldsymbol{G}(x), \Delta \boldsymbol{W} \rangle| \le \tau, \qquad x \in \operatorname{support}(\mathcal{D}). \tag{5}$$

This LMO (5) is well-posed (Proposition B.2), but enforcing its constraints requires every per-sample gradient $\boldsymbol{G}(x)$. Instead we relax the constraint from holding for every $x \in \operatorname{support}(\mathcal{D})$ to holding with high probability, by defining an *outergradient* set $\mathcal{G}_\delta$ such that

$$\Pr_{x \sim \mathcal{D}}(\boldsymbol{G}(x) \in \mathcal{G}_\delta) \ge 1 - \delta. \tag{6}$$

Imposing $|\langle \widetilde{\boldsymbol{G}}, \Delta \boldsymbol{W} \rangle| \le \tau$ for every $\widetilde{\boldsymbol{G}} \in \mathcal{G}_\delta$ then ensures, by (6), that the per-sample constraint $|\langle \boldsymbol{G}(x), \Delta \boldsymbol{W} \rangle| \le \tau$ holds with probability at least $1 - \delta$ over $x \sim \mathcal{D}$.

To guide our design of the outergradient set, we first observe that per-sample gradients of linear layers are rank one.

Lemma 2.1 (Gradients are rank one)**.** For every $x \in \operatorname{support}(\mathcal{D})$ the per-sample gradient factors as $\boldsymbol{G}(x) = b(x)\, x^\top$, where $b(x) = \frac{d}{dz}\ell(z)\big|_{z = \boldsymbol{W}x} \in \mathbb{R}^{d_{\text{out}}}$ is the output-side gradient.

Secondly, we note that when measured in an appropriate norm, the size of per-sample gradients is bounded with high probability. Define $\boldsymbol{\Sigma}$ to be the second moment of the per-sample gradients and $\|\cdot\|_{\boldsymbol{\Sigma}}$ to be the associated Mahalanobis norm

$$\boldsymbol{\Sigma} := \mathbb{E}_x\left[\operatorname{vec}(\boldsymbol{G}(x))\operatorname{vec}(\boldsymbol{G}(x))^\top\right], \qquad \|\boldsymbol{M}\|_{\boldsymbol{\Sigma}}^2 := \operatorname{vec}(\boldsymbol{M})^\top \boldsymbol{\Sigma}^\dagger \operatorname{vec}(\boldsymbol{M}). \tag{7}$$

Lemma B.1 gives $\mathbb{E}_x\left[\|\boldsymbol{G}(x)\|_{\boldsymbol{\Sigma}}^2\right] = \operatorname{rank}(\boldsymbol{\Sigma})$, so by Markov's inequality $\|\boldsymbol{G}(x)\|_{\boldsymbol{\Sigma}}^2 \le \operatorname{rank}(\boldsymbol{\Sigma})/\delta$ with probability at least $1 - \delta$. Combined with Lemma 2.1, this implies that the set

$$\mathcal{G}_\delta(\boldsymbol{\Sigma}) := \left\{ \widetilde{\boldsymbol{G}} = bx^\top \;\middle|\; b \in \mathbb{R}^{d_{\text{out}}},\ x \in \mathbb{R}^{d_{\text{in}}}, \|\widetilde{\boldsymbol{G}}\|_{\boldsymbol{\Sigma}}^2 \le \operatorname{rank}(\boldsymbol{\Sigma})/\delta \right\} \tag{8}$$

satisfies (6).

Kronecker approximation. Since $\boldsymbol{\Sigma}$ contains $(d_{\text{out}} d_{\text{in}})^2$ entries, storing and inverting it is infeasible. Following a long line of work [16, 34, 37, 48], we approximate $\boldsymbol{\Sigma}$ by a Kronecker product of symmetric positive definite matrices $\boldsymbol{P} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{out}}}$ and $\boldsymbol{Q} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$,

$$\boldsymbol{\Sigma} \approx \boldsymbol{Q} \otimes \boldsymbol{P}. \tag{9}$$

We treat the *preconditioners* $\boldsymbol{P}$ and $\boldsymbol{Q}$ as given here and fit them online from batch gradients in Appendix D.

Imposing the constraint in (5) for every $\widetilde{\boldsymbol{G}} \in \mathcal{G}_\delta(\boldsymbol{Q} \otimes \boldsymbol{P})$ gives the following preconditioned version of Muon (Lemma 2.2).

Lemma 2.2 (Preconditioned Muon)**.** The LMO

$$\min_{\Delta \boldsymbol{W}} \langle \boldsymbol{G}, \Delta \boldsymbol{W} \rangle \quad \text{subject to} \quad |\langle \widetilde{\boldsymbol{G}}, \Delta \boldsymbol{W} \rangle| \leq \tau \qquad \text{for every} \quad \widetilde{\boldsymbol{G}} \in \mathcal{G}_\delta(\boldsymbol{Q} \otimes \boldsymbol{P}), \tag{10}$$

is equivalent, up to a constant rescaling of τ, to

$$\min_{\Delta \boldsymbol{W}} \langle \boldsymbol{G}, \Delta \boldsymbol{W} \rangle \quad \text{subject to} \quad \left\| \boldsymbol{P}^{1/2} \Delta \boldsymbol{W} \boldsymbol{Q}^{1/2} \right\|_2 \leq \tau, \tag{11}$$

which has the closed-form solution

$$\Delta \boldsymbol{W} = -\tau \, \boldsymbol{P}^{-1/2} \operatorname{msign}(\boldsymbol{P}^{-1/2} \boldsymbol{G} \boldsymbol{Q}^{-1/2}) \boldsymbol{Q}^{-1/2}. \tag{12}$$

With $\boldsymbol{P} = \boldsymbol{I}_{d_{\text{out}}}$ and $\boldsymbol{Q} = \boldsymbol{I}_{d_{\text{in}}}$, the update (12) becomes the Muon update (3).

3 PoLoRA: Preconditioned Orthogonalized LoRA

So far we have described an update for the weight matrix $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. We now adapt it to the product structure of LoRA (1). We first derive the *direction* of the factor updates, followed by their magnitude. Updating both factors gives

$$\Delta \boldsymbol{W} = (\boldsymbol{B} + \Delta \boldsymbol{B})(\boldsymbol{A} + \Delta \boldsymbol{A}) - \boldsymbol{B}\boldsymbol{A} \approx \boldsymbol{B} \Delta \boldsymbol{A} + \Delta \boldsymbol{B} \boldsymbol{A}. \tag{13}$$

Recall that $\boldsymbol{G} := \nabla_W \mathcal{L}(\boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A})$. The corresponding first-order change in the loss is $\langle \boldsymbol{G}_A, \Delta \boldsymbol{A} \rangle + \langle \boldsymbol{G}_B, \Delta \boldsymbol{B} \rangle$, where the chain rule gives the factor gradients

$$\boldsymbol{G}_A := \nabla_A \mathcal{L}(\boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A}) = \boldsymbol{B}^\top \boldsymbol{G}, \qquad \boldsymbol{G}_B := \nabla_B \mathcal{L}(\boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A}) = \boldsymbol{G}\boldsymbol{A}^\top. \tag{14}$$

Direction. Replacing $\Delta \boldsymbol{W}$ in (11) with its linearization (13) gives the LMO

$$\begin{aligned} &\underset{\Delta \boldsymbol{A}, \Delta \boldsymbol{B}}{\text{minimize}} \quad \langle \boldsymbol{G}, \boldsymbol{B} \Delta \boldsymbol{A} + \Delta \boldsymbol{B} \boldsymbol{A} \rangle \\ &\text{subject to} \quad \|\boldsymbol{P}^{1/2} (\boldsymbol{B} \Delta \boldsymbol{A} + \Delta \boldsymbol{B} \boldsymbol{A}) \boldsymbol{Q}^{1/2}\|_2 \leq \tau. \end{aligned} \tag{15}$$

The constraint in (15) couples $\Delta \boldsymbol{A}$ and $\Delta \boldsymbol{B}$. To decouple them, we split it using the triangle inequality,

$$\|\boldsymbol{P}^{1/2} (\boldsymbol{B} \Delta \boldsymbol{A} + \Delta \boldsymbol{B} \boldsymbol{A}) \boldsymbol{Q}^{1/2}\|_2 \leq \|\boldsymbol{P}^{1/2} \boldsymbol{B} \Delta \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2 + \|\boldsymbol{P}^{1/2} \Delta \boldsymbol{B} \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2,$$

so that bounding each term on the right by $\tau/2$ satisfies the constraint. The objective also splits,

$$\langle \boldsymbol{G}, \boldsymbol{B} \Delta \boldsymbol{A} + \Delta \boldsymbol{B} \boldsymbol{A} \rangle = \langle \boldsymbol{G}_A, \Delta \boldsymbol{A} \rangle + \langle \boldsymbol{G}_B, \Delta \boldsymbol{B} \rangle,$$

so the problem separates into two factorwise LMOs

$$\begin{aligned} &\underset{\Delta \boldsymbol{A}}{\text{minimize}} \ \langle \boldsymbol{G}_A, \Delta \boldsymbol{A} \rangle \\ &\text{subject to} \ \|\boldsymbol{P}^{1/2} \boldsymbol{B} \Delta \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2 \leq \tau, \end{aligned} \qquad \begin{aligned} &\underset{\Delta \boldsymbol{B}}{\text{minimize}} \ \langle \boldsymbol{G}_B, \Delta \boldsymbol{B} \rangle \\ &\text{subject to} \ \|\boldsymbol{P}^{1/2} \Delta \boldsymbol{B} \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2 \leq \tau, \end{aligned} \tag{16}$$

where we absorb the factor $1/2$ into τ, since the magnitudes are set separately below.

The LMOs in (16) admit a closed-form solution. Define the matrices

$$\boldsymbol{C}_B = \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{B} \in \mathbb{R}^{r \times r}, \qquad \boldsymbol{C}_A = \boldsymbol{A} \boldsymbol{Q} \boldsymbol{A}^\top \in \mathbb{R}^{r \times r}. \tag{17}$$

Using these $r \times r$ matrices, we rewrite the norms in the two constraints of (16) as

$$\|\boldsymbol{P}^{1/2} \boldsymbol{B} \Delta \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2 = \|\boldsymbol{C}_B^{1/2} \Delta \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2, \qquad \|\boldsymbol{P}^{1/2} \Delta \boldsymbol{B} \boldsymbol{A} \boldsymbol{Q}^{1/2}\|_2 = \|\boldsymbol{P}^{1/2} \Delta \boldsymbol{B} \boldsymbol{C}_A^{1/2}\|_2. \tag{18}$$

By (18) and Lemma 2.2, $\Delta \boldsymbol{A} = -\tau \boldsymbol{D}_A$ and $\Delta \boldsymbol{B} = -\tau \boldsymbol{D}_B$ solve the LMOs (16), where

$$\begin{aligned} \boldsymbol{D}_A &= \boldsymbol{C}_B^{-1/2} \operatorname{msign}\left(\boldsymbol{C}_B^{-1/2} \boldsymbol{G}_A \boldsymbol{Q}^{-1/2}\right) \boldsymbol{Q}^{-1/2}, \\ \boldsymbol{D}_B &= \boldsymbol{P}^{-1/2} \operatorname{msign}\left(\boldsymbol{P}^{-1/2} \boldsymbol{G}_B \boldsymbol{C}_A^{-1/2}\right) \boldsymbol{C}_A^{-1/2}. \end{aligned} \tag{19}$$

Setting $\boldsymbol{P} = \boldsymbol{I}_{d_{\text{out}}}$ and $\boldsymbol{Q} = \boldsymbol{I}_{d_{\text{in}}}$ in (19) gives what we call the *Product Muon* step. It is at the core of the updates in several recent works [9, 28, 32, 51] which we discuss in Appendix C.

Magnitude. Having derived the update directions, we now set the factor updates to have the same spectral norm, $\|\Delta \boldsymbol{A}\|_2 = \|\Delta \boldsymbol{B}\|_2 = \rho$. To choose ρ, we ensure that the linearized merged update (13) has spectral norm bounded by the learning rate η. Using the triangle inequality and submultiplicativity, we have

$$\|\Delta \boldsymbol{W}\|_2 \approx \|\boldsymbol{B}\Delta \boldsymbol{A} + \Delta \boldsymbol{B}\boldsymbol{A}\|_2 \leq \|\boldsymbol{B}\|_2\|\Delta \boldsymbol{A}\|_2 + \|\boldsymbol{A}\|_2\|\Delta \boldsymbol{B}\|_2. \tag{20}$$

Setting the right-hand side of (20) to η, and substituting in $\|\Delta \boldsymbol{A}\|_2 = \|\Delta \boldsymbol{B}\|_2 = \rho$ gives

$$\rho = \frac{\eta}{\|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2} \tag{21}$$

which ensures $\|\Delta \boldsymbol{W}\|_2 \leq \eta$. Consequently, given the directions $\boldsymbol{D}_A$ and $\boldsymbol{D}_B$, we define the factor updates to be

$$\Delta \boldsymbol{A} = -\rho \frac{\boldsymbol{D}_A}{\|\boldsymbol{D}_A\|_2}, \qquad \Delta \boldsymbol{B} = -\rho \frac{\boldsymbol{D}_B}{\|\boldsymbol{D}_B\|_2}. \tag{22}$$

Optimizer step. To make the update (22) practical for minibatch training, we integrate the following ingredients, resulting in the PoLoRA step summarized in Algorithm 1:

- *Diagonal curvature.* To reduce overhead, we use diagonal preconditioners $\boldsymbol{P} = \mathrm{diag}(p)$ and $\boldsymbol{Q} = \mathrm{diag}(q)$, as in Adafactor [45] (line 2), and fit them to the second moment of the weight gradient (9) using the factor gradients, by adapting the KL-Shampoo update [34] to the diagonal case (lines 7 and 8; Appendix D).
- *Averaging.* We replace each factor gradient with a look-ahead momentum estimate (line 1), and average p and q with an EMA (lines 7 and 8; Appendix D).
- *Normalization.* The Kronecker product in (9) is invariant to the rescaling $(\boldsymbol{P}, \boldsymbol{Q}) \mapsto (a\boldsymbol{P}, a^{-1}\boldsymbol{Q})$. To make the EMA updates for $\boldsymbol{P}$ and $\boldsymbol{Q}$ invariant to this rescaling as well, we normalize each preconditioner so that its largest entry is one (line 2; Appendix D).
- *Fast numerical subroutines.* We estimate the spectral norms in lines 5 and 6 by power iteration (Appendix E.1). We use Gram Newton–Schulz iterations [52] with PolarExpress coefficients [1] to compute the matrix sign and inverse square roots in lines 3 and 4 (Appendix E.2).

4 Experiments

We now empirically evaluate PoLoRA, comparing it to several baselines in Section 4.2, investigating which design choices of PoLoRA are responsible for its efficacy in Section 4.3, and testing its behavior across ranks and on a low-resource language in Section 4.4.

4.1 Setup

Comparison. For every model, dataset, and rank, we only vary the optimizer and learning rate. All methods use LoRA on every linear layer except the output layer, with the standard initialization $\boldsymbol{B} = 0$ and random $\boldsymbol{A}$ [20].[2] For each optimizer, we sweep the learning rate under a constant schedule [12, 43] with no weight decay and select the run with the lowest held-out loss.

The main comparisons finetune OLMo-2-1B [38], Llama-3.2-1B [15], Qwen2.5-1.5B [50], and Llama-3-8B [15] on code (OpenCoder [21]) and math (OpenMathInstruct-2 [47]) datasets, using rank $r = 256$ unless stated otherwise. Appendix A gives the full experimental configurations.

Baselines. We compare PoLoRA (Section 3) to the following optimizers:

- **Adam** [29] — our baseline, applied independently to the two factors $\boldsymbol{A}$ and $\boldsymbol{B}$.
- **Muon** [24] — the solution of the spectral LMO (2), applied independently to each factor.
- **iMuon** [32] — a product-aware spectral LMO (26) applied to momentum in the product space.

[2] The ablation variant without magnitude control diverges at $\boldsymbol{B} = 0$ and hence uses a nonzero initialization (Appendix E).

Algorithm 1 PoLoRA step for one LoRA pair $(\boldsymbol{A}, \boldsymbol{B})$

Hyperparameters: learning rate η, momentum decay β_1, curvature decay β_2, numerical-stability constant ε
Initialize: $\boldsymbol{M}_A, \boldsymbol{M}_B \leftarrow 0$; $\quad p, q \leftarrow \varepsilon \mathbf{1}$

Update momentum

$\boldsymbol{M}_A \leftarrow \beta_1 \boldsymbol{M}_A + (1-\beta_1)\boldsymbol{G}_A, \qquad \boldsymbol{M}_B \leftarrow \beta_1 \boldsymbol{M}_B + (1-\beta_1)\boldsymbol{G}_B$ ▷ buffer + look-ahead

$\widehat{\boldsymbol{M}}_A \leftarrow \beta_1 \boldsymbol{M}_A + (1-\beta_1)\boldsymbol{G}_A, \qquad \widehat{\boldsymbol{M}}_B \leftarrow \beta_1 \boldsymbol{M}_B + (1-\beta_1)\boldsymbol{G}_B$

Compute directions

$\boldsymbol{P} \leftarrow \operatorname{diag}\big(p/\|p\|_\infty\big), \qquad \boldsymbol{Q} \leftarrow \operatorname{diag}\big(q/\|q\|_\infty\big)$

$\boldsymbol{C}_B \leftarrow \boldsymbol{B}^\top \boldsymbol{P}\, \boldsymbol{B}, \qquad \boldsymbol{C}_A \leftarrow \boldsymbol{A}\, \boldsymbol{Q}\, \boldsymbol{A}^\top$

$\boldsymbol{D}_A \leftarrow \boldsymbol{C}_B^{-1/2}\ \operatorname{msign}\big(\boldsymbol{C}_B^{-1/2}\, \widehat{\boldsymbol{M}}_A\, \boldsymbol{Q}^{-1/2}\big)\, \boldsymbol{Q}^{-1/2}$ ▷ preconditioned matrix sign (19)

$\boldsymbol{D}_B \leftarrow \boldsymbol{P}^{-1/2}\ \operatorname{msign}\big(\boldsymbol{P}^{-1/2}\, \widehat{\boldsymbol{M}}_B\, \boldsymbol{C}_A^{-1/2}\big)\, \boldsymbol{C}_A^{-1/2}$

Balance magnitude

$\rho \leftarrow \eta/(\|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2)$ ▷ spectral-norm control (21)

$\boldsymbol{A} \leftarrow \boldsymbol{A} - \rho\, \boldsymbol{D}_A / \max(\|\boldsymbol{D}_A\|_2, \varepsilon), \qquad \boldsymbol{B} \leftarrow \boldsymbol{B} - \rho\, \boldsymbol{D}_B / \max(\|\boldsymbol{D}_B\|_2, \varepsilon)$

Update preconditioners

$q \leftarrow \beta_2\, q + (1-\beta_2)\operatorname{diag}\big(\boldsymbol{G}_A^\top \boldsymbol{C}_B^{-1} \boldsymbol{G}_A\big)/r$ ▷ coupled estimator (34)

$p \leftarrow \beta_2\, p + (1-\beta_2)\operatorname{diag}\big(\boldsymbol{G}_B\, \boldsymbol{C}_A^{-1} \boldsymbol{G}_B^\top\big)/r$

- **LoRA-RITE** [51] — an adaptive LoRA optimizer that preconditions factor gradients in a transformation-invariant basis.

We use the authors' official implementations for iMuon and LoRA-RITE and provide additional background on these methods in Appendix C.

4.2 Computational Efficiency

We score each optimizer by its *steps-to-Adam*, the number of steps its tuned run needs to match the final loss of tuned Adam. The *step speedup* is then the ratio of Adam's total step count to the steps-to-Adam. Across our experiments, the step speedup of PoLoRA is 1.2–1.7× (Table 1). In the setting of Figure 1, PoLoRA also outperforms Muon, iMuon, and LoRA-RITE, none of which improves over Adam. Additional learning curves can be found in Appendix G.

Wall-clock. The optimizer FLOPs of each PoLoRA step are independent of the batch size, while the model forward and backward passes scale with batch size (Appendix F). Thus the relative per-step time cost of PoLoRA shrinks as the batch grows. At our batch size (Appendix A) it adds at most 3% wall-clock overhead compared to Adam on all four base models (Table 1), and the step speedup carries to wall-clock almost unchanged.

4.3 Component Ablation

The essential ingredients in PoLoRA can be decomposed into three components:

- **Product-awareness**. The direction LMO (16) uses the structure of the merged product $\boldsymbol{BA}$ as opposed to treating $\boldsymbol{A}$ and $\boldsymbol{B}$ as independent matrices.
- **Curvature**. The update accounts for how different directions change the loss by preconditioning the gradient with an adaptive curvature estimate (9).
- **Magnitude**. The update controls the sizes of both the factor steps and the merged update (21).

	Code		Math	
base model	step	wall-clock	step	wall-clock
OLMo-2-1B	1.61×	1.57×	1.74×	1.70×
Llama-3.2-1B	1.55×	1.51×	1.63×	1.58×
Qwen2.5-1.5B	1.30×	1.27×	1.64×	1.60×
Llama-3-8B	1.20×	1.18×	1.59×	1.56×

Table 1: PoLoRA speedup over Adam across models and datasets. Four base models at rank $r = 256$, finetuned on code and math datasets. The wall-clock column discounts the step speedup (Section 4.2) by the per-step overhead (Section 4.2). PoLoRA improves over Adam in every setting, by a wider margin on math than on code.

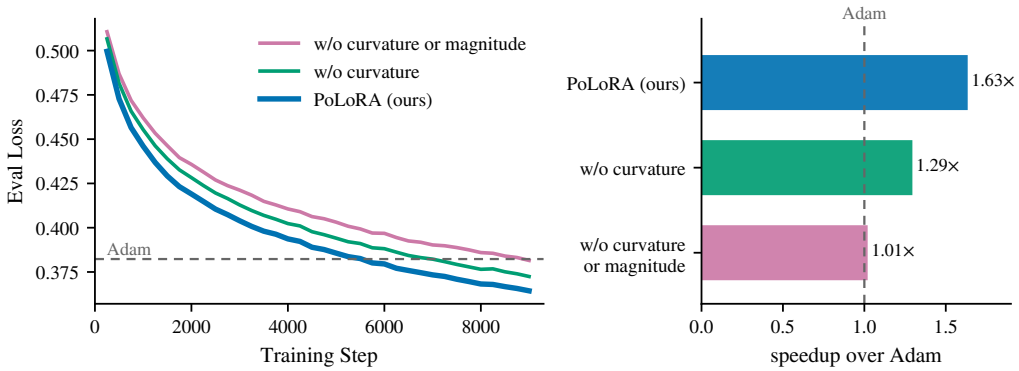

Figure 2: Curvature and magnitude both improve PoLoRA. Component ablation on Llama-3.2-1B finetuned on math at rank $r = 256$. The left panel shows loss curves as the curvature and then the magnitude control are removed from PoLoRA in turn. The right panel shows the speedup over Adam for each variant.

In Figure 2, removing both the curvature preconditioning ($\boldsymbol{P}$ and $\boldsymbol{Q}$ set to identity) and the magnitude control ($\tau = \eta/2$ in (16)) eliminates the speedup over Adam. Removing them sequentially shows that each accounts for about half of the speedup.

4.4 Rank and Distribution Shift

We next vary the LoRA rank and how well the pretraining data represents the finetuning dataset.

Rank. We sweep the rank r on Llama-3.2-1B finetuned on math. As the rank grows, the step speedup of PoLoRA over Adam grows too (Table 2; learning curves in Figure 5(c)). PoLoRA also exhibits learning-rate transfer: its optimal learning rate is stable across ranks, whereas Adam's shifts (Figure 3). We also observe reduced sensitivity to the learning rate compared to Adam (Figure 3).

Distribution shift. Coding data is well represented in the pretraining data of the Qwen2.5-1.5B model [50], while data for a low-resource language like Bengali is much less represented. From Figure 4 we observe that the speedup over Adam is larger on the Bengali dataset (Aya [46]) than on code. This suggests that the advantage of PoLoRA over Adam grows with how much the model must adapt to the finetuning data.

5 Related Work

Here we review related work and contrast PoLoRA with the existing literature.

Hyperparameter rescaling for LoRA. Several recent methods refine LoRA training with Adam [20, 29] by rescaling hyperparameters such as the learning rates, adapter scale, or initialization variance to account for the low-rank structure. Motivated by large-width feature-learning

rank r	step speedup	wall-clock speedup
32	1.48×	1.44×
64	1.54×	1.50×
128	1.52×	1.49×
256	1.63×	1.58×

Table 2: PoLoRA speedup over Adam across LoRA ranks. Llama-3.2-1B finetuned on the math dataset, with only the rank varied across rows. The wall-clock column discounts the step speedup (Section 4.2) by the per-step overhead (Section 4.2).

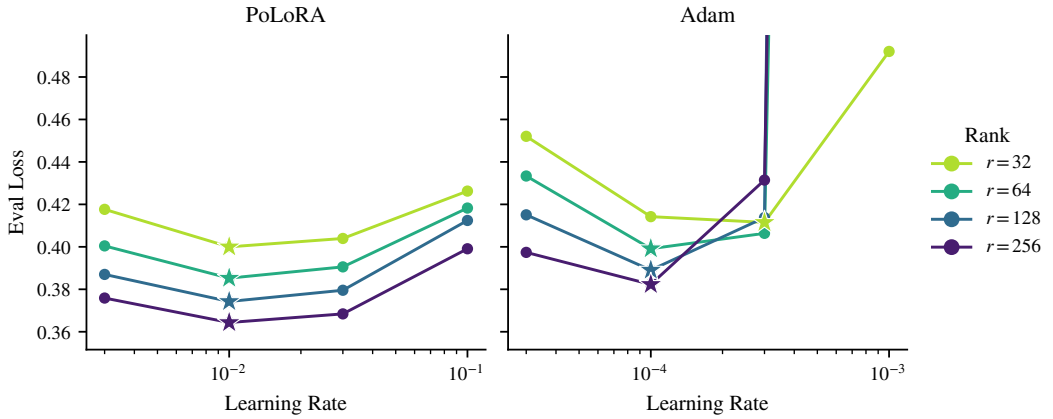

Figure 3: PoLoRA exhibits learning-rate transfer. Final loss versus learning rate of Llama-3.2-1B finetuned on the math dataset at each rank $r \in \{32, 64, 128, 256\}$. The optimal learning rate for PoLoRA stays at 10^{-2} across ranks, while Adam's drifts.

arguments, LoRA+ [17] assigns $\boldsymbol{B}$ a larger learning rate than $\boldsymbol{A}$, and follow-up work [18, 31] extends this analysis to other initializations. Rank-stabilized LoRA [25] replaces the α/r adapter scale with $\alpha/\sqrt{r}$ to avoid the rank-dependent shrinkage that makes high-rank adapters hard to train, and μA [10] studies how these hyperparameters should scale with rank to keep the optimal learning rate stable. In contrast, PoLoRA replaces the Adam update itself rather than rescaling these hyperparameters.

Matrix-aware updates. Recently, optimizers that operate at the matrix level, rather than treating parameters as a flattened vector, have attracted attention [35, 41, 49]. Muon [24] solves the spectral-norm LMO for a weight matrix, producing an update equal to the matrix sign of the momentum gradient [1] (Lemma B.3). Shampoo [16], SOAP [48], and KL-Shampoo [34] estimate the gradient curvature as Kronecker-factored second moments. Our preconditioners use a diagonal version of the KL-Shampoo estimator, adapted to LoRA (Appendix D). Mousse [53] uses such a second-moment estimate to whiten the gradient before a matrix-sign step. PoLoRA likewise applies a matrix-sign step to a preconditioned gradient, with a norm derived from controlling the per-sample loss change (Section 2). We carry this update from the weight matrix to the LoRA product and add magnitude control.

Product-invariant LoRA optimizers. The learned update $\boldsymbol{BA}$ is a product of two matrices, so its factors are determined only up to the change of basis $(\boldsymbol{A}, \boldsymbol{B}) \mapsto (\boldsymbol{NA}, \boldsymbol{BN}^{-1})$ for invertible $\boldsymbol{N}$. Since Adam is not invariant to this transformation, a line of work has proposed LoRA optimizers that are. LoRA-RITE removes the basis dependence with an adaptive preconditioner [51], while Riemannion optimizes $\boldsymbol{BA}$ directly on the fixed-rank matrix manifold [5]. iMuon, LoRA-Muon, and Compositional Muon derive invariant updates from spectral-norm LMOs [9, 28, 32]. As discussed in Appendix C, the momentum-free LMO updates of LoRA-RITE, LoRA-Muon, and Compositional Muon's half-split rule all reduce to the same Product Muon step, as does iMuon's intrinsic LMO. PoLoRA does not satisfy this invariance yet outperforms iMuon and LoRA-RITE (Figure 1), which suggests that invariance is not the limiting factor in this setting.

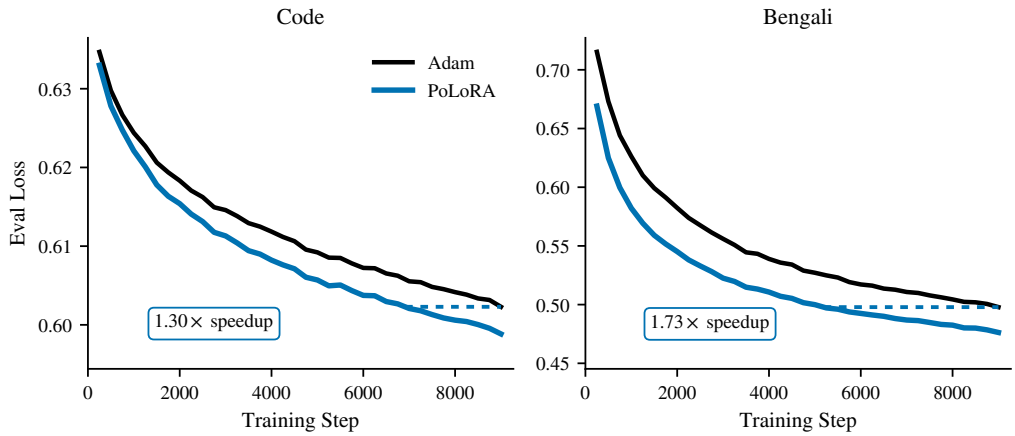

Figure 4: The speedup over Adam is larger on the low-resource language. Evaluation loss over training steps on Qwen2.5-1.5B at rank $r = 256$, finetuned on the code dataset (left) and the Bengali dataset (right).

Magnitude control for low-rank products. iMuon, LoRA-Muon, and Compositional Muon bound only $\|\boldsymbol{B}\Delta\boldsymbol{A}\|_2$ and $\|\Delta\boldsymbol{B}\boldsymbol{A}\|_2$ but not the factor steps $\|\Delta\boldsymbol{A}\|_2$ and $\|\Delta\boldsymbol{B}\|_2$. As a result, when $\boldsymbol{A}$ or $\boldsymbol{B}$ has small singular values, the updates $\Delta\boldsymbol{A}$ and $\Delta\boldsymbol{B}$ can become arbitrarily large. Spectron [22] and QuacK [2] explicitly control the step size of matrix product factors by scaling it with the factor norms. Spectron rescales both factor updates to have spectral norm $\eta/(\|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2 + 1)$, and QuacK scales each factor's learning rate by the inverse norm of the other. Similar to Spectron, PoLoRA rescales both factor steps to have a common spectral norm $\rho = \eta/(\|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2)$.

6 Conclusion

We derived PoLoRA from a steepest descent step that controls the change in the per-sample loss. This derivation leads to a practical optimizer that combines a product-aware spectral direction, lightweight curvature preconditioning, and explicit control of factor-step magnitudes. In our experiments, PoLoRA is consistently faster than Adam, and its optimal learning rate transfers across ranks.

The method is still built on approximations. For efficiency the preconditioners are diagonal, and the metric they induce is a Kronecker-factored approximation of the per-sample second moment. The derivation also relies on a linearized loss change and a linearized product update, so the optimizer does not control higher-order effects or the dropped $\Delta\boldsymbol{B}\Delta\boldsymbol{A}$ term. Finally, our evaluation focuses on LoRA finetuning for language models and on held-out loss as the main measure of progress; broader downstream evaluations, longer training horizons, other adapter families, and different hardware or batch-size regimes remain open.

These limitations point to several future directions. One is to design better outergradient approximations that remain cheap enough for adapter training. Another is to incorporate second-order information [13] while keeping the update tractable. More broadly, the per-sample loss-control perspective (5) and (10) may be useful beyond LoRA, for pretraining or other compositions of maps.

Reproducibility Statement

Appendix A lists the full configuration of every setting, including the learning-rate grids and the seed-to-seed standard deviation of the final loss, and Appendix B contains the proofs. Appendix E describes how Algorithm 1 is implemented, and the supplementary material contains the code for PoLoRA. All base models and datasets are publicly available (Section 4.1).

AI Use Statement

We used generative AI tools (Claude Code and Codex) to implement methods, including the optimizer, the baselines, and the training, evaluation, and data-formatting pipeline; to propose and refine hypotheses and mathematical claims; to assist in developing the theoretical framework; to give feedback on the design of experiments; and to interpret results. We also used them to create figures, draft and edit parts of the paper, search and summarize the literature, and format references. We did not use them to write proofs or to provide the ingredients of proofs, which the authors derived by hand. Synthetic data generation, translation, and qualitative data analysis do not apply to this work. The authors reviewed all AI-assisted work. Implementations were checked with unit tests and with equivalence tests against the official implementations of iMuon and LoRA-RITE, and descriptions of cited methods were checked against the original papers. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

[1] N. Amsel et al. *The Polar Express: Optimal Matrix Sign Methods and Their Application to the Muon Algorithm*. 2025. arXiv: 2505.16932 [cs.LG].

[2] B. Anson and L. Aitchison. *Controlling changes to attention logits*. 2025. arXiv: 2511.21377 [cs.LG].

[3] J. Bernstein and L. Newhouse. "Modular Duality in Deep Learning". In: *Proceedings of the 42nd International Conference on Machine Learning*. Vol. 267. Proceedings of Machine Learning Research. PMLR, 2025, pp. 3920–3930.

[4] D. Biderman et al. "LoRA Learns Less and Forgets Less". In: *Transactions on Machine Learning Research (TMLR)* (2024). arXiv: 2405.09673 [cs.LG].

[5] V. Bogachev et al. *LoRA meets Riemannion: Muon Optimizer for Parametrization-independent Low-Rank Adapters*. 2025. arXiv: 2507.12142 [cs.LG].

[6] R. Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2021. arXiv: 2108.07258 [cs.LG].

[7] D. Carlson, V. Cevher, and L. Carin. "Stochastic Spectral Descent for Restricted Boltzmann Machines". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, May 2015, pp. 111–119.

[8] D. E. Carlson et al. "Preconditioned Spectral Descent for Deep Learning". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015, pp. 2971–2979.

[9] F. L. Cesista et al. *LoRA-Muon: Spectral Steepest Descent on the Low-Rank Manifold*. 2026. arXiv: 2606.12921 [cs.LG].

[10] N. Chen, S. Villar, and S. Hayou. *Learning Rate Scaling across LoRA Ranks and Transfer to Full Finetuning*. 2026. arXiv: 2602.06204 [cs.LG].

[11] DeepSeek-AI. *DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence*. 2026. arXiv: 2606.19348 [cs.CL].

[12] T. Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG].

[13] Z. Du and W. Su. *The Newton-Muon Optimizer*. 2026. arXiv: 2604.01472 [math.OC].

[14] GLM-4.5 Team. *GLM-4.5: Agentic, Reasoning, and Coding (ARC) Foundation Models*. 2025. arXiv: 2508.06471 [cs.CL].

[15] A. Grattafiori et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI].

[16] V. Gupta, T. Koren, and Y. Singer. “Shampoo: Preconditioned Stochastic Tensor Optimization”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1842–1850. arXiv: 1802.09568 [cs.LG].

[17] S. Hayou, N. Ghosh, and B. Yu. “LoRA+: Efficient Low Rank Adaptation of Large Models”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. PMLR, 2024, pp. 17783–17806. arXiv: 2402.12354 [cs.LG].

[18] S. Hayou, N. Ghosh, and B. Yu. “The Impact of Initialization on LoRA Finetuning Dynamics”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson et al. Vol. 37. Curran Associates, Inc., 2024, pp. 117015–117040.

[19] S. Hayou, N. Ghosh, and B. Yu. *PLoP: Precise LoRA Placement for Efficient Finetuning of Large Models*. 2025. arXiv: 2506.20629 [cs.LG].

[20] E. J. Hu et al. “LoRA: Low-Rank Adaptation of Large Language Models”. In: *International Conference on Learning Representations (ICLR)*. 2022. arXiv: 2106.09685 [cs.CL].

[21] S. Huang, T. Cheng, et al. *OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models*. 2024. arXiv: 2411.04905 [cs.CL].

[22] P. Janson, E. Oyallon, and E. Belilovsky. *Stabilizing Native Low-Rank LLM Pretraining*. International Conference on Machine Learning (ICML). 2026. arXiv: 2602.12429 [cs.LG].

[23] K. Jordan et al. *modded-nanogpt: Speedrunning the NanoGPT Baseline*. 2024.

[24] K. Jordan et al. *Muon: An optimizer for hidden layers in neural networks*. https://kellerjordan.github.io/posts/muon/. 2024.

[25] D. Kalajdzievski. *A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA*. 2023. arXiv: 2312.03732 [cs.LG].

[26] J. Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG].

[27] A. Karpathy. *nanochat: The Best ChatGPT that $100 Can Buy*. 2025.

[28] B. Keigwin et al. *Towards Compositional Steepest Descent*. https://blog.tilderesearch.com/blog/compositional-muon. Tilde Research. 2026.

[29] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. 2015. arXiv: 1412.6980 [cs.LG].

[30] B. Kulis, M. A. Sustik, and I. S. Dhillon. “Low-Rank Kernel Learning with Bregman Matrix Divergences”. In: *Journal of Machine Learning Research* 10 (2009), pp. 341–376.

[31] S. Li et al. “Beyond zero initialization: Investigating the impact of non-zero initialization on lora fine-tuning dynamics”. In: *International Conference on Machine Learning (ICML)* (2025).

[32] Y. Li et al. *Intrinsic Muon: Spectral Optimization on Riemannian Matrix Manifolds*. 2026. arXiv: 2605.09238 [cs.LG].

[33] Z. Li et al. *NorMuon: Making Muon More Efficient and Scalable*. 2025. arXiv: 2510.05491 [cs.LG].

[34] W. Lin et al. “Understanding and Improving Shampoo and SOAP via Kullback–Leibler Minimization”. In: *International Conference on Learning Representations (ICLR)*. 2026. arXiv: 2509.03378 [stat.ML].

[35] J. Liu et al. “Muon is scalable for llm training”. In: *arXiv preprint arXiv:2502.16982* (2025).

[36] S. Mangrulkar et al. *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*. https://github.com/huggingface/peft. 2022.

[37] J. Martens and R. Grosse. *Optimizing Neural Networks with Kronecker-factored Approximate Curvature*. 2015. arXiv: 1503.05671 [cs.LG].

[38] OLMo Team. *2 OLMo 2 Furious*. Conference on Language Modeling (COLM). 2025. arXiv: 2501.00656 [cs.CL].

[39] T. Parshakova et al. “Muon Does Not Converge on Convex Lipschitz Functions”. In: *arXiv preprint arXiv:2605.08980* (2026).

[40] T. Pethick et al. “Training Deep Learning Models with Norm-Constrained LMOs”. In: *Proceedings of the 42nd International Conference on Machine Learning*. Ed. by A. Singh et al. Vol. 267. Proceedings of Machine Learning Research. PMLR, July 2025, pp. 49069–49104.

[41] S. Qiu et al. *Hyperparameter Transfer Enables Consistent Gains of Matrix-Preconditioned Optimizers Across Scales*. Advances in Neural Information Processing Systems (NeurIPS). 2025. arXiv: 2512.05620 [cs.LG].

[42] S. Raschka. *Practical Tips for Finetuning LLMs Using LoRA (Low-Rank Adaptation)*. https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms. Ahead of AI. 2023.

[43] J. Schulman and T. M. Lab. "LoRA Without Regret". In: *Thinking Machines Lab: Connectionism* (2025). https://thinkingmachines.ai/blog/lora/.

[44] A. Semenov, M. Pagliardini, and M. Jaggi. *Benchmarking Optimizers for Large Language Model Pretraining*. 2025. arXiv: 2509.01440 [cs.LG].

[45] N. Shazeer and M. Stern. *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost*. 2018. arXiv: 1804.04235 [cs.LG].

[46] S. Singh, F. Vargus, D. Dsouza, et al. *Aya Dataset: An Open-Access Collection for Multilingual Instruction Tuning*. 2024. arXiv: 2402.06619 [cs.CL].

[47] S. Toshniwal et al. *OpenMathInstruct-2: Accelerating AI for Math with Massive Open-Source Instruction Data*. 2024. arXiv: 2410.01560 [cs.CL].

[48] N. Vyas et al. *SOAP: Improving and Stabilizing Shampoo using Adam*. 2024. arXiv: 2409.11321 [cs.LG].

[49] K. Wen et al. *Fantastic Pretraining Optimizers and Where to Find Them*. 2025. arXiv: 2509.02046 [cs.LG].

[50] A. Yang et al. *Qwen2.5 Technical Report*. 2024. arXiv: 2412.15115 [cs.CL].

[51] J.-N. Yen et al. "LoRA Done RITE: Robust Invariant Transformation Equilibration for LoRA Optimization". In: *International Conference on Learning Representations*. 2025. arXiv: 2410.20625 [cs.LG].

[52] J. Zhang et al. *Gram Newton–Schulz: A Fast, Hardware-Aware Newton–Schulz Algorithm for Muon*. https://tridao.me/blog/2026/gram-newton-schulz/. 2026.

[53] Y. Zhang et al. *Mousse: Rectifying the Geometry of Muon with Curvature-Aware Preconditioning*. 2026. arXiv: 2603.09697 [cs.LG].

Appendix Contents

A Experimental Details **14**

B Missing Proofs **15**

C Momentum-Free Reductions to Product Muon **17**

D Estimating the Preconditioners **19**

E Implementation Details **22**

E.1 Spectral-Norm Estimation 22

E.2 Matrix Sign and Inverse Square Root via Gram Newton–Schulz 22

F FLOP Overhead per Optimizer Step **23**

G Additional Learning Curves **24**

H Factor Self-Balancing **26**

A Experimental Details

This section gives the full configuration behind Section 4.1. Within each setting (a model, a dataset, and a rank), the optimizer and its learning rate are the only things we vary; everything in Table 3 is held fixed. For each optimizer we sweep the learning rate and report the run with the lowest held-out loss. In the setting of Figure 1, the standard deviation of the final loss measured over four random seeds is 0.0007 for Adam and 0.0005 for PoLoRA.

Models. Four pretrained base models, each adapted with LoRA: OLMo-2-1B [38], Llama-3.2-1B [15], Qwen2.5-1.5B [50], and Llama-3-8B [15].

Datasets. Three instruction-tuning datasets: code (OpenCoder [21]), math (OpenMathInstruct-2 [47]), and Bengali (Aya [46]).

Data preparation.

- Each dataset is tokenized once per base model, using that model's own tokenizer.
- We hold out 1% of each dataset for evaluation, and use this split for both learning-rate selection and all reported losses.
- Training sequences are packed into fixed 2048-token blocks, and evaluation sequences are padded to 2048 tokens.
- Loss is computed on response tokens only; prompt and padding tokens are masked.

Fixed training configuration. Table 3 lists the settings held fixed across all runs. We adapt all linear layers except `lm_head` to isolate the efficacy of the optimizer. Prior work, however, suggests that adapting only a subset of modules recovers most of the performance [19, 43].

setting	value
sequence length	2048
batch size	16
training steps	9000
LoRA rank r	256 (varied in Section 4.4)
LoRA α	r
LoRA dropout	0
LoRA $\boldsymbol{B}$ init	zeros
LoRA-adapted layers	all linear except the output (`lm_head`)
precision	bf16
max gradient norm	1.0 (disabled for LoRA-RITE)
LR schedule	constant

Table 3: Training configuration, held fixed across optimizers and settings except where noted.

Optimizer settings. The learning-rate grid is spaced by factors of three, wide enough to bracket each optimizer's optimal learning rate. All other optimizer hyperparameters are fixed. Table 4 lists them for the baseline methods, and PoLoRA uses:

- momentum decay $\beta_1 = 0.9$ and curvature decay $\beta_2 = 0.99$,
- 8 Gram Newton–Schulz iterations (Appendix E.2),
- 8 power iterations per spectral-norm estimate (Appendix E.1),
- relative damping $\delta = 10^{-4}$ (Appendix E),
- numerical-stability constant $\varepsilon = 10^{-12}$ (Appendix E).

Baseline implementations. We run iMuon and LoRA-RITE through the authors' official implementations, with the hyperparameters in Table 4. LoRA-RITE is invariant to reparameterizations of the LoRA factors [51, Def. 1]. A global gradient-norm clip would break this invariance, so we run LoRA-RITE without it.

	Adam	Muon	iMuon	LoRA-RITE
β_1	0.9	0.9	0.95	0.9
β_2	0.999	—	—	0.999
ε	10^{-8}	—	—	10^{-6}
Newton–Schulz iterations	—	8	5	—

Table 4: Baseline hyperparameters. β_1 is the momentum decay, β_2 the second-moment decay, and ε the numerical-stability constant.

Hardware. Each run uses a single GPU. Per-step overheads (Section 4.2) are measured on an NVIDIA RTX PRO 6000 (Blackwell) with `torch.compile` enabled. Reproducing all our experiments takes about 850 GPU-hours.

B Missing Proofs

Setup. As in Section 2, $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is a linear layer with input $x \sim \mathcal{D}$ and loss $\ell(\boldsymbol{W}x)$, $\boldsymbol{G}(x) = \nabla_W \ell(\boldsymbol{W}x)$ is the per-sample gradient, and $\boldsymbol{G} = \mathbb{E}_x[\boldsymbol{G}(x)]$ is the gradient of the expected loss. The second moment $\boldsymbol{\Sigma}$ of $\boldsymbol{G}(x)$ and its Mahalanobis norm $\|\cdot\|_{\boldsymbol{\Sigma}}$ are defined in (7).

We first collect the gradient properties used to motivate the outergradient set (8).

Lemma 2.1 (Gradients are rank one)**.** For every $x \in \operatorname{support}(\mathcal{D})$ the per-sample gradient factors as $\boldsymbol{G}(x) = b(x)\, x^\top$, where $b(x) = \frac{d}{dz}\ell(z)\big|_{z=\boldsymbol{W}x} \in \mathbb{R}^{d_{\text{out}}}$ is the output-side gradient.

Proof of Lemma 2.1. The rank-one property is the chain rule: $\boldsymbol{W}$ encodes a linear layer $x \mapsto \boldsymbol{W}x$, so

$$\nabla_W \ell(\boldsymbol{W}x) = \frac{d}{dz}\ell(z)\bigg|_{z=\boldsymbol{W}x} x^\top = b(x)x^\top. \qquad \square$$

The Markov bound behind the outergradient set (8) rests on the following identity.

Lemma B.1 (Expected $\boldsymbol{\Sigma}$-norm)**.** The per-sample gradient satisfies $\mathbb{E}_x\left[\|\boldsymbol{G}(x)\|_{\boldsymbol{\Sigma}}^2\right] = \operatorname{rank}(\boldsymbol{\Sigma}) \le d_{\text{out}} d_{\text{in}}$.

Proof. By definition of $\boldsymbol{\Sigma}$ and linearity of trace and expectation,

$$\begin{aligned}
\mathbb{E}_x\left[\|\boldsymbol{G}(x)\|_{\boldsymbol{\Sigma}}^2\right] &= \mathbb{E}_x\left[\operatorname{vec}(\boldsymbol{G}(x))^\top \boldsymbol{\Sigma}^\dagger \operatorname{vec}(\boldsymbol{G}(x))\right] \\
&= \mathbb{E}_x\left[\operatorname{Tr}(\boldsymbol{\Sigma}^\dagger \operatorname{vec}(\boldsymbol{G}(x)) \operatorname{vec}(\boldsymbol{G}(x))^\top)\right] \\
&= \operatorname{Tr}(\boldsymbol{\Sigma}^\dagger \boldsymbol{\Sigma}) = \operatorname{rank}(\boldsymbol{\Sigma}),
\end{aligned}$$

which is at most $d_{\text{out}} d_{\text{in}}$ since $\boldsymbol{\Sigma} \in \mathbb{R}^{d_{\text{out}} d_{\text{in}} \times d_{\text{out}} d_{\text{in}}}$. $\square$

We verify that the per-sample LMO (5) is well-posed.

Proposition B.2 (Well-posedness of the per-sample LMO)**.** For every $\tau \ge 0$, the per-sample LMO (5) attains a finite minimum.

Proof. The feasible set is nonempty, closed, and convex, since it is an intersection of closed slabs and contains $\Delta\boldsymbol{W} = 0$. For every feasible $\Delta\boldsymbol{W}$, by Jensen's inequality we have that

$$|\langle \boldsymbol{G}, \Delta\boldsymbol{W}\rangle| \le \mathbb{E}_{x\sim\mathcal{D}}\left[|\langle \boldsymbol{G}(x), \Delta\boldsymbol{W}\rangle|\right] \le \tau,$$

so the objective $\langle \boldsymbol{G}, \Delta\boldsymbol{W}\rangle$ is bounded below by $-\tau$. Let S be the span of $\{\boldsymbol{G}(x) : x \in \operatorname{support}(\mathcal{D})\}$. The objective and constraints are unchanged by projecting $\Delta\boldsymbol{W}$ onto S, and the feasible set restricted

to S is compact. Therefore the minimum is attained. The minimizer need not be unique, since the objective is constant along $S^{\perp}$. □

Next we solve the spectral-norm LMO whose solution is the Muon update direction (Section 2).

Lemma B.3 (Spectral norm LMO)**.** Let $\boldsymbol{G} \in \mathbb{R}^{m\times n}$ and $\eta > 0$, and write $\|\boldsymbol{G}\|_{\mathrm{nuc}} = \sum_i \sigma_i(\boldsymbol{G})$ for the nuclear norm, the sum of the singular values. Then

$$\min_{\|\boldsymbol{X}\|_2 \le \eta} \langle \boldsymbol{G}, \boldsymbol{X}\rangle = -\eta \|\boldsymbol{G}\|_{\mathrm{nuc}},$$

and $\boldsymbol{X}^\star = -\eta\, \mathrm{msign}(\boldsymbol{G})$ is the unique minimizer of least Frobenius norm.

Proof of Lemma B.3. We lower-bound the objective by norm duality and exhibit a feasible point that attains it. By duality of the spectral and nuclear norms, every feasible $\boldsymbol{X}$ satisfies

$$\langle \boldsymbol{G}, \boldsymbol{X}\rangle \ge -\|\boldsymbol{G}\|_{\mathrm{nuc}}\|\boldsymbol{X}\|_2 \ge -\eta\|\boldsymbol{G}\|_{\mathrm{nuc}}.$$

Let $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ be a reduced singular value decomposition, so $\boldsymbol{S} = \mathrm{diag}(\sigma_i(\boldsymbol{G}))$. The choice

$$\boldsymbol{X} = -\eta \boldsymbol{U}\boldsymbol{V}^\top = -\eta\, \mathrm{msign}(\boldsymbol{G})$$

is feasible and satisfies

$$\langle \boldsymbol{G}, \boldsymbol{X}\rangle = -\eta\, \mathrm{Tr}(\boldsymbol{S}) = -\eta\|\boldsymbol{G}\|_{\mathrm{nuc}},$$

so it attains the lower bound and is therefore optimal. The minimum-norm statement follows from Lemma C.2 [39]. □

Replacing $\boldsymbol{\Sigma}$ by $\boldsymbol{Q}\otimes\boldsymbol{P}$ (9) in the outergradient set (8) turns the per-sample loss constraint into a preconditioned spectral-norm constraint.

Lemma B.4 (Outergradient set under a Kronecker factorization)**.** Let $\boldsymbol{P} \in \mathbb{R}^{d_{\mathrm{out}}\times d_{\mathrm{out}}}$ and $\boldsymbol{Q} \in \mathbb{R}^{d_{\mathrm{in}}\times d_{\mathrm{in}}}$ be symmetric positive definite, and for a constant $c > 0$ let

$$\mathcal{G} = \left\{ bx^\top \;\middle|\; b \in \mathbb{R}^{d_{\mathrm{out}}},\ x \in \mathbb{R}^{d_{\mathrm{in}}},\ \|bx^\top\|^2_{\boldsymbol{Q}\otimes\boldsymbol{P}} \le c \right\}.$$

Then for every $\Delta\boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{out}}\times d_{\mathrm{in}}}$,

$$\max_{\widetilde{\boldsymbol{G}}\in\mathcal{G}} \langle \widetilde{\boldsymbol{G}}, \Delta\boldsymbol{W}\rangle = \sqrt{c}\,\|\boldsymbol{P}^{1/2}\Delta\boldsymbol{W}\boldsymbol{Q}^{1/2}\|_2.$$

Proof of Lemma B.4. We evaluate the maximum by a change of variables that turns the constraint into a product of Euclidean balls. Since $\boldsymbol{P}$ and $\boldsymbol{Q}$ are positive definite, $(\boldsymbol{Q}\otimes\boldsymbol{P})^\dagger = \boldsymbol{Q}^{-1}\otimes\boldsymbol{P}^{-1}$. For $bx^\top \in \mathcal{G}$, column-major vectorization gives $\mathrm{vec}(bx^\top) = x\otimes b$, so

$$\|bx^\top\|^2_{\boldsymbol{Q}\otimes\boldsymbol{P}} = (x\otimes b)^\top(\boldsymbol{Q}^{-1}\otimes\boldsymbol{P}^{-1})(x\otimes b) = (x^\top\boldsymbol{Q}^{-1}x)(b^\top\boldsymbol{P}^{-1}b).$$

Writing $b = \boldsymbol{P}^{1/2}u$ and $x = \boldsymbol{Q}^{1/2}v$, the constraint becomes $\|u\|_2^2\|v\|_2^2 \le c$ and the objective becomes

$$\langle bx^\top, \Delta\boldsymbol{W}\rangle = b^\top\Delta\boldsymbol{W}x = u^\top\boldsymbol{P}^{1/2}\Delta\boldsymbol{W}\boldsymbol{Q}^{1/2}v.$$

Maximizing over the balls gives the spectral norm,

$$\max_{\widetilde{\boldsymbol{G}}\in\mathcal{G}} \langle \widetilde{\boldsymbol{G}}, \Delta\boldsymbol{W}\rangle = \max_{\|u\|_2\|v\|_2\le\sqrt{c}} u^\top\boldsymbol{P}^{1/2}\Delta\boldsymbol{W}\boldsymbol{Q}^{1/2}v = \sqrt{c}\,\|\boldsymbol{P}^{1/2}\Delta\boldsymbol{W}\boldsymbol{Q}^{1/2}\|_2. \qquad \square$$

We then solve that preconditioned spectral-norm LMO, which sets the PoLoRA direction.

Lemma 2.2 (Preconditioned Muon)**.** The LMO

$$\min_{\Delta\boldsymbol{W}} \langle \boldsymbol{G}, \Delta\boldsymbol{W}\rangle \quad \text{subject to} \quad |\langle \widetilde{\boldsymbol{G}}, \Delta\boldsymbol{W}\rangle| \le \tau \quad \text{for every} \quad \widetilde{\boldsymbol{G}} \in \mathcal{G}_\delta(\boldsymbol{Q}\otimes\boldsymbol{P}), \tag{10}$$

is equivalent, up to a constant rescaling of τ, to

$$\min_{\Delta \boldsymbol{W}} \langle \boldsymbol{G}, \Delta \boldsymbol{W} \rangle \quad \text{subject to} \quad \left\| \boldsymbol{P}^{1/2} \Delta \boldsymbol{W} \boldsymbol{Q}^{1/2} \right\|_2 \leq \tau, \tag{11}$$

which has the closed-form solution

$$\Delta \boldsymbol{W} = -\tau \, \boldsymbol{P}^{-1/2} \, \mathrm{msign}(\boldsymbol{P}^{-1/2} \boldsymbol{G} \boldsymbol{Q}^{-1/2}) \boldsymbol{Q}^{-1/2}. \tag{12}$$

Proof. By Lemma B.4, the maximum of $\langle \widetilde{\boldsymbol{G}}, \Delta \boldsymbol{W} \rangle$ over $\widetilde{\boldsymbol{G}} \in \mathcal{G}_\delta(\boldsymbol{Q} \otimes \boldsymbol{P})$ is a constant multiple of $\|\boldsymbol{P}^{1/2} \Delta \boldsymbol{W} \boldsymbol{Q}^{1/2}\|_2$. Since $\mathcal{G}_\delta(\boldsymbol{Q} \otimes \boldsymbol{P})$ is centrally symmetric, the constraint in (10) is equivalent to (11) once that constant is absorbed into τ.

It remains to solve the preconditioned spectral-norm LMO. Set

$$\Delta \widetilde{\boldsymbol{W}} := \boldsymbol{P}^{1/2} \Delta \boldsymbol{W} \boldsymbol{Q}^{1/2}, \qquad \widetilde{\boldsymbol{G}} := \boldsymbol{P}^{-1/2} \boldsymbol{G} \boldsymbol{Q}^{-1/2}.$$

Then $\langle \boldsymbol{G}, \Delta \boldsymbol{W} \rangle = \langle \widetilde{\boldsymbol{G}}, \Delta \widetilde{\boldsymbol{W}} \rangle$ and the constraint is $\|\Delta \widetilde{\boldsymbol{W}}\|_2 \leq \tau$. By Lemma B.3, a least Frobenius-norm solution is

$$\Delta \widetilde{\boldsymbol{W}} = -\tau \ \mathrm{msign}(\widetilde{\boldsymbol{G}}).$$

Undoing the change of variables gives

$$\Delta \boldsymbol{W} = -\tau \, \boldsymbol{P}^{-1/2} \, \mathrm{msign}(\boldsymbol{P}^{-1/2} \boldsymbol{G} \boldsymbol{Q}^{-1/2}) \boldsymbol{Q}^{-1/2},$$

which is (12). □

Remark B.5 (Second-order term in the merged update)**.** The magnitude rule bounds the *linearized* merged update; here we check that restoring the second-order term dropped in that linearization leaves the bound essentially unchanged. Writing $s = \|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2$, the rule (21) sets $\rho = \eta / s$ to bound the linearized update $\boldsymbol{B}\Delta\boldsymbol{A} + \Delta\boldsymbol{B}\boldsymbol{A}$ of (13). The actual update also contains the quadratic term $\Delta\boldsymbol{B}\Delta\boldsymbol{A}$ dropped there,

$$\Delta \boldsymbol{W} = \boldsymbol{B}\Delta\boldsymbol{A} + \Delta\boldsymbol{B}\boldsymbol{A} + \Delta\boldsymbol{B}\Delta\boldsymbol{A}.$$

With $\|\Delta\boldsymbol{A}\|_2 = \|\Delta\boldsymbol{B}\|_2 = \rho$, the triangle inequality and submultiplicativity give

$$\|\Delta \boldsymbol{W}\|_2 \leq \rho s + \rho^2 = \eta\Big(1 + \tfrac{\eta}{s^2}\Big),$$

so the actual update exceeds the target η by at most the relative amount η / s^2. At the standard initialization $\boldsymbol{B} = 0$ with $\boldsymbol{A}$ having independent entries of variance $\Theta(1/d_{\text{in}})$, we have $s = \|\boldsymbol{A}\|_2 = \Theta(1)$, so the overshoot is $O(\eta)$ and shrinks as the factors grow.

C Momentum-Free Reductions to Product Muon

In this appendix we show that the LMO updates of LoRA-RITE [51], LoRA-Muon [9], and Compositional Muon's half-split rule [28] all reduce to Product Muon (23) once their moment accumulation is removed, as does the *intrinsic spectral* LMO that Corollary 4.1 of [32] states for iMuon. The officially implemented iMuon optimizer, however, differs (see (27)).

Setup. As in Section 3, let $\boldsymbol{G} = \nabla_W \mathcal{L}(\boldsymbol{W})\big|_{\boldsymbol{W} = \boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A}}$ be the gradient with respect to the weight matrix, $\boldsymbol{G}_A = \boldsymbol{B}^\top \boldsymbol{G}$ and $\boldsymbol{G}_B = \boldsymbol{G}\boldsymbol{A}^\top$ be the factor gradients, and $\boldsymbol{S}_A = \boldsymbol{A}\boldsymbol{A}^\top$ and $\boldsymbol{S}_B = \boldsymbol{B}^\top\boldsymbol{B}$ be the $r \times r$ factor Gram matrices.

Product Muon is defined as the two factorwise LMOs (16) with $\boldsymbol{P} = \boldsymbol{I}_{d_{\text{out}}}$ and $\boldsymbol{Q} = \boldsymbol{I}_{d_{\text{in}}}$,

$$\begin{aligned} \Delta\boldsymbol{A} &\in \arg\min_{\Delta\boldsymbol{A}} \langle \boldsymbol{G}_A, \Delta\boldsymbol{A} \rangle \quad \text{subject to} \quad \|\boldsymbol{B}\Delta\boldsymbol{A}\|_2 \leq \tau \\ \Delta\boldsymbol{B} &\in \arg\min_{\Delta\boldsymbol{B}} \langle \boldsymbol{G}_B, \Delta\boldsymbol{B} \rangle \quad \text{subject to} \quad \|\Delta\boldsymbol{B}\boldsymbol{A}\|_2 \leq \tau. \end{aligned} \tag{23}$$

By Lemma 2.2, a solution to this LMO is

$$\begin{aligned}\Delta \boldsymbol{A} &= -\tau\, \boldsymbol{S}_B^{-1/2}\, \mathrm{msign}\Big(\boldsymbol{S}_B^{-1/2}\boldsymbol{G}_A\Big)\,,\\ \Delta \boldsymbol{B} &= -\tau\ \mathrm{msign}\Big(\boldsymbol{G}_B\boldsymbol{S}_A^{-1/2}\Big)\,\boldsymbol{S}_A^{-1/2}.\end{aligned} \tag{24}$$

We will use the fact that for any matrix $\boldsymbol{M}$ with full column rank and reduced SVD $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$, the matrix sign equals

$$\boldsymbol{M}(\boldsymbol{M}^\top\boldsymbol{M})^{-1/2} = \boldsymbol{U}\boldsymbol{V}^\top = \mathrm{msign}(\boldsymbol{M}). \tag{25}$$

LoRA-RITE. Algorithm 1 of LoRA-RITE [51] adds three pieces of memory to a base step. These are an EMA first moment with parameter β_1, an accumulated second moment transported across the changing factor basis, and an "escaped-mass" floor added to that second moment. Dropping all three, we get the authors' momentum-free update (their Eq. (14)). In our notation, with $\widehat{\boldsymbol{G}}_B := \boldsymbol{G}_B\boldsymbol{S}_A^{-1/2}$, it becomes

$$\Delta \boldsymbol{B} = -\eta\widehat{\boldsymbol{G}}_B(\widehat{\boldsymbol{G}}_B^\top\widehat{\boldsymbol{G}}_B)^{-1/2}\boldsymbol{S}_A^{-1/2}.$$

With no accumulation, by (25) the middle factor is the matrix sign, so

$$\Delta \boldsymbol{B} = -\eta\,\mathrm{msign}(\widehat{\boldsymbol{G}}_B)\boldsymbol{S}_A^{-1/2} = -\eta\,\mathrm{msign}\left(\boldsymbol{G}_B\boldsymbol{S}_A^{-1/2}\right)\boldsymbol{S}_A^{-1/2},$$

which is the $\boldsymbol{B}$-update solving the Product Muon LMO (23) with budget $\tau = \eta$. Analogous arguments hold for the $\boldsymbol{A}$-update.

iMuon, momentum-free. iMuon relies on a choice of *ambient matrix* $\boldsymbol{H} \in \mathbb{R}^{d_{\mathrm{out}}\times d_{\mathrm{in}}}$. The intrinsic spectral LMO of Corollary 4.1 maps this ambient matrix to the factor updates as follows

$$\Delta \boldsymbol{B}(\boldsymbol{H}) = \mathrm{msign}\left(\boldsymbol{H}\boldsymbol{A}^\top\boldsymbol{S}_A^{-1/2}\right)\boldsymbol{S}_A^{-1/2}, \qquad \Delta \boldsymbol{A}(\boldsymbol{H}) = \boldsymbol{S}_B^{-1/2}\,\mathrm{msign}\left(\boldsymbol{S}_B^{-1/2}\boldsymbol{B}^\top\boldsymbol{H}\right). \tag{26}$$

For the base Algorithm 1 in [32], the ambient matrix is the raw gradient $\boldsymbol{G}$, so (26) again gives the Product Muon update directions of (24). Our experiments benchmark the official iMuon implementation, whose momentum scheme is described in Appendix K of [32], which differs from this momentum-free decoupled update, as we show next.

iMuon, with momentum. The official implementation (described in their Appendix K) does not pass the raw gradient to the intrinsic LMO (26). It maintains momentum buffers m_A and m_B with parameter β, forms the look-aheads $\widetilde{\boldsymbol{G}}_A = \boldsymbol{G}_A + \beta m_A$ and $\widetilde{\boldsymbol{G}}_B = \boldsymbol{G}_B + \beta m_B$, and uses as input the combined ambient matrix

$$\widehat{\boldsymbol{M}} = \widetilde{\boldsymbol{G}}_B\boldsymbol{A} + \boldsymbol{B}\widetilde{\boldsymbol{G}}_A.$$

Turning momentum off ($\beta = 0$) removes the buffers but not the recombination:

$$\widehat{\boldsymbol{M}} = \boldsymbol{G}_B\boldsymbol{A} + \boldsymbol{B}\boldsymbol{G}_A = \boldsymbol{G}\,\boldsymbol{A}^\top\boldsymbol{A} + \boldsymbol{B}\boldsymbol{B}^\top\boldsymbol{G}\ \neq\ \boldsymbol{G}.$$

Plugging this $\widehat{\boldsymbol{M}}$ into (26) and using $\boldsymbol{G}_A\boldsymbol{A}^\top = \boldsymbol{B}^\top\boldsymbol{G}_B$ gives

$$\begin{aligned}\widehat{\boldsymbol{M}}\boldsymbol{A}^\top &= \boldsymbol{G}_B\boldsymbol{S}_A + (\boldsymbol{B}\boldsymbol{B}^\top)\boldsymbol{G}_B,\\ \Delta \boldsymbol{B}(\widehat{\boldsymbol{M}}) &= \mathrm{msign}\big(\boldsymbol{G}_B\boldsymbol{S}_A^{1/2} + (\boldsymbol{B}\boldsymbol{B}^\top)\boldsymbol{G}_B\boldsymbol{S}_A^{-1/2}\big)\boldsymbol{S}_A^{-1/2},\end{aligned} \tag{27}$$

which differs from the Product Muon update (24). The argument for $\Delta\boldsymbol{A}$ is symmetric. Thus Product Muon is the momentum-free limit of the iMuon theory (Corollary 4.1), but not of the implemented optimizer.

LoRA-Muon. The momentum-free LMOs of the concurrent LoRA-Muon paper [9] (its Eqs. (7) and (8)) are precisely the decoupled factorwise LMOs (23). LoRA-Muon further shows that the resulting update is equivalent to the simplified QR-coordinate form of LoRA-RITE. LoRA-Muon also shows that Spectron is not *gauge-invariant*, that is, its update to $\boldsymbol{B}\boldsymbol{A}$ changes under the rescaling $(\boldsymbol{A}, \boldsymbol{B}) \mapsto (c\boldsymbol{A}, c^{-1}\boldsymbol{B})$. Our ρ in (21) is not gauge-invariant either, but the factors stay balanced (Appendix H).

Compositional Muon. Compositional Muon [28] derives product-aware spectral updates for the attention products $\boldsymbol{W}_Q\boldsymbol{W}_K^\top$ and $\boldsymbol{W}_O\boldsymbol{W}_V$. For the QK product, its half-split rule imposes the two constraints $\|\Delta\boldsymbol{W}_Q\boldsymbol{W}_K^\top\|_2 \le \epsilon/2$ and $\|\boldsymbol{W}_Q\Delta\boldsymbol{W}_K^\top\|_2 \le \epsilon/2$ separately, and the resulting LMOs are solved by

$$\Delta\boldsymbol{W}_Q = -\tfrac{\epsilon}{2}\ \mathrm{msign}\big(\boldsymbol{G}_Q\boldsymbol{C}_K^{-1}\big)\,\boldsymbol{C}_K^{-1}, \qquad \boldsymbol{C}_K = (\boldsymbol{W}_K^\top\boldsymbol{W}_K)^{1/2},$$

where $\boldsymbol{G}_Q$ is the gradient with respect to $\boldsymbol{W}_Q$, and symmetrically for $\boldsymbol{W}_K$. Setting $\boldsymbol{W}_Q = \boldsymbol{B}$ and $\boldsymbol{W}_K = \boldsymbol{A}^\top$ gives $\boldsymbol{C}_K = (\boldsymbol{A}\boldsymbol{A}^\top)^{1/2}$ and $\boldsymbol{G}_Q = \boldsymbol{G}_B$, and the pair of updates becomes the Product Muon step (24) with $\tau = \epsilon/2$.

D Estimating the Preconditioners

In Section 2 we treated the preconditioners $\boldsymbol{P}$ and $\boldsymbol{Q}$ as given when writing the update (12). Here we describe how we compute them using online updates from the factor gradients and a normalization that makes these updates invariant to the rescaling $(\boldsymbol{P}, \boldsymbol{Q}) \mapsto (a\boldsymbol{P}, a^{-1}\boldsymbol{Q})$, arriving at the p, q updates (34) of Algorithm 1.

Notation. Consider a given linear layer. Let t denote the current optimizer step. Whenever it is clear from context, we suppress the subscript t. For each step $s \le t$, let $\boldsymbol{A}_s, \boldsymbol{B}_s$ be the LoRA factors and $\boldsymbol{G}_s \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ the gradient of the batch loss with respect to the weight matrix $\boldsymbol{W}_s = \boldsymbol{W}_0 + \boldsymbol{B}_s\boldsymbol{A}_s$. Write

$$\boldsymbol{G}_{A,s} = \boldsymbol{B}_s^\top\boldsymbol{G}_s \in \mathbb{R}^{r\times d_{\text{in}}}, \qquad \boldsymbol{G}_{B,s} = \boldsymbol{G}_s\boldsymbol{A}_s^\top \in \mathbb{R}^{d_{\text{out}}\times r} \tag{28}$$

for the factor gradients (14).

EMA averaging. The outergradient set (8) is built from the population second moment $\boldsymbol{\Sigma}$ of the per-sample gradients, but backpropagation returns only the batch gradients $\boldsymbol{G}_s$, so we estimate $\boldsymbol{\Sigma}$ online. For any sequence x_s, write the EMA of the sequence up to step t as

$$\mathrm{EMA}^{\beta}_{s\le t}[x_s] := (1-\beta)\sum_{s\le t}\beta^{t-s}x_s,$$

with decay factor β. Just as we replace the factor gradients $\boldsymbol{G}_A$ and $\boldsymbol{G}_B$ in the objective of Section 3 with averaged versions $\widehat{\boldsymbol{M}}_A$ and $\widehat{\boldsymbol{M}}_B$ (line 1 of Algorithm 1), we replace the second-moment matrix $\boldsymbol{\Sigma}$ of the per-sample gradients (7) with the EMA of the outer products of the vectorized batch gradients,

$$\overline{\boldsymbol{\Sigma}}_t := \mathrm{EMA}^{\beta_2}_{s\le t}[\mathrm{vec}(\boldsymbol{G}_s)\,\mathrm{vec}(\boldsymbol{G}_s)^\top],$$

with $\beta_2 \in [0,1]$ the curvature decay in Algorithm 1. The average $\overline{\boldsymbol{\Sigma}}_t$ is an online estimate of the second moment $\boldsymbol{\Sigma}_n$ of the batch gradient over n samples, which differs from $\boldsymbol{\Sigma}$.

Lemma D.1 (Batch-gradient second moment)**.** Let $x_1, \dots, x_n \sim \mathcal{D}$ be independent, let $\boldsymbol{G}_n = \frac{1}{n}\sum_{i=1}^n \boldsymbol{G}(x_i)$ be the batch gradient, and let $\boldsymbol{\Sigma}_n := \mathbb{E}\big[\mathrm{vec}(\boldsymbol{G}_n)\,\mathrm{vec}(\boldsymbol{G}_n)^\top\big]$. Then

$$\boldsymbol{\Sigma}_n = \tfrac{1}{n}\boldsymbol{\Sigma} + \big(1-\tfrac{1}{n}\big)\,\mathrm{vec}(\boldsymbol{G})\,\mathrm{vec}(\boldsymbol{G})^\top \succeq \tfrac{1}{n}\boldsymbol{\Sigma}, \tag{29}$$

and the set

$$\mathcal{G}_{n,\delta} := \Big\{\, bx^\top \;\Big|\; b\in\mathbb{R}^{d_{\text{out}}},\ x\in\mathbb{R}^{d_{\text{in}}},\ \|bx^\top\|^2_{\boldsymbol{\Sigma}_n} \le n\,\mathrm{rank}(\boldsymbol{\Sigma})/\delta \,\Big\}$$

satisfies $\Pr_{x\sim\mathcal{D}}\big(\boldsymbol{G}(x)\in\mathcal{G}_{n,\delta}\big) \ge 1-\delta$.

Proof. Write $g_i = \mathrm{vec}(\boldsymbol{G}(x_i))$ and $g = \mathrm{vec}(\boldsymbol{G}) = \mathbb{E}_x\,[\mathrm{vec}(\boldsymbol{G}(x))]$. By independence, $\mathbb{E}[g_ig_j^\top] = gg^\top$ for $i \ne j$ and $\mathbb{E}[g_ig_i^\top] = \boldsymbol{\Sigma}$, so

$$\boldsymbol{\Sigma}_n = \frac{1}{n^2}\sum_{i,j=1}^n \mathbb{E}[g_ig_j^\top] = \frac{1}{n}\boldsymbol{\Sigma} + \frac{n-1}{n}\,gg^\top,$$

which gives (29) since $gg^\top \succeq 0$. The mean g lies in the range of $\boldsymbol{\Sigma}$, since $v^\top\boldsymbol{\Sigma}v = 0$ forces $v^\top\,\mathrm{vec}(\boldsymbol{G}(x)) = 0$ almost surely. Hence $\boldsymbol{\Sigma}_n$ and $\boldsymbol{\Sigma}$ have the same range, on which (29) gives

$\boldsymbol{\Sigma}_n^\dagger \preceq n\boldsymbol{\Sigma}^\dagger$, so $\|\boldsymbol{M}\|^2_{\boldsymbol{\Sigma}_n} \le n\|\boldsymbol{M}\|^2_{\boldsymbol{\Sigma}}$ for every $\boldsymbol{M}$. By Lemma B.1 and Markov's inequality, $\|\boldsymbol{G}(x)\|^2_{\boldsymbol{\Sigma}} \le \operatorname{rank}(\boldsymbol{\Sigma})/\delta$ with probability at least $1-\delta$, and $\boldsymbol{G}(x)$ is rank one by Lemma 2.1, so $\boldsymbol{G}(x) \in \mathcal{G}_{n,\delta}$. □

Hence building the outergradient set (8) from $\boldsymbol{\Sigma}_n$, which $\overline{\boldsymbol{\Sigma}}_t$ estimates online, in place of $\boldsymbol{\Sigma}$ keeps the guarantee of Section 2. The factor n in the level only rescales the constraint by $\sqrt{n}$ (Lemma B.4), which is absorbed into τ.

Factor-gradient moments. As in Section 2, to make the update tractable we approximate $\overline{\boldsymbol{\Sigma}}_t \approx \boldsymbol{Q}_t \otimes \boldsymbol{P}_t$ (9) with diagonal matrices $\boldsymbol{P}_t$ and $\boldsymbol{Q}_t$. Backpropagation gives efficient access only to the factor gradients, through which we can estimate their averaged second moments,

$$\overline{\boldsymbol{\Sigma}}_{\boldsymbol{A},t} := \mathrm{EMA}^{\beta_2}_{s\le t}\big[\mathrm{vec}(\boldsymbol{G}_{A,s})\,\mathrm{vec}(\boldsymbol{G}_{A,s})^\top\big], \qquad \overline{\boldsymbol{\Sigma}}_{\boldsymbol{B},t} := \mathrm{EMA}^{\beta_2}_{s\le t}\big[\mathrm{vec}(\boldsymbol{G}_{B,s}^\top)\,\mathrm{vec}(\boldsymbol{G}_{B,s}^\top)^\top\big]. \tag{30}$$

Suppose the Kronecker approximation is exact, that is, $\overline{\boldsymbol{\Sigma}}_t = \boldsymbol{Q} \otimes \boldsymbol{P}$.

Using the chain rule (28), that is, $\boldsymbol{G}_{A,s} = \boldsymbol{B}_s^\top \boldsymbol{G}_s$ and $\boldsymbol{G}_{B,s}^\top = \boldsymbol{A}_s \boldsymbol{G}_s^\top$, it follows by vectorizing that

$$\mathrm{vec}(\boldsymbol{G}_{A,s}) = (\boldsymbol{I} \otimes \boldsymbol{B}_s^\top)\,\mathrm{vec}(\boldsymbol{G}_s), \qquad \mathrm{vec}(\boldsymbol{G}_{B,s}^\top) = (\boldsymbol{I} \otimes \boldsymbol{A}_s)\,\mathrm{vec}(\boldsymbol{G}_s^\top) = (\boldsymbol{I} \otimes \boldsymbol{A}_s)\,\mathcal{K}\,\mathrm{vec}(\boldsymbol{G}_s).$$

We used that $\mathrm{vec}(\boldsymbol{X}^\top) = \mathcal{K}\,\mathrm{vec}(\boldsymbol{X})$, where the commutation matrix $\mathcal{K}$ is a permutation matrix satisfying $\mathcal{K}(\boldsymbol{U} \otimes \boldsymbol{V})\mathcal{K}^\top = \boldsymbol{V} \otimes \boldsymbol{U}$ for square $\boldsymbol{U}$ and $\boldsymbol{V}$.

We now make an approximation by replacing $\boldsymbol{A}_s$ and $\boldsymbol{B}_s$ with their current values $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ in the above, that is, we will use

$$\mathrm{vec}(\boldsymbol{G}_{A,s}) \approx (\boldsymbol{I} \otimes \boldsymbol{B}_t^\top)\,\mathrm{vec}(\boldsymbol{G}_s), \qquad \mathrm{vec}(\boldsymbol{G}_{B,s}^\top) \approx (\boldsymbol{I} \otimes \boldsymbol{A}_t)\,\mathcal{K}\,\mathrm{vec}(\boldsymbol{G}_s).$$

Using this approximation and the mixed-product rule $(\boldsymbol{U} \otimes \boldsymbol{V})(\boldsymbol{W} \otimes \boldsymbol{Z}) = \boldsymbol{UW} \otimes \boldsymbol{VZ}$ in (30) gives

$$\begin{aligned}
\overline{\boldsymbol{\Sigma}}_{\boldsymbol{A},t} &= (\boldsymbol{I} \otimes \boldsymbol{B}_t^\top)(\boldsymbol{Q} \otimes \boldsymbol{P})(\boldsymbol{I} \otimes \boldsymbol{B}_t) = \boldsymbol{Q} \otimes \boldsymbol{C}_{B_t}(\boldsymbol{P}),\\
\overline{\boldsymbol{\Sigma}}_{\boldsymbol{B},t} &= (\boldsymbol{I} \otimes \boldsymbol{A}_t)\,\mathcal{K}(\boldsymbol{Q} \otimes \boldsymbol{P})\mathcal{K}^\top(\boldsymbol{I} \otimes \boldsymbol{A}_t^\top) = (\boldsymbol{I} \otimes \boldsymbol{A}_t)(\boldsymbol{P} \otimes \boldsymbol{Q})(\boldsymbol{I} \otimes \boldsymbol{A}_t^\top) = \boldsymbol{P} \otimes \boldsymbol{C}_{A_t}(\boldsymbol{Q}),
\end{aligned} \tag{31}$$

where $\boldsymbol{C}_{B_t}(\boldsymbol{P}) := \boldsymbol{B}_t^\top \boldsymbol{P} \boldsymbol{B}_t$ and $\boldsymbol{C}_{A_t}(\boldsymbol{Q}) := \boldsymbol{A}_t \boldsymbol{Q} \boldsymbol{A}_t^\top$.

Fitting the preconditioners. Fitting the observed moments to the predictions of the Kronecker model in (31) requires a measure of discrepancy. Following KL-Shampoo [34], we use the log-determinant divergence [30], which up to terms not involving S is

$$D(\boldsymbol{\Sigma}, S) = \mathrm{Tr}(\boldsymbol{\Sigma} S^{-1}) + \log\det S.$$

We fit each preconditioner with the other held fixed:

$$\begin{aligned}
\hat{q}(\boldsymbol{P}) &:= \arg\min_{q>0} D\big(\overline{\boldsymbol{\Sigma}}_{\boldsymbol{A},t}, \mathrm{diag}(q) \otimes \boldsymbol{C}_{B_t}(\boldsymbol{P})\big),\\
\hat{p}(\boldsymbol{Q}) &:= \arg\min_{p>0} D\big(\overline{\boldsymbol{\Sigma}}_{\boldsymbol{B},t}, \mathrm{diag}(p) \otimes \boldsymbol{C}_{A_t}(\boldsymbol{Q})\big).
\end{aligned} \tag{32}$$

A self-consistent pair (p^*, q^*) satisfies $q^* = \hat{q}(\mathrm{diag}(p^*))$ and $p^* = \hat{p}(\mathrm{diag}(q^*))$. With $\boldsymbol{P}$ fixed, the first objective in (32) separates over the coordinates of q. Minimizing each coordinate yields a closed-form solution for $\hat{q}(\boldsymbol{P})$, and likewise for $\hat{p}(\boldsymbol{Q})$:

$$\begin{aligned}
\hat{q}(\boldsymbol{P}) &= \frac{1}{r}\,\mathrm{diag}\left(\mathrm{EMA}^{\beta_2}_{s\le t}\big[\boldsymbol{G}_{A,s}^\top \boldsymbol{C}_{B_t}(\boldsymbol{P})^{-1} \boldsymbol{G}_{A,s}\big]\right),\\
\hat{p}(\boldsymbol{Q}) &= \frac{1}{r}\,\mathrm{diag}\left(\mathrm{EMA}^{\beta_2}_{s\le t}\big[\boldsymbol{G}_{B,s} \boldsymbol{C}_{A_t}(\boldsymbol{Q})^{-1} \boldsymbol{G}_{B,s}^\top\big]\right).
\end{aligned} \tag{33}$$

Each minimizer depends on the other preconditioner through $\boldsymbol{C}_{B_t}(\boldsymbol{P})$ or $\boldsymbol{C}_{A_t}(\boldsymbol{Q})$.

Online update. Re-evaluating the minimizers in (33) at each step would require storing every past factor gradient, since $\boldsymbol{C}_{B_t}(\boldsymbol{P})$ and $\boldsymbol{C}_{A_t}(\boldsymbol{Q})$ change as $\boldsymbol{P}$ and $\boldsymbol{Q}$ are updated. PoLoRA instead evaluates each factor gradient once, at its own step, and accumulates the results in the vectors p_t and q_t:

$$\begin{aligned} \hat{q}_t(\boldsymbol{P}_t) &= \frac{1}{r}\operatorname{diag}\big(\boldsymbol{G}_{A,t}^\top \boldsymbol{C}_{B_t}(\boldsymbol{P}_t)^{-1}\boldsymbol{G}_{A,t}\big), & \hat{p}_t(\boldsymbol{Q}_t) &= \frac{1}{r}\operatorname{diag}\big(\boldsymbol{G}_{B,t}\boldsymbol{C}_{A_t}(\boldsymbol{Q}_t)^{-1}\boldsymbol{G}_{B,t}^\top\big), \\ q_{t+1} &= \beta_2 q_t + (1-\beta_2)\hat{q}_t(\boldsymbol{P}_t), & p_{t+1} &= \beta_2 p_t + (1-\beta_2)\hat{p}_t(\boldsymbol{Q}_t), \\ \boldsymbol{Q}_{t+1} &= \operatorname{diag}\big(q_{t+1}/\|q_{t+1}\|_\infty\big), & \boldsymbol{P}_{t+1} &= \operatorname{diag}\big(p_{t+1}/\|p_{t+1}\|_\infty\big). \end{aligned} \tag{34}$$

Up to the exponentially decayed initialization $\beta_2^{t+1}q_0$, unrolling the q_{t+1} update in (34) gives the minimizer $\hat{q}(\boldsymbol{P})$ of (33) with the step-s matrix $\boldsymbol{C}_{B_s}(\boldsymbol{P}_s)$ in place of a fixed $\boldsymbol{C}_{B_t}(\boldsymbol{P})$, and likewise for p_{t+1}. In Algorithm 1, the preconditioner updates (lines 7 and 8) reuse the matrices $\boldsymbol{C}_{B_t}$ and $\boldsymbol{C}_{A_t}$ already computed for the direction step (line 2), so each step inverts $\boldsymbol{C}_{B_t}$ and $\boldsymbol{C}_{A_t}$ once rather than twice.

Normalization. The pair $(\boldsymbol{P}, \boldsymbol{Q})$ enters the approximation of $\overline{\boldsymbol{\Sigma}}_t$ only through the product $\boldsymbol{Q} \otimes \boldsymbol{P}$, and

$$(a^{-1}\boldsymbol{Q}) \otimes (a\boldsymbol{P}) = \boldsymbol{Q} \otimes \boldsymbol{P} \qquad \text{for every } a > 0,$$

so we want the optimizer to behave identically at $(\boldsymbol{P}, \boldsymbol{Q})$ and $(a\boldsymbol{P}, a^{-1}\boldsymbol{Q})$. The update direction is the same at both, but

$$\hat{q}_t(a\boldsymbol{P}_t) = a^{-1}\hat{q}_t(\boldsymbol{P}_t), \qquad \hat{p}_t(a^{-1}\boldsymbol{Q}_t) = a\,\hat{p}_t(\boldsymbol{Q}_t),$$

so we normalize q_{t+1}, p_{t+1} in (34) to make the p, q updates invariant as well.

E Implementation Details

This section details how Algorithm 1 initializes its state and implements its numerical subroutines.

Initialization. The EMA vectors start at $p = q = \varepsilon\mathbf{1}$, which the normalization in line 2 maps to the identity metric at the first step. Since $\boldsymbol{B} = 0$ at initialization, $\boldsymbol{G}_A$, $\widehat{\boldsymbol{M}}_A$, and $\boldsymbol{C}_B$ all vanish, so $\boldsymbol{D}_A = 0$ and $\boldsymbol{A}$ stays fixed (line 6) until $\boldsymbol{B}$ first becomes nonzero. The $\boldsymbol{B}$ factor updates from the first step, with $\boldsymbol{C}_A = \boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^\top = \boldsymbol{A}\boldsymbol{A}^\top$, so its first update coincides in direction with the Product Muon step (24). The ablation variant without magnitude control (Section 4.3) takes the step $\Delta\boldsymbol{A} = -\tau\boldsymbol{D}_A$ in place of line 6, and $\boldsymbol{D}_A$ contains $(\boldsymbol{B}^\top\boldsymbol{P}\boldsymbol{B})^{-1/2}$, which diverges as $\boldsymbol{B} \to 0$. This variant therefore uses the initialization of [31], which draws $\boldsymbol{B}_0$ at the scale of $\boldsymbol{A}_0$ and replaces the base weight by $\boldsymbol{W}_0' = \boldsymbol{W}_0 - \boldsymbol{B}_0\boldsymbol{A}_0$, so that $\boldsymbol{W}_0' + \boldsymbol{B}_0\boldsymbol{A}_0 = \boldsymbol{W}_0$.

Relative damping. Each inverse in Algorithm 1 is damped so that directions of nearly zero curvature do not blow up. For a positive semidefinite matrix $\boldsymbol{C}$ ($\boldsymbol{C}_A$ or $\boldsymbol{C}_B$ in (17)) we damp relative to the top eigenvalue,

$$\widehat{\boldsymbol{C}} = \boldsymbol{C} + \max\big(\delta\lambda_{\max}(\boldsymbol{C}),\, \varepsilon\big)\boldsymbol{I},$$

which caps the condition number of $\widehat{\boldsymbol{C}}$ at $(1+\delta)/\delta \approx 1/\delta$. We apply the same relative damping to the diagonal preconditioners $\boldsymbol{P}$ and $\boldsymbol{Q}$. Since the normalization in line 2 sets their largest entry to one, the damped matrices are simply $\boldsymbol{P} + \delta\boldsymbol{I}$ and $\boldsymbol{Q} + \delta\boldsymbol{I}$.

E.1 Spectral-Norm Estimation

Algorithm 1 requires repeated estimates of spectral norms. We estimate each $\|\boldsymbol{M}\|_2$ by warm-started power iteration on the smaller of the two Gram matrices $\boldsymbol{M}\boldsymbol{M}^\top$ and $\boldsymbol{M}^\top\boldsymbol{M}$ (Algorithm 2), caching the leading vector across optimizer steps so it starts near the top singular vector. A cold or stale start can underestimate $\|\boldsymbol{M}\|_2$, and since the optimizer divides by this estimate, that would risk destabilizing training. We therefore return the larger of the power-iteration value and the maximum row norm of $\boldsymbol{M}$, which does not depend on the cached start vector. We use $K_{\text{pow}} = 8$ iterations. Since power iteration returns a lower bound on $\|\boldsymbol{M}\|_2$, the bound $\|\Delta\boldsymbol{W}\|_2 \le \eta$ in (20) holds up to the error of this estimate.

Algorithm 2 Guarded spectral-norm estimate of $\|\boldsymbol{M}\|_2$

Input: $\boldsymbol{M} \in \mathbb{R}^{m\times n}$, $m \le n$ (else $\boldsymbol{M}^\top$); cached $v \in \mathbb{R}^m$; iterations K_{pow}

$l \leftarrow \max_i \|\boldsymbol{M}_{i,:}\|_2$
if v is invalid **then** ▷ absent, zero, or nonfinite
  $v \leftarrow \boldsymbol{M}\mathbf{1}$
end if
for $k = 1, \dots, K_{\text{pow}}$ **do** ▷ iterate the smaller Gram $\boldsymbol{M}\boldsymbol{M}^\top$
  $w \leftarrow \boldsymbol{M}^\top v$; $v \leftarrow \boldsymbol{M}w/\|\boldsymbol{M}w\|_2$
end for
return $\max\{\|\boldsymbol{M}^\top v\|_2,\ l\}$

E.2 Matrix Sign and Inverse Square Root via Gram Newton–Schulz

Beyond the spectral norm, Algorithm 1 needs the matrix sign $\operatorname{msign}(\cdot)$ and the inverse square root of the $r \times r$ curvature matrix. Both reduce to an inverse square root of a small positive semidefinite matrix: the sign is $\operatorname{msign}(\boldsymbol{X}) = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1/2}\boldsymbol{X}$ (for a full row rank matrix $\boldsymbol{X}$; for a tall $\boldsymbol{X}$, apply to $\boldsymbol{X}^\top$ and transpose the result), and the curvature term is $\boldsymbol{C}^{-1/2}$ directly. We compute that inverse square root by the Gram Newton–Schulz iteration [52] (Algorithm 3), which uses only $r \times r$ matrix multiplications and needs no eigendecomposition or SVD.

Quintic Newton–Schulz. Each iteration applies an odd degree-five polynomial

$$p_k(s) = a_k s + b_k s^3 + c_k s^5. \tag{35}$$

The coefficients (a_k, b_k, c_k) are the PolarExpress coefficients [1]. They are fit so that the composition of the K maps p_k drives every singular value in $[10^{-3}, 1]$ toward 1. We compute them once and reuse them for every optimizer step.

Gram iteration. Rather than iterating on the rectangular matrix, we iterate on its $r \times r$ Gram matrix [52]. Let $\boldsymbol{R}_k = \boldsymbol{Y}_k \boldsymbol{Y}_k^\top \in \mathbb{R}^{r \times r}$ be the Gram matrix of the rectangular iterate $\boldsymbol{Y}_k$. For $k = 1, \ldots, K$ the rectangular update and its Gram matrix evolve together in closed form,

$$\begin{aligned} \boldsymbol{M}_k &= a_k \boldsymbol{I} + b_k \boldsymbol{R}_{k-1} + c_k \boldsymbol{R}_{k-1}^2, \\ \boldsymbol{Y}_k &= \boldsymbol{M}_k \boldsymbol{Y}_{k-1}, \\ \boldsymbol{R}_k &= \boldsymbol{Y}_k \boldsymbol{Y}_k^\top = \boldsymbol{M}_k \boldsymbol{R}_{k-1} \boldsymbol{M}_k, \end{aligned}$$

so the whole iteration runs on $\boldsymbol{R}_k$ alone. Because $\boldsymbol{M}_k$ is a polynomial in $\boldsymbol{R}_{k-1}$, it shares the singular vectors of $\boldsymbol{Y}_{k-1}$ and moves only its singular values; since the coefficients (35) drive the normalized singular values to 1, we have $\boldsymbol{Y}_K \approx \operatorname{msign}(\boldsymbol{Y}_0)$. The Gram update thus implements the same quintic Newton–Schulz map on the singular values, at the cost of only $r \times r$ matrix multiplications.

Shared accumulator. The method also maintains an accumulator $\boldsymbol{Z}_k$ collecting the product of the maps $\boldsymbol{M}_k$: starting from $\boldsymbol{Z}_0 = \boldsymbol{I}$, each iteration sets $\boldsymbol{Z}_k = \boldsymbol{M}_k \boldsymbol{Z}_{k-1}$, so by induction

$$\boldsymbol{Y}_k = \boldsymbol{Z}_k \boldsymbol{Y}_0, \qquad \boldsymbol{R}_k = \boldsymbol{Z}_k \boldsymbol{R}_0 \boldsymbol{Z}_k^\top.$$

Since $\boldsymbol{R}_0 \succ 0$, $\operatorname{msign}(\boldsymbol{Y}_0)$ has orthonormal rows, so $\boldsymbol{R}_K = \boldsymbol{Y}_K \boldsymbol{Y}_K^\top \approx \boldsymbol{I}$. Each $\boldsymbol{M}_k$ is a polynomial in $\boldsymbol{R}_{k-1}$ and hence in $\boldsymbol{R}_0$, so $\boldsymbol{Z}_K$ is symmetric and commutes with $\boldsymbol{R}_0$; then $\boldsymbol{R}_K = \boldsymbol{Z}_K^2 \boldsymbol{R}_0 \approx \boldsymbol{I}$ gives $\boldsymbol{Z}_K \approx \boldsymbol{R}_0^{-1/2}$. Algorithm 3 initializes $\boldsymbol{R}_0 = \boldsymbol{S}/\gamma$, so the returned accumulator $\boldsymbol{Z} = \boldsymbol{Z}_K$ satisfies

$$\boldsymbol{Z} \approx (\boldsymbol{S}/\gamma)^{-1/2} = \sqrt{\gamma}\, \boldsymbol{S}^{-1/2}.$$

This gives both quantities Algorithm 1 needs:

- **Matrix sign.** Call Algorithm 3 with $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{X}^\top$ and $\gamma = \|\boldsymbol{X}\|_F^2$, giving
$$\operatorname{msign}(\boldsymbol{X}) = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1/2}\boldsymbol{X} \approx \boldsymbol{Z}\boldsymbol{X}/\sqrt{\gamma}.$$
- **Inverse square root.** Call Algorithm 3 with the damped curvature $\boldsymbol{S} = \widehat{\boldsymbol{C}}$ (Appendix E) and $\gamma = \operatorname{Tr}(\widehat{\boldsymbol{C}})$, giving
$$\widehat{\boldsymbol{C}}^{-1/2} \approx \boldsymbol{Z}/\sqrt{\gamma}.$$

Stability and precision. We run the Gram Newton–Schulz iteration in fp32, which is inexpensive because every matrix is only $r \times r$. We use $K = 8$ iterations.

Algorithm 3 Gram Newton–Schulz iteration

Input: positive definite $\boldsymbol{S} \in \mathbb{R}^{r \times r}$; $\gamma \geq \lambda_{\max}(\boldsymbol{S})$; coefficients $\{(a_k, b_k, c_k)\}_{k=1}^K$

$\boldsymbol{R} \leftarrow \boldsymbol{S}/\gamma$; $\boldsymbol{Z} \leftarrow \boldsymbol{I}$ ▷ normalize so $\lambda_{\max}(\boldsymbol{R}) \leq 1$
for $k = 1, \ldots, K$ **do**
  $\boldsymbol{M} \leftarrow a_k \boldsymbol{I} + b_k \boldsymbol{R} + c_k \boldsymbol{R}^2$
  $\boldsymbol{Z} \leftarrow \boldsymbol{M}\boldsymbol{Z}$; $\boldsymbol{R} \leftarrow \boldsymbol{M}\boldsymbol{R}\boldsymbol{M}$
end for
return $\boldsymbol{Z} \approx (\boldsymbol{S}/\gamma)^{-1/2}$

F FLOP Overhead per Optimizer Step

To measure the theoretical overhead of each optimizer step, we count leading-order matrix-multiplication FLOPs, with one multiply-add counted as two FLOPs. The count below is for one adapted linear layer with frozen base weight $\boldsymbol{W}$ and rank-r LoRA factors $\boldsymbol{A}, \boldsymbol{B}$,

$$\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \quad \boldsymbol{B} \in \mathbb{R}^{d_{\text{out}} \times r}, \quad \boldsymbol{A} \in \mathbb{R}^{r \times d_{\text{in}}},$$

with $r \ll d_{\text{in}}, d_{\text{out}}$. Throughout, K denotes the number of Newton–Schulz iterations, and n_{tok} denotes the number of tokens processed per optimization step.

Forward and backward. For a linear layer $Y = \boldsymbol{X}\boldsymbol{W}^\top$, the forward pass costs $2d_{\text{in}}d_{\text{out}}n_{\text{tok}}$ FLOPs. In full finetuning, given the backward gradient $\boldsymbol{G}_Y = \nabla_Y \mathcal{L}$, the backward pass computes both the input and the weight gradient,

$$\nabla_X \mathcal{L} = \boldsymbol{G}_Y \boldsymbol{W}, \qquad \nabla_W \mathcal{L} = \boldsymbol{G}_Y^\top \boldsymbol{X},$$

and costs twice the forward. LoRA freezes the base weight $\boldsymbol{W}$ and does not form $\nabla_W \mathcal{L}$, so its backward through $\boldsymbol{W}$ is a single forward-equivalent matmul:

$$\boldsymbol{C}_{\text{fwd}}^{\text{base}} = \boldsymbol{C}_{\text{bwd}}^{\text{base}} = 2d_{\text{in}}d_{\text{out}}n_{\text{tok}}. \tag{36}$$

The frozen base weights thus contribute $4Nn_{\text{tok}}$ FLOPs per step, where N is the total number of frozen adapted-layer parameters, versus the usual $6Nn_{\text{tok}}$ for full finetuning [26]. The LoRA factors add $r(d_{\text{in}} + d_{\text{out}})$ trainable parameters per layer and FLOPs of order $n_{\text{tok}}r(d_{\text{in}} + d_{\text{out}})$, which we omit from the count, making the overhead fraction below an overestimate.

Optimizer. The PoLoRA update (Algorithm 1) runs once per step and does not depend on n_{tok}. We count the operations that scale with d_{in} or d_{out}: forming

$$\boldsymbol{C}_B = \boldsymbol{B}^\top \boldsymbol{P}\boldsymbol{B}, \qquad \boldsymbol{C}_A = \boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^\top,$$

the preconditioned sign inputs, the small-side Gram matrices for the two matrix signs, applying the accumulated Gram maps to recover the signs, post-preconditioning the signs, and computing the preconditioner updates

$$\operatorname{diag}(\boldsymbol{G}_A^\top \boldsymbol{C}_B^{-1} \boldsymbol{G}_A), \qquad \operatorname{diag}(\boldsymbol{G}_B \boldsymbol{C}_A^{-1} \boldsymbol{G}_B^\top).$$

With diagonal $\boldsymbol{P}, \boldsymbol{Q}$, these operations contribute $12r^2(d_{\text{in}} + d_{\text{out}})$ leading FLOPs.

The Newton–Schulz iterations themselves run in the small $r \times r$ Gram space. One Gram Newton–Schulz iteration costs $8r^3$ FLOPs in the notation of Appendix E.2: $2r^3$ to form R^2, $2r^3$ for $Z \leftarrow MZ$, and $4r^3$ for $R \leftarrow MRM$. Each layer uses four calls of K iterations each (two matrix signs, two curvature inverse square roots), so the total small-matrix work is $32Kr^3$. Ignoring lower-order elementwise operations, diagonal scalings, and spectral-norm power iterations, the leading optimizer cost is

$$\boldsymbol{C}_{\text{opt}} = 12r^2(d_{\text{in}} + d_{\text{out}}) + 32Kr^3. \tag{37}$$

Overhead fraction. Dividing (37) by (36) gives the per-layer optimizer FLOP overhead

$$\frac{\boldsymbol{C}_{\text{opt}}}{\boldsymbol{C}_{\text{fwd}}^{\text{base}} + \boldsymbol{C}_{\text{bwd}}^{\text{base}}} = \frac{12r^2(d_{\text{in}} + d_{\text{out}}) + 32Kr^3}{4d_{\text{in}}d_{\text{out}}n_{\text{tok}}} = \frac{3r^2}{n_{\text{tok}}}\left(\frac{1}{d_{\text{in}}} + \frac{1}{d_{\text{out}}}\right) + \frac{8Kr^3}{d_{\text{in}}d_{\text{out}}n_{\text{tok}}}. \tag{38}$$

For fixed model dimensions, rank, and Newton–Schulz iteration count, the optimizer FLOP fraction decays as $1/n_{\text{tok}}$. The first term is governed by the smaller of the two layer dimensions. The ratio of the second term to the first is $8Kr/(3(d_{\text{in}} + d_{\text{out}}))$, so the Newton–Schulz term matters most on the square attention matrices and least on the wide feed-forward matrices. The whole-model overhead averages these fractions with weights proportional to layer size, so it is set mainly by the large feed-forward matrices, which have the smallest fraction.

Benefits of Gram Newton–Schulz. The $32Kr^3$ cost of the Gram Newton–Schulz iteration is the only term that depends on K. The iteration is used to compute the matrix sign and the curvature inverse square roots on the small $r \times r$ Gram matrix rather than on the tall $d \times r$ factor (Appendix E.2), saving a factor of about $d/(2r)$ per iteration. Iterating on the factor instead would make the optimizer step roughly $2\times$ costlier, increasing its share of the wall-clock time.

G Additional Learning Curves

Table 1 shows the step and wall-clock speedups over Adam across model families on code and math. Figure 5 complements this with the evaluation loss over training steps: rows (a) and (b) span the model families on code and math, and row (c) the rank sweep. Each panel shows Adam and PoLoRA at their optimal learning rates.

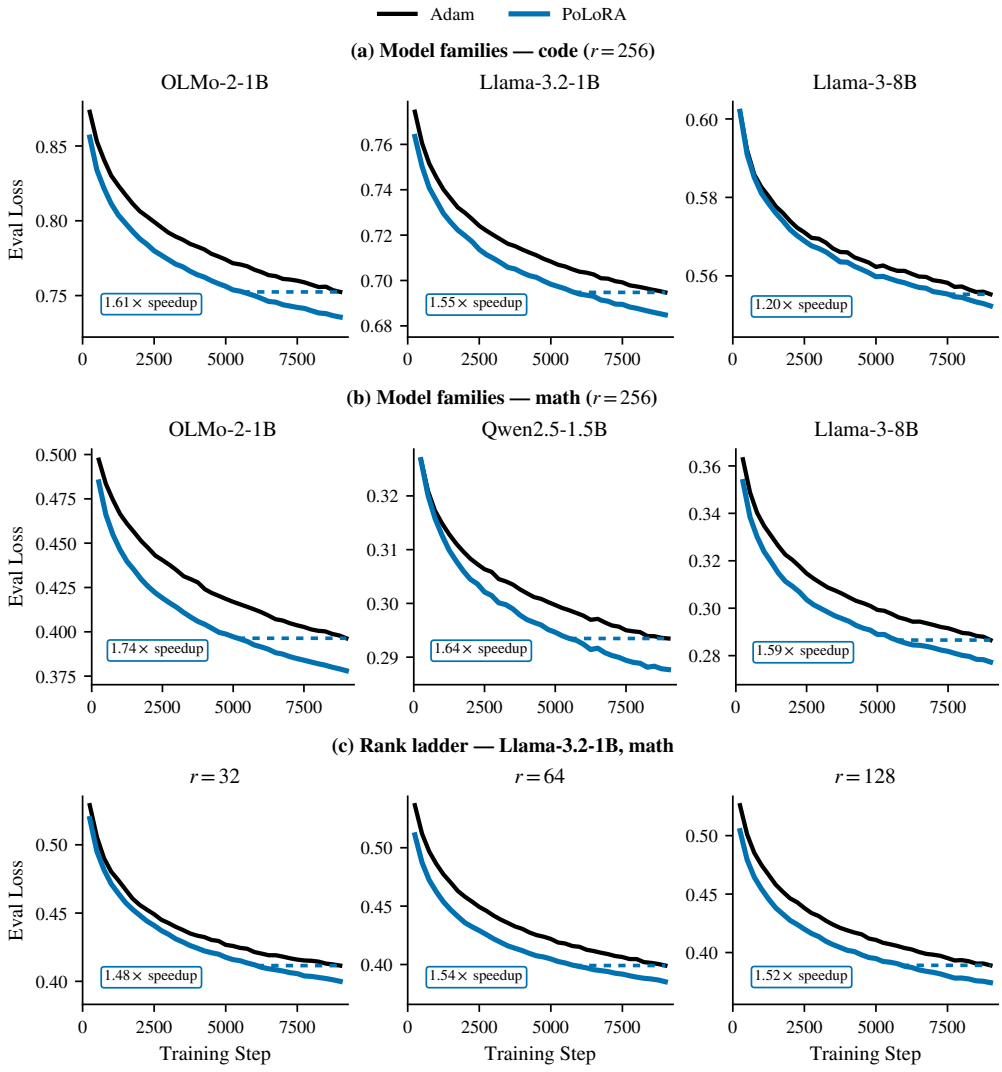

Figure 5: PoLoRA speedup over Adam across settings. Evaluation loss over training steps, Adam vs PoLoRA at each setting's best learning rate. **(a)** model families finetuned on code and **(b)** on math, at rank $r = 256$; **(c)** the rank sweep on Llama-3.2-1B (math). The dashed span marks the steps PoLoRA saves in reaching Adam's final loss. Settings already drawn as curves in the main text (Figures 1, 2 and 4) are not repeated.

H FACTOR SELF-BALANCING

The update size $\rho = \eta/(\|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2)$ (21) depends on the factorization of the merged product $\boldsymbol{BA}$. A rescaling $(\boldsymbol{A}, \boldsymbol{B}) \mapsto (c\boldsymbol{A}, c^{-1}\boldsymbol{B})$ with $c > 0$ preserves $\boldsymbol{BA}$, and hence the layer's output, while changing the denominator to

$$c\|\boldsymbol{A}\|_2 + c^{-1}\|\boldsymbol{B}\|_2 \;\geq\; 2\sqrt{\|\boldsymbol{A}\|_2\|\boldsymbol{B}\|_2}.$$

Equality holds when the two rescaled norms are equal, so an imbalanced factorization of the same product takes smaller factor steps. LoRA-Muon [9] raises this concern for Spectron [22], which sets the spectral norm of the factor updates to $\eta/(\|\boldsymbol{A}\|_2 + \|\boldsymbol{B}\|_2 + 1)$ (Section 5), but the factors stay nearly balanced. Starting from $\boldsymbol{B} = 0$, the ratio $\|\boldsymbol{B}\|_2/\|\boldsymbol{A}\|_2$ climbs and settles near 1 across ranks (Figure 6).

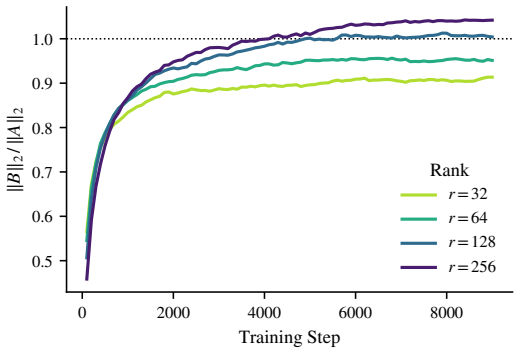

Figure 6: Factor norm self-balancing in PoLoRA. Spectral-norm ratio $\|\boldsymbol{B}\|_2/\|\boldsymbol{A}\|_2$ over training steps for Llama-3.2-1B finetuned on the math dataset at each rank. The ratio settles near 1.