APPROXIMATE MUON WITH LOW-RANK ADAPTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

The Muon optimizer shows clear benefits versus alternatives when pretraining neural networks. However, it is used less frequently for parameter-efficient fine-tuning (PEFT). One potential reason is that the most common PEFT method, LoRA, does not naturally combine with Muon since it is not mathematically possible to orthogonalize the weight update given by a low-rank parameterization. In this paper, we address this issue by approximating the solution to a relaxed Muon objective in the low-rank setting via linearization and then least-squares. We provide an efficient implementation that uses matmul operations only, as opposed to more complex linear algebra decomposition routines. Our method, sMuon (small Muon), performs favourably across SFT and a ReLoRA pretraining experiment. While results are model- and eval-dependent, we find overall that using Muon for low-rank fine-tuning provides moderate performance improvements.

1 INTRODUCTION

Muon (Jordan et al., 2024b) is a neural network optimizer which orthogonalizes updates at each layer, equalizing all singular values. This can be interpreted as steepest descent under the spectral norm (Bernstein & Newhouse, 2024b), thereby controlling the worst-case change to the layer as a linear operator, rather than viewing it as a flattened vector. Muon has demonstrated excellent results in the pretraining of LLMs, often outperforming AdamW, leading to its increasing adoption (Liu et al., 2025; Shah et al., 2025; Kimi Team, 2025; Team et al., 2026; Xu et al., 2026). However, Muon is less commonly used for fine-tuning (FT), particularly PEFT (Parameter-Efficient Fine-Tuning). One difficulty is that Muon does not straightforwardly combine with LoRA (Hu et al., 2021), the dominant PEFT method. In principle, Muon requires that we orthogonalize the descent step for the whole layer, but this is not generally realizable in the LoRA parameterization, necessitating approximation. The simplest and most common workaround is 'per-factor Muon', which ignores the geometry of the whole layer and orthogonalizes changes to B and A separately (Qu et al., 2026; Janson et al., 2026; Kang et al., 2026). Other works retain the whole layer's geometry and introduce an approximation elsewhere, with Cesista et al. (2026) relaxing the Muon objective using the triangle inequality, and Bogachev et al. (2026) recasting the problem as Riemannian optimization on a low-rank manifold. Alternative approaches that more closely fit with Muon's original 'steepest descent under the spectral norm' motivation remain underexplored.

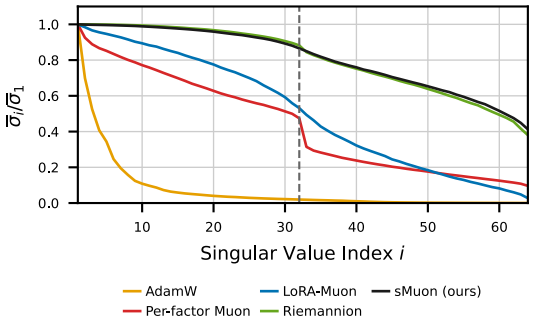

Figure 1: Average singular values of the weight updates (attention weight, layer 6 of 12), during ReLoRA pretraining ($r = 32$), normalized by $\overline{\sigma}_1$. Grey dashed line is the LoRA rank.

In this paper, we linearize the low-rank Muon objective and use least squares to find the step closest to the optimal Muon step. This approximation orthogonalizes the gradient and projects it onto the appropriate low-rank subspace. Empirically, we find this retains a more uniform spectrum (Fig. 1) than alternatives such as per-factor Muon and LoRA-Muon (Cesista et al., 2026), while avoiding expensive matrix decompositions required by Riemannion (Bogachev et al., 2026). The resulting

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.

algorithm is intuitive (Alg. 1), but a naive implementation requires orthogonalizing a full-size matrix. We therefore provide a mathematically equivalent formulation (Alg. 2) that is fast and memory-efficient, only requiring small matrix-multiplications. Finally, we show that our optimizer, sMuon (small Muon), achieves competitive performance across a range of commonsense and coding SFT benchmarks, as well as in ReLoRA pretraining experiments. In summary:

- We derive a method for training LoRA adapters with Muon by taking a first-order approximation of the spectral-norm-regularized steepest descent objective. The method is available in closed form, and requires less optimizer state than AdamW.
- We provide a matmul-only algorithm of the method, sMuon, that is efficient for modern hardware, with comparable speed to other Muon methods such as per-factor Muon and LoRA-Muon, while being significantly faster than Riemannion for larger LoRA rank.
- We run SFT experiments on four base models, three AdamW-pretrained and one Muon-pretrained, over eleven benchmarks to compare low-rank Muon optimizers. We find that our optimizer gives strong performance, performing best on the Muon-pretrained model with the top accuracy in six out of eleven tasks with Moonlight-16B-A3B.
- In the ReLoRA pretraining setting, we illustrate that Muon LoRA methods are generally stronger than AdamW LoRA. Specifically we find that our method attains the lowest final validation loss, tying with Riemannion, but being $\sim 30\%$ faster.

2 Related Work

Muon and spectral steepest descent. Muon orthogonalizes the gradient descent step for each weight matrix, so that the singular values of weight updates are equal (Jordan et al., 2024b). This update is exactly steepest descent under the spectral norm (Bernstein & Newhouse, 2024a; Bernstein, 2025). Muon has been shown to scale to billion- and trillion-parameter models for LLM pretraining, with considerable efficiency gains over AdamW (Liu et al., 2025; Kimi Team, 2025).

Muon for fine-tuning and optimizer mismatch. Although Muon excels at pretraining, it is used far less for fine-tuning. Liu et al. (2025) note that fine-tuning an AdamW-pretrained model with Muon (or vice-versa) degrades performance. Qu et al. (2026) suggest that the differing implicit biases of Adam and Muon are to blame, but that constraining the update (e.g. via LoRA) narrows the gap between Muon and AdamW fine-tuning. Muon's increasing adoption makes this question more important than ever, so we evaluate on both AdamW- and Muon-pretrained base models.

Geometry-aware and Muon-based LoRA optimizers. The most closely related works are Cesista et al. (2026) and Bogachev et al. (2026); these both attempt to approximate a low-rank solution to the spectral-regularized gradient descent problem. LoRA-Muon (Cesista et al., 2026) is derived by approximating the spectral steepest-descent optimization problem using the triangle inequality applied to the spectral norm regularizer, and then minimizing to first order. We similarly linearize (we expect second-order terms to be suitably small), but we avoid the triangle inequality, as this is not necessarily tight under the spectral norm. For their 'Riemannion' optimizer, Bogachev et al. (2026) present the problem as Riemannian optimization on a constrained low-rank manifold. While some key ideas of this method are similar to our method (projecting onto the low-rank space), Riemannian optimization also involves a 'retract' step. One issue with their retraction is requiring a truncated SVD, which cannot be efficiently replaced with an efficient matmul-only approximation. In practice this leads to Riemannion being slower than alternatives, especially for larger LoRA rank. By contrast, we obtain the step using least squares on the orthogonalized gradient direction, avoiding expensive manifold constraints that don't appear to benefit model performance. The PoLoRA (Ghosh et al., 2026) method is again similar to our work, but can be understood as a version of LoRA-Muon with additional preconditioning. LoRA-RITE (Yen et al., 2025) and Riemannian-preconditioned LoRA (Zhang & Pilanci, 2024) attempt to keep the adapters invariant to parameterization. Another popular approach is to simply apply Muon to B and A independently (Qu et al., 2026; Janson et al., 2026; Mo et al., 2025). This ignores the geometry of the linear layer and causes the update to depend on the adapter parameterization, whereas our update is first-order parameterization-invariant (see App. A.3). Appendix C has a more detailed comparison of the various approaches.

Low-rank optimization beyond LoRA fine-tuning. Some works, such as Spectron (Janson et al., 2026), LORO (Mo et al., 2025), GaLore (Zhao et al., 2024), and LoRA-Pre (Wang et al., 2026) target

efficient low-rank pretraining, either through stabilization, or optimizer state compression. He et al. (2025) accelerate Muon itself with a low-rank orthogonalization routine that exploits the low-rank structure of the gradient, but operate on full dense weight matrices rather than in the LoRA adapter setting. In contrast to all of these, our goal is a simple, principled, memory-efficient Muon update for the composite LoRA adapter $\Delta W = BA$, rather than the full base weight W_0.

3 Background

3.1 Muon

Given a gradient matrix $G_W \in \mathbb{R}^{d_1 \times d_2}$ for weight matrix $W \in \mathbb{R}^{d_1 \times d_2}$, Muon selects the step direction $\delta W^\star \in \mathbb{R}^{d_1 \times d_2}$ through a regularized minimization under the spectral norm $\|\cdot\|_2$ (largest singular value) (Bernstein & Newhouse, 2024a):

$$\delta W^\star = \arg\min_{\delta W} \left[\langle G_W, \delta W \rangle + \frac{\lambda}{2} \|\delta W\|_2^2 \right]. \tag{1}$$

The solution, assuming G_W is full rank, is proportional to $-\operatorname{msign}(G_W)$, where $\operatorname{msign}(X) = UV^\top$ for singular value decomposition $X = U\Sigma V^\top$, thereby orthogonalizing the gradient (or, in practice, the momentum). The solution is not unique for low-rank G_W, though we use the convention that $\operatorname{msign}(X) = U \operatorname{sign}(\Sigma) V^\top$, so that non-zero singular values are promoted, and the rest remain equal to zero.

3.2 LoRA

LoRA (Hu et al., 2021) is a parameter-efficient method for fine-tuning neural networks. LoRA adds adapters in the form $\Delta W = BA$ to the pretrained base weight W_0 of a layer, with $B \in \mathbb{R}^{d_1 \times r}$ and $A \in \mathbb{R}^{r \times d_2}$ where $r \ll d_1, d_2$. Typically B is initialized to zero to ensure that, initially, $\Delta W = 0$. The factors B, A are usually chosen by minimizing the fine-tuning loss with the AdamW optimizer. We instead seek to find the analogue to Muon's whitened update in the LoRA setting.

4 Method

Let $\Delta W = BA$ be the adapter with gradient $G_{\Delta W}$. Our goal is to find updates $\delta B, \delta A$ for the factors B, A such that the implied update $\delta \Delta W$ to the adapter ΔW has an approximately flat spectrum. We emphasize that simply orthogonalizing the factor updates $\delta B, \delta A$ does not imply that $\delta \Delta W$ itself is orthogonal. Under spectral steepest descent, the optimal adapter update $\delta \Delta W^\star$ is given by

$$\delta \Delta W^\star = \arg\min_{\delta \Delta W} \left[\left\langle G_{\Delta W},\, \delta \Delta W \right\rangle + \tfrac{\lambda}{2} \left\| \delta \Delta W \right\|_2^2 \right]. \tag{2}$$

If $r = \min(d_1,\, d_2)$ and B, A have full rank, then ΔW is full rank and this is an identical problem to Eq. (1), which is solved by whitening the gradient (Bernstein & Newhouse, 2024a), i.e.,

$$\delta \Delta W^\star = -\eta \operatorname{msign}(G_{\Delta W}), \tag{3}$$

where proportionality constants have been absorbed into η. However, we do not get the same solution under a low-rank constraint $r < \min(d_1, d_2)$. We have,

$$\delta \Delta W = (B + \delta B)(A + \delta A) - BA = \delta B\, A + B\, \delta A + \delta B\, \delta A, \tag{4}$$

but generally it is not possible to find $\delta B,\ \delta A$ that satisfy $\delta \Delta W = -\eta \operatorname{msign}(G_{\Delta W})$. The problem here is that the change $\delta \Delta W$ has rank at most $2r$, but $\operatorname{msign}(G_{\Delta W})$ is usually full rank.

We proceed by writing Eq. (2) in terms of the adapter matrices,

$$(\delta B^\star, \delta A^\star) = \arg\min_{\delta B, \delta A} \left[\left\langle G_{\Delta W},\, \delta B\, A + B\, \delta A + \delta B\, \delta A \right\rangle + \tfrac{\lambda}{2} \left\| \delta B\, A + B\, \delta A + \delta B\, \delta A \right\|_2^2 \right]. \tag{5}$$

This is a non-linear optimization problem due to the second-order terms, and it has no known closed-form solution. For small updates $\delta B, \delta A$, we approximate by dropping these second-order terms, giving

$$(\delta B^\star, \delta A^\star) = \arg\min_{\delta B, \delta A} \left[\left\langle G_{\Delta W},\, \delta B\, A + B\, \delta A \right\rangle + \tfrac{\lambda}{2} \left\| \delta B\, A + B\, \delta A \right\|_2^2 \right]. \tag{6}$$

Since it is generally not possible to achieve the rank-unconstrained minimum, we instead try to get as close as possible. We solve a least-squares problem for the linearized update map (overloading the notation $\delta B^\star$, $\delta A^\star$ for convenience),

$$(\delta B^\star, \delta A^\star) = \underset{\delta B,\, \delta A}{\arg\min} \left\| \delta\Delta W^\star - (\delta B\, A + B\, \delta A) \right\|_F^2. \tag{7}$$

Taking gradients of the objective w.r.t. δB and δA and setting to zero, we have,

$$0 = -2(\delta\Delta W^\star - \delta B\, A - B\, \delta A)A^\top, \tag{8a}$$

$$0 = -2B^\top(\delta\Delta W^\star - \delta B\, A - B\, \delta A). \tag{8b}$$

Assuming B and A have full rank, right-multiplying Eq. (8a) by $(AA^\top)^{-1}$ and left-multiplying Eq. (8b) by $(B^\top B)^{-1}$ gives

$$\delta\Delta W^\star A^\dagger = \delta B\, AA^\dagger + B\, \delta A\, A^\dagger = \delta B + B\delta A\, A^\dagger, \tag{9a}$$

$$B^\dagger\delta\Delta W^\star = B^\dagger\delta B\, A + B^\dagger B\, \delta A = B^\dagger\delta B\, A + \delta A, \tag{9b}$$

where $B^\dagger = (B^\top B)^{-1}B^\top$ and $A^\dagger = A^\top(AA^\top)^{-1}$ are the Moore-Penrose inverses satisfying $B^\dagger B = I = AA^\dagger$. Note that the minimizer is not unique: if $(\delta B, \delta A)$ is a minimizer of Eq. (7), then so is $(\delta B + BS, \delta A - SA)$ since,

$$\delta B\, A + B\, \delta A = (\delta B + BS)A + B\, (\delta A - SA). \tag{10}$$

These pairs give the same linearized weight change. Choosing $S = -B^\dagger\delta B$ makes $B^\dagger(\delta B + BS) = 0$, so Eq. (9b) shows that

$$\delta A^\star = B^\dagger\delta\Delta W^\star \tag{11a}$$

is one particular minimizer, and then substituting into Eq. (9a) gives the corresponding $\delta B^\star$,

$$\delta B^\star = (I - BB^\dagger)\delta\Delta W^\star A^\dagger. \tag{11b}$$

4.1 Low-rank gradients

While the solution $(\delta B^\star, \delta A^\star)$ given by Eq. (11) is simple, it is not efficiently computed. The $\delta\Delta W^\star = -\eta\,\mathrm{msign}(G_{\Delta W})$ term requires orthogonalization of a large matrix (remember that $G_{\Delta W} \in \mathbb{R}^{d_1\times d_2}$), which we would like to avoid. Fortunately, we do not actually require the full gradient $G_{\Delta W}$, and can instead use the low-rank gradient,

$$H := G_{\Delta W} - (I - P_B)\, G_{\Delta W}\, (I - P_A) = P_B G_{\Delta W} + (I - P_B)G_{\Delta W} P_A, \tag{12}$$

where $P_B = BB^\dagger$ and $P_A = A^\dagger A$ are projections onto the column space of B and row space of A respectively. The low-rank gradient H is the projection of the gradient onto the subspace of weight updates that the current LoRA adapter can express. The two gradients H and $G_{\Delta W}$ are first-order equivalent, in the sense that for any $\delta B, \delta A$,

$$\langle G_{\Delta W},\, \delta B\, A + B\delta A\rangle = \langle H, \delta B\, A + B\delta A\rangle. \tag{13}$$

Thus when linearizing the Muon objective, we can replace $G_{\Delta W}$ with H to obtain,

$$(\delta B, \delta A) = \underset{\delta B,\delta A}{\arg\min} \left[\langle H,\, \delta B\, A + B\, \delta A\rangle + \tfrac{\lambda}{2}\|\delta B\, A + B\, \delta A\|_2^2\right]. \tag{14}$$

In particular, this means we use $\delta\Delta W^\star = -\eta\,\mathrm{msign}(H)$. We show in the next section how this can be exploited to compute the update efficiently.

4.2 Efficient, matmul-only computation

We now show how to compute the descent steps $\delta B^\star$ and $\delta A^\star$ efficiently. One of the principal advantages of our method versus Riemannion is that it permits a matmul-only solution — Riemannion requires truncated SVDs during the retraction step, which makes parallelization/batching of computation slower (see Table 4). In particular, the most expensive operations required are $d \times r$ matmuls, which are easily optimized with deep learning libraries.

Algorithm 1: sMuon

Input: $B \in \mathbb{R}^{d_1 \times r}$, $A \in \mathbb{R}^{r \times d_2}$; momenta M_B, M_A; lr η; weight decay λ

// Low-rank momentum

$H \leftarrow (B^\dagger)^\top M_A + (I - BB^\dagger)\, M_B\, (A^\dagger)^\top$

// Compute step

$\delta A \leftarrow B^\dagger \operatorname{msign}(H)$

$\delta B \leftarrow (I - BB^\dagger) \operatorname{msign}(H)\, A^\dagger$

// Update parameters (split wd)

$s \leftarrow \sqrt{1 - \lambda\eta}$

$B \leftarrow s\, B - \frac{\eta}{s}\, \delta B$

$A \leftarrow s\, A - \frac{\eta}{s}\, \delta A$

// Project momentum

$M_B \leftarrow H\, A^\top$

$M_A \leftarrow B^\top H$

Algorithm 2: Efficient/matmul-only sMuon

Input: Same as Alg. 1; jitter $\epsilon > 0$

$S_B \leftarrow \texttt{invroot}(B^\top B + \epsilon I)$

$S_A \leftarrow \texttt{invroot}(AA^\top + \epsilon I)$

// Factor $H = BX + YA$

$X \leftarrow S_B^2 M_A$

$Y \leftarrow M_B S_A^2 - B S_B^2 (B^\top M_B S_A^2)$

// Bases for $\mathrm{col}(H)$, $\mathrm{row}(H)$

$U_2 \leftarrow \operatorname{msign}(Y)$

$V_1 \leftarrow \operatorname{msign}(A^\top)$

$V_2 \leftarrow \operatorname{msign}(X^\top - V_1(V_1^\top X^\top))$

// $2r \times 2r$ core

$\Omega \leftarrow \operatorname{msign}\left(\begin{bmatrix} S_B M_A V_1 & S_B M_A V_2 \\ (U_2^\top Y)(AV_1) & 0 \end{bmatrix}\right)$

// Compute step

$\delta A \leftarrow S_B(\Omega_{11} V_1^\top + \Omega_{12} V_2^\top)$

$\delta B \leftarrow U_2\, \Omega_{21} S_A$

// Update parameters

$s \leftarrow \sqrt{1 - \lambda\eta}$

$B' \leftarrow sB - \frac{\eta}{s}\delta B$,

$A' \leftarrow sA - \frac{\eta}{s}\delta A$

// Project momentum

$M_A \leftarrow (B'^\top B)X + (B'^\top Y)A$

$M_B \leftarrow B(XA'^\top) + Y(AA'^\top)$

$B \leftarrow B'$, $A \leftarrow A'$

The gradients for B and A are $G_B = G_{\Delta W} A^\top$ and $G_A = B^\top G_{\Delta W}$ respectively; the low-rank gradient of Eq. (12) can therefore be written as,

$$H = B \underbrace{(B^\top B)^{-1} G_A}_{=:X} + \underbrace{(I - P_B)\, G_B\, (AA^\top)^{-1}}_{=:Y} A, \tag{15}$$

where $X \in \mathbb{R}^{r \times d_2}$ and $Y \in \mathbb{R}^{d_1 \times r}$. Since B, A, X, Y have rank at most r, we have $\operatorname{rank}(H) \le 2r$.

The key identity we exploit is as follows. Let $U \in \mathbb{R}^{d_1 \times 2r}$, $V \in \mathbb{R}^{d_2 \times 2r}$ be any matrices with orthonormal columns, such that the column space of U contains the column space of H and the column space of V contains the row space of H, then[1],

$$\operatorname{msign}(H) = U \operatorname{msign}(U^\top H V)\, V^\top. \tag{16}$$

Eq. (16) lets us convert a $d_1 \times d_2$ orthogonalization into a $2r \times 2r$ orthogonalization, and we will show shortly that instantiating $H \in \mathbb{R}^{d_1 \times d_2}$ can be avoided. To choose U and V, note that since $H = BX + YA$, its column space satisfies $\mathrm{col}(H) \subseteq \mathrm{col}(B) + \mathrm{col}(Y)$, while its row space satisfies $\mathrm{row}(H) \subseteq \mathrm{row}(A) + \mathrm{row}(X)$. We therefore construct orthonormal bases for these two spaces as,

$$U = \begin{pmatrix} U_1 & U_2 \end{pmatrix}, \qquad V = \begin{pmatrix} V_1 & V_2 \end{pmatrix}, \tag{17}$$

with each block a polar factor (and therefore orthonormal columns) in the appropriate space,

$$U_1 = B(B^\top B)^{-1/2}, \qquad U_2 = Y(Y^\top Y)^{-1/2}, \tag{18a}$$

$$V_1 = A^\top (AA^\top)^{-1/2}, \qquad V_2 = Z(Z^\top Z)^{-1/2}, \qquad \text{where } Z = (I - P_A)X^\top. \tag{18b}$$

[1] To see why Eq. (16) is true, note that since $UU^\top$ and $VV^\top$ act as the identity on the column and row spaces of H, we have $H = U(U^\top H V)V^\top$. Let $P, \Sigma, Q^\top = \mathrm{SVD}(U^\top H V)$. Then $H = (UP)\Sigma(VQ)^\top$ is a valid, albeit truncated, SVD of H (UP and VQ have orthonormal columns), so $\operatorname{msign}(H) = (UP)(VQ)^\top = U(PQ^\top)V^\top = U \operatorname{msign}(U^\top H V)\, V^\top$.

We have $U_1^\top U_2 = 0$ since $B^\top Y = B^\top (I - P_B) G_B (AA^\top)^{-1} = 0$, and $V_1^\top V_2 = 0$ since $AZ = A(I - P_A)X^\top = 0$. So U and V have orthonormal columns, satisfying the conditions for Eq. (16).

Let $S_B = (B^\top B)^{-1/2}$ and $S_A = (AA^\top)^{-1/2}$. We calculate $U^\top HV$ efficiently,

$$\begin{aligned} U^\top HV &= \begin{pmatrix} U_1^\top B & U_1^\top Y \\ U_2^\top B & U_2^\top Y \end{pmatrix} \begin{pmatrix} XV_1 & XV_2 \\ AV_1 & AV_2 \end{pmatrix} \\ &= \begin{pmatrix} S_B^{-1} & 0 \\ 0 & U_2^\top Y \end{pmatrix} \begin{pmatrix} XV_1 & XV_2 \\ AV_1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} S_B G_A V_1 & S_B G_A V_2 \\ (U_2^\top Y)(AV_1) & 0 \end{pmatrix}, \end{aligned} \tag{19}$$

which avoids instantiating H directly. Writing its orthogonalization in blocks of size $r \times r$,

$$\Omega := \mathrm{msign}(U^\top HV) = \begin{pmatrix} \Omega_{11} & \Omega_{12} \\ \Omega_{21} & \Omega_{22} \end{pmatrix}, \tag{20}$$

we now calculate the final factor updates $\delta B^\star$ and $\delta A^\star$. Substituting $\mathrm{msign}(H) = U\Omega V^\top$ into the previous expressions for $\delta B^\star$ and $\delta A^\star$ (Eq. 11) gives,

$$\delta A^\star = -\eta B^\dagger U \Omega V^\top = -\eta S_B \left(\Omega_{11} V_1^\top + \Omega_{12} V_2^\top \right), \tag{21a}$$

where we used $B^\dagger U_1 = S_B$ and $B^\dagger U_2 = 0$. Similarly,

$$\delta B^\star = -\eta (I - P_B) U \Omega V^\top A^\dagger = -\eta U_2 \, \Omega_{21} \, S_A, \tag{21b}$$

using $(I - P_B)U_1 = 0$, $(I - P_B)U_2 = U_2$, $V_1^\top A^\dagger = S_A$, and $V_2^\top A^\dagger = 0$.

All components of $\delta B^\star$ and $\delta A^\star$ can be computed with matmuls, matrix inverse roots, and msign. In practice, both the $r \times r$ inverse square roots and the msign's can be evaluated with matrix polynomial iterations (Amsel et al., 2026; Zhang et al., 2026), which are themselves sequences of matrix products.

4.3 Algorithm

We combine the above updates with split weight decay (Cesista et al., 2026) to match the weight decay dynamics of full-parameter training to first order, and momentum projection (inspired by momentum transport of Bogachev et al. (2026), see App. A.3 for details) to ensure reparameterization invariance, yielding the sMuon optimizer for LoRA adapters. We provide Alg. 1 as a pedagogical version, while Alg. 2 incorporates the efficient matmul-only computations from the previous section.

We also discuss a more numerically stable variant in Appendix B. One cause for concern is the practice of initializing the adapter $\Delta W = BA = 0$, and then computing inverse roots of $B^\top B$ and $AA^\top$. We found that, despite initializing $\Delta W = 0$, calculating inverse roots of $AA^\top$ and $B^\top B$ was not an issue when using jitter and a semi-orthonormal B, $A = 0$ initialization.

5 Experiments

We compare sMuon to regular LoRA with AdamW, per-factor Muon (LoRA with Muon applied to each adapter weight B and A individually), LoRA-Muon (Cesista et al., 2026), and Riemannion (Bogachev et al., 2026). We evaluated on standard commonsense (Clark et al., 2018; 2019; Zellers et al., 2019; Mihaylov et al., 2018; Bisk et al., 2020; Sakaguchi et al., 2021) and coding (Chen et al., 2021; Austin et al., 2021) benchmarks after training with SFT. We experimented with four models, Llama-3.2-3B (Grattafiori et al., 2024), Qwen2.5-3B (Qwen Team, 2025), DeepSeek-V2-Lite (DeepSeek-AI, 2024), and Moonlight-16B-A3B (Liu et al., 2025). The first two models are dense models; the last two are sparse MoE models. The Moonlight model is of particular note because it was pretrained with Muon, whereas the others are all AdamW-pretrained. We also tested the methods in a pretraining setting using ReLoRA (Lialin et al., 2023) which periodically merges and refreshes the adapter. We tuned the learning rate for each method and model, with example learning rate sweeps for the commonsense tasks shown in Appendix D.

Table 1: Per-eval accuracies (%) grouped by base model, after LoRA fine-tuning with different optimizers. Each column is an optimizer (best run by validation loss). The best optimizer in each row is bolded, and cells are shaded from best (green) to worst (white).

Eval	AdamW	Per-factor Muon	LoRA-Muon	Riemannion	sMuon (ours)
Qwen2.5-3B					
Piqa	81.7	82.1	81.7	**82.7**	81.7
Winogrande	**62.3**	61.2	61.4	61.0	61.5
OBQA	78.4	**79.2**	76.0	77.4	77.0
Hellaswag	**81.7**	78.7	76.9	77.6	80.3
BoolQ	**67.7**	66.5	65.6	**67.7**	67.4
Arc-C	83.3	**83.6**	82.9	83.3	**83.6**
MMLU	64.2	64.9	**65.0**	64.9	64.9
Llama-3.2-3B					
Piqa	**77.9**	77.1	74.2	76.9	76.6
Winogrande	51.0	52.0	53.8	**54.9**	52.4
OBQA	67.2	68.4	63.4	**69.4**	66.8
Hellaswag	63.7	**64.9**	58.9	64.3	63.3
BoolQ	63.1	63.4	63.8	**64.1**	63.3
Arc-C	**67.9**	66.2	60.5	64.9	63.2
MMLU	54.4	54.8	52.7	**54.9**	54.4
DeepSeek-V2-Lite					
Piqa	**79.2**	76.7	78.2	75.5	**79.2**
Winogrande	58.2	58.8	**59.7**	55.8	58.2
OBQA	68.0	**70.8**	69.0	61.2	**70.8**
Hellaswag	74.4	71.3	71.2	65.4	**76.4**
BoolQ	64.6	64.1	64.6	64.7	**65.2**
Arc-C	**70.6**	64.9	**70.6**	67.2	67.6
MMLU	**55.4**	54.7	53.1	54.1	53.5
Moonlight-16B-A3B					
Piqa	84.7	84.5	85.6	84.4	**86.2**
Winogrande	59.4	61.2	63.3	64.0	**65.8**
OBQA	81.0	81.2	81.2	78.6	**82.4**
Hellaswag	77.9	80.5	84.7	82.3	**85.9**
BoolQ	68.4	67.1	**69.8**	68.5	69.5
Arc-C	85.6	81.9	84.3	83.3	**87.0**
MMLU	66.1	65.9	65.6	**66.2**	64.5

While our results are both model- and eval-dependent, sMuon performs strongly overall, especially when combined with the Moonlight-16B-A3B base model, a Muon-pretrained model (Tables 1 and 2). sMuon also achieved the joint-best validation loss in the ReLoRA pretraining setting (Table 3). We attribute performance gains, particularly over AdamW and per-factor Muon, to more accurately approximating the Muon objective.

Experiment details. LoRA-Muon and Riemannion were chosen as baselines in addition to the simpler AdamW and per-factor Muon baselines as these are the most similar to our method. Unfortunately, we could not find implementations for LoRA-Muon or Riemannion, so we implemented these ourselves. For Riemannion and our method, we use an $A = 0$ and B orthonormal initialization, and for the remaining methods we used a standard initialization ($B = 0$, A random Gaussian). We scale the learning rate to approximately match AdamW RMS for all Muon optimizers (for details see App. A.1). For all experiments, we used weight decay 0.01, momentum 0.9, inserted LoRA adapters into all

Table 2: Code eval accuracies (%) grouped by base model, after LoRA fine-tuning with different optimizers. Each column is an optimizer (best run by validation loss). The best optimizer in each row is bolded, and cells are shaded green (best) to white (worst).

Eval	AdamW	Per-factor Muon	LoRA-Muon	Riemannion	sMuon (ours)
Qwen2.5-3B					
HumanEval	**54.9**	20.1	43.3	31.1	51.8
HumanEval+	**48.8**	17.1	39.0	25.6	43.9
MBPP	65.9	**69.3**	68.0	65.9	62.7
MBPP+	57.1	**60.6**	58.7	56.9	53.4
Llama-3.2-3B					
HumanEval	22.0	18.3	29.9	**30.5**	29.9
HumanEval+	19.5	16.5	25.0	26.8	**27.4**
MBPP	43.1	35.7	42.1	**46.3**	45.2
MBPP+	35.4	30.4	34.1	38.1	**38.6**
DeepSeek-V2-Lite					
HumanEval	**36.6**	29.3	22.0	23.2	26.8
HumanEval+	**28.7**	23.8	18.3	19.5	23.2
MBPP	52.6	**53.7**	48.4	49.5	51.3
MBPP+	**45.5**	45.0	43.4	43.7	44.7
Moonlight-16B-A3B					
HumanEval	49.4	**57.9**	53.7	40.2	54.9
HumanEval+	42.7	**48.8**	47.0	35.4	**48.8**
MBPP	**71.7**	71.2	71.2	68.8	71.2
MBPP+	60.1	60.1	**61.4**	58.7	60.3

Table 3: Final validation losses after ReLoRA pretraining for different optimizers, along with their optimal learning rate (the optimal LR minimises the validation loss at the final logged step). The best optimizer is bolded and cells are shaded green (best) to white (worst).

	AdamW	Per-factor Muon	LoRA-Muon	Riemannion	sMuon (ours)
Final val. loss (↓)	3.83	3.70	3.68	**3.66**	**3.66**
Optimal LR	0.001	0.006	0.001	0.0006	0.0006

hidden layers (except the router for models with MoE). We used GH200 for all experiments except to produce Table 4 for which we used A100. We used bf16 to store weights and optimizer states, but used fp32 to compute the step for Riemannion, sMuon, and LoRA-Muon.

For the ReLoRA (Lialin et al., 2023) experiments, we started with the modded-nanogpt codebase at 'record 8' (Jordan et al., 2024a), which trains a 162M parameter transformer from scratch (using features including QK-norm, ReLU^2 activations) on the FineWeb dataset (Penedo et al., 2024), and evaluates on a 10M token validation set. We initialize LoRA adapters with rank 64, and train for a total of 3000 steps; every 500 steps we merge the adapters into the base weights and reinitialize the adapters; we use a cosine decay learning rate schedule, though after each merge+reinitialization, we also reset the learning rate to zero and warm back up to the target learning rate over 50 steps (a.k.a. jagged cosine schedule). We swept learning rates in 3e-4, 6e-4, 1e-3, 3e-3, 6e-3, 1e-2.

For the HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) coding evals, we fine-tuned for 3 epochs on completing Python functions from the CodeAlpaca dataset (Chaudhary, 2023). For the MMLU (Hendrycks et al., 2021) eval, we fine-tuned on a 50/50 mixture of FLAN (Longpre et al., 2023) and MMLU training subset. For the remaining evals, we trained on a mixture of FLAN and CommonsenseQA (Talmor et al., 2019), and training subset for each eval. All SFT evals were

Table 4: All Muon optimizers are slightly slower than AdamW, but Riemannion adds huge overhead at larger rank. We show mean time for LoRA adapter optimizer step (ms, 2 s.f.), as well as the proportion (%) of the total step time. Figures were obtained while training a 12 layer GPT model with $d_{\text{model}} = 768$. We ran these timings on an A100, calculating the mean time over 25 steps, with different ranks $r \in \{2^4, 2^5, 2^6, 2^7\}$, batch size 128, 4 gradient accumulation steps.

Step time, ms (% total) ↓	$r = 16$	$r = 32$	$r = 64$	$r = 128$
AdamW	0.66 (0.047%)	0.59 (0.041%)	0.65 (0.046%)	0.78 (0.054%)
Per-factor Muon	1.4 (0.10%)	1.6 (0.11%)	2.0 (0.14%)	3.6 (0.25%)
LoRA-Muon	4.5 (0.32%)	4.8 (0.33%)	6.3 (0.44%)	10 (0.70%)
Riemannion	90 (6.0%)	300 (17%)	700 (33%)	1700 (54%)
sMuon (ours)	8.7 (0.62%)	8.9 (0.62%)	12 (0.80%)	20 (1.4%)
Fwd+bwd, ms	1400	1432	1429	1459

0-shot. For SFT training runs, we used $\alpha = 32$, $r = 16$, gradient clipping (clipping at 1), and we used roughly 2^{16} tokens per batch.

Muon-based LoRA is strongest on the Muon-pretrained model. We originally hypothesized that Muon-based LoRA methods might perform better with Moonlight, since it is a Muon-pretrained model. We do find some evidence for this when aggregating over all benchmarks in Tables 1 and 2: AdamW only achieves top performance on one of the eleven tasks, while sMuon is best for six. Some methods are markedly below others on certain tasks (e.g. per-factor Muon with Llama-3.2-3B on HumanEval, Table 2). This could be due to optimizer instabilities or noise in the benchmarks or selection procedure.

Benefits of Muon pretraining remain with ReLoRA. The ReLoRA experiment (Table 3) removes optimizer mismatch entirely by training from scratch. The separation between optimizer families is clear. Every Muon-based method achieves lower validation loss than LoRA+AdamW. One explanation for the strong Muon performance is amplification of the small but real signal in the long tail of the gradient's spectrum (Wang et al., 2025).

Efficiency gain from the matmul-only implementation. Table 4 compares step times for the different optimizers. AdamW is the fastest; the only computation required by AdamW is element-wise, meaning that data communication is likely to be the largest overhead. Per-factor Muon is the next fastest, because computation is dominated by two $\mathrm{msign}(\cdot)$ computations (that can be batched). Both LoRA-Muon and sMuon require more matmuls than per-factor Muon, but the compute time is still negligible compared to the forward+backward pass time per step of ~ 1.4s. On the other hand, computing the Riemannion step is much slower, adding $\sim 50\%$ overhead to the forward+backward pass when $r = 64$, and even longer for $r = 128$. The computation is slow because Riemannion requires thin QRs and $2r \times 2r$ SVDs, which add overhead for larger r. Note that the retraction step in Riemannion specifically requires truncated SVD, which cannot be replaced by a more efficient Newton-Schulz algorithm. We found that compiling the core computation (i.e. the matmuls), and batching the computation leads to big efficiency gains with all the Muon methods except Riemannion.

6 Conclusion

We considered optimizing low-rank adapters using an approximate form of Muon. The resulting step is available in closed form, is invariant to the adapter parameterization $(B, A) \mapsto (BS, S^{-1}A)$ and is cheap, costing $O(r^2(d_1 + d_2) + r^3)$ time and $O(r \max(d_1, d_2))$ memory, using less optimizer state than LoRA+AdamW. Our implementation, sMuon, has empirically flatter spectra than per-factor Muon and out-performs other methods on the Moonlight Muon-pretrained model. In ReLoRA pretraining every Muon-based method beats AdamW, with sMuon and Riemannion the strongest. In general, by respecting the geometry of the full layer, rather than treating B and A as unrelated matrices, we often benefit, and there is little extra cost.

AI USE STATEMENT

In this work, we used generative AI tools for the following required-disclosure tasks: assist in the writing of proofs, implement methods, provide critical ingredients for proving mathematical claims, clean and reformat dataset. We have not used generative AI tools for the following tasks: generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses design or provide feedback on research methodology or experiments, assist with translation, support qualitative and thematic data analysis, or interpret results. In addition, we used AI for the follow recommended-disclosure tasks: sourcing/searching for information, edit a research paper to improve readability, identify relevant literature.

We have reviewed all AI-assisted work, including evaluating LLM-generated research ideas for potential plagiarism through a manual literature survey, verifying LLM-generated code for correctness, and manually verifying mathematical proofs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper concerns the generic problem of optimising neural networks, which is applicable to all domains, and we therefore do not feel any specific ethical concerns need to be highlighted beyond those that apply to the field in general.

REPRODUCIBILITY STATEMENT

All mathematical results in this paper are provided with proofs and assumptions, either in the main text or in the appendix. We also have provided our experiment code to aid reproducibility. Additional SFT details and pretraining curves are in Appendices E and F.

REFERENCES

Noah Amsel, David Persson, Christopher Musco, and Robert M. Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jeremy Bernstein. Deriving muon, 2025. URL `https://jeremybernste.in/writing/deriving-muon`.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024a.

Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. *arXiv preprint arXiv:2410.21265*, 2024b.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. arXiv:1911.11641.

Vladimir Bogachev, Vladimir Aletov, Alexander Molozhavenko, Denis Bobkov, Vera Soboleva, Aibek Alanov, and Maxim Rakhuba. Lora meets riemannion: Muon optimizer for parametrization-independent low-rank adapters. In *International Conference on Learning Representations*, volume 2026, pp. 82190–82215, 2026.

Franz Louis Cesista, Katherine Crowson, Cédric Simal, and Stella Biderman. Lora-muon: Spectral steepest descent on the low-rank manifold. *arXiv preprint arXiv:2606.12921*, 2026.

Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. `https://github.com/sahil280114/codealpaca`, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Nikhil Ghosh, Tetiana Parshakova, and Robert M Gower. Polora: A preconditioned orthogonalized lora optimizer. *arXiv preprint arXiv:2607.17620*, 2026.

Luc Giraud, Julien Langou, Miroslav Rozložník, and Jasper van den Eshof. Rounding error analysis of the classical gram-schmidt orthogonalization process. *Numerische Mathematik*, 2005.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Chuan He, Zhanwang Deng, and Zhaosong Lu. Low-rank orthogonalization for large-scale matrix optimization with applications to foundation model training. *arXiv preprint arXiv:2509.11983*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2009.03300.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Paul Janson, Edouard Oyallon, and Eugene Belilovsky. Stabilizing native low-rank llm pretraining. *arXiv preprint arXiv:2602.12429*, 2026.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, Boza Vlado, Jiacheng You, Franz Cesista, Braden Koszarsky, and Grad62304977. modded-nanogpt: Speedrunning the NanoGPT baseline. `https://github.com/KellerJordan/modded-nanogpt`, 2024a.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 2024b.

Changmin Kang, Jihun Yun, Baekrok Shin, Yeseul Cho, and Chulhee Yun. Uniform spectral growth and convergence of muon in lora-style matrix factorization. *arXiv preprint arXiv:2602.06385*, 2026.

Kimi Team. Kimi k2: Open agentic intelligence, 2025. URL `https://arxiv.org/abs/2507.20534`.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. ReLoRA: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, and Adam Roberts. The Flan collection: Designing data and methods for effective instruction tuning. *International Conference on Machine Learning (ICML)*, 2023. arXiv:2301.13688.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1260.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank riemannian optimization. In *International Conference on Learning Representations*, volume 2025, pp. 7907–7936, 2025.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024. arXiv:2406.17557.

Xingyu Qu, Peigeng Huang, and Samuel Horvath. Can muon fine-tune adam-pretrained models? *arXiv preprint arXiv:2605.10468*, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 2021. arXiv:1907.10641.

Ishaan Shah, Anthony M Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, et al. Practical efficiency of muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of NAACL-HLT*, 2019. arXiv:1811.00937.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Shuche Wang, Fengzhuo Zhang, Jiaxiang Li, Cunxiao Du, Chao Du, Tianyu Pang, Zhuoran Yang, Mingyi Hong, and Vincent YF Tan. Muon outperforms adam in tail-end associative memory learning. *arXiv preprint arXiv:2509.26030*, 2025.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Taming momentum: Rethinking optimizer states through low-rank approximation. *arXiv preprint arXiv:2602.24283*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Jui-Nan Yen, Si Si, Zhao Meng, Felix Yu, Sai Surya Duvvuri, Inderjit S Dhillon, Cho-Jui Hsieh, and Sanjiv Kumar. LoRA done RITE: Robust invariant transformation equilibration for LoRA optimization. In *International Conference on Learning Representations*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, July 2019.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned lora for fine-tuning foundation models. *arXiv preprint arXiv:2402.02347*, 2024.

Jack Zhang, Noah Amsel, Berlin Chen, and Tri Dao. Gram newton-schulz, 2026. URL https://dao-ailab.github.io/blog/2026/gram-newton-schulz/.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

A Other algorithmic details

We discuss some pre-existing techniques that we incorporated into sMuon.

A.1 Match AdamW RMS

Inspired by Liu et al. (2025), we adopt a learning rate adjustment. The full-parameter scaling is $0.2\sqrt{\max(d_{\text{in}}, d_{\text{out}})}$, with the overall update, including weight decay, becoming

$$x \leftarrow x - \lambda\eta x - 0.2\eta\sqrt{\max(d_{\text{in}}, d_{\text{out}})} \cdot \delta x. \tag{22}$$

To match this in the LoRA setting, we used

$$x \leftarrow x - \lambda\eta x - 0.2\eta\sqrt{d_{\text{in}} d_{\text{out}}/r} \cdot \delta x. \tag{23}$$

We loosely justify this heuristic: assume the weight update $\delta\Delta W$ is orthogonalized. The rank of $\delta\Delta W$ is at most $2r$ (it can be written as the difference of two rank r matrices), and therefore $\|\delta\Delta W\|_F^2 \le 2r$. So $\|\delta\Delta W\|_{\text{rms}} \le \sqrt{2r/(d_{\text{in}} d_{\text{out}})}$. We absorb the $\sqrt{2}$ factor, and use a scale of $0.2\sqrt{d_{\text{in}} d_{\text{out}}/r}$.

A.2 Split weight decay

If we apply weight decay to B and A separately, this gives different dynamics compared to full-parameter training. Concretely,

$$\begin{aligned} & A_{t+1} = (1-\eta\lambda)A_t - \eta\,\delta A, \quad B_{t+1} = (1-\eta\lambda)B_t - \eta\,\delta B \\ \implies \delta\Delta W &= \big[(1-\eta\lambda)B_t - \eta\,\delta B\big]\big[(1-\eta\lambda)A_t - \eta\,\delta A\big] - B_t A_t \\ &= -\big[1-(1-\eta\lambda)^2\big]B_t A_t - \eta(1-\eta\lambda)\big(B_t\,\delta A + \delta B\,A_t\big) + O(\eta^2), \end{aligned}$$

which is not recognizable in terms of full-parameter dynamics. We adopt the split weight decay of Cesista et al. (2026) to preserve the first-order weight decay dynamics. Setting $s = \sqrt{1-\lambda\eta}$, LoRA factors update as,

$$A_{t+1} = s\,A_t - \tfrac{\eta}{s}\,\delta A, \tag{24}$$

$$B_{t+1} = s\,B_t - \tfrac{\eta}{s}\,\delta B. \tag{25}$$

Expanding the resulting weight change $\delta\Delta W = B_{t+1}A_{t+1} - B_t A_t$ gives,

$$\delta\Delta W = \big(sB_t - \tfrac{\eta}{s}\,\delta B\big)\big(sA_t - \tfrac{\eta}{s}\,\delta A\big) - B_t A_t \tag{26}$$

$$= s^2 B_t A_t - \eta(\delta B\,A_t + B_t\,\delta A) + \tfrac{\eta^2}{s^2}\,\delta B\,\delta A - B_t A_t \tag{27}$$

$$= (s^2-1)B_t A_t - \eta(\delta B\,A_t + B_t\,\delta A) + \tfrac{\eta^2}{s^2}\,\delta B\,\delta A \tag{28}$$

$$= \underbrace{-\lambda\eta\,\Delta W_t}_{\text{decay term}} \underbrace{-\,\eta(\delta B\,A_t + B_t\,\delta A)}_{\text{gradient term}} + O(\eta^2) \tag{29}$$

which we identify as having the dynamics of decoupled weight decay (Loshchilov & Hutter, 2017).

A.3 Invariance to parameterization

In practice, we orthogonalize a momentum rather than a raw gradient. The natural choice for momentum accumulation is $M_A \leftarrow \beta M_A + (1-\beta)G_A$ and $M_B \leftarrow \beta M_B + (1-\beta)G_B$, and to reconstruct the effective momentum by substituting them for G_A, G_B in the equation factorizing H,

$$\bar{H}_t = (B_t^\dagger)^\top M_{A,t} + (I - P_{B_t})\,M_{B,t}(A_t^\dagger)^\top. \tag{30}$$

Here, $G_B = G_{\Delta W} A^\top$ and $G_A = B^\top G_{\Delta W}$ are the gradients of the adapter matrices B and A respectively. Consider two training runs, $(B_t,\ A_t)$ and (B'_t, A'_t), that differ only in their parameterizations at each step,

$$(B'_t, A'_t) = (B_t S_t,\ S_t^{-1} A_t). \tag{31}$$

Since $\Delta W'_t = B'_t A'_t = B_t A_t = \Delta W_t$, we want the optimizer to take equivalent steps throughout training, regardless of S_t. For this, it is necessary that $\bar{H}'_t = \bar{H}_t$.

Without transporting momentum buffers The adapter gradients in the primed run are $G'_{A,k} = (B'_k)^\top G_{\Delta W,k} = S_k^\top G_{A,k}$ and $G'_{B,k} = G_{\Delta W,k}(A'_k)^\top = G_{B,k} S_k^{-\top}$. Unrolling the EMA, we have,

$$M'_{A,t} = \sum_{k\le t} (1-\beta)\beta^{t-k}\, S_k^\top G_{A,k}, \qquad M'_{B,t} = \sum_{k\le t} (1-\beta)\beta^{t-k}\, G_{B,k} S_k^{-\top}. \tag{32}$$

Thus, the 'primed' run's effective momentum is,

$$\bar{H}'_t = \sum_{k\le t} (1-\beta)\beta^{t-k} \Big[(B_t^\dagger)^\top\, S_t^{-\top} S_k^\top\, G_{A,k} \;+\; (I - P_{B_t})\, G_{B,k}\, S_k^{-\top} S_t^\top\, (A_t^\dagger)^\top \Big], \tag{33}$$

which differs from $\bar{H}_t$ by the factors $S_t^{-\top} S_k^\top$ and $S_k^{-\top} S_t^\top$. These are the identity only when $S_k = S_t$, so in general $\bar{H}'_t \neq \bar{H}_t$. **With momentum transport** We fix the issue by re-deriving the buffers from $\bar{H}$ against the factors as they stand at the end of the step,

$$M_A \leftarrow B^\top \bar{H}, \qquad M_B \leftarrow \bar{H} A^\top. \tag{34}$$

Geometrically, this projects $\bar{H}$ onto the space in which our updates, $\delta B\, A + B\, \delta A$, exist. The momenta at the next step are then,

$$M_{A,t+1} = B_{t+1}^\top \big(\beta \bar{H}_t + (1-\beta) G_{\Delta W,t+1}\big), \tag{35}$$

$$M_{B,t+1} = \big(\beta \bar{H}_t + (1-\beta) G_{\Delta W,t+1}\big) A_{t+1}^\top. \tag{36}$$

Assuming $\bar{H}'_t = \bar{H}_t$, we have $M'_{A,t+1} = (B'_{t+1})^\top \big(\beta \bar{H}_t + (1-\beta) G_{\Delta W,t+1}\big) = S_{t+1}^\top M_{A,t+1}$ and similarly $M'_{B,t+1} = M_{B,t+1} S_{t+1}^{-\top}$. By further substitution,

$$\begin{aligned}
\bar{H}'_{t+1} &= \big(S_{t+1}^{-1} B_{t+1}^\dagger\big)^\top S_{t+1}^\top M_{A,t+1} + (I - P_{B_{t+1}})\, M_{B,t+1} S_{t+1}^{-\top} \big(A_{t+1}^\dagger S_{t+1}\big)^\top \\
&= (B_{t+1}^\dagger)^\top \underbrace{S_{t+1}^{-\top} S_{t+1}^\top}_{I} M_{A,t+1} + (I - P_{B_{t+1}})\, M_{B,t+1} \underbrace{S_{t+1}^{-\top} S_{t+1}^\top}_{I} (A_{t+1}^\dagger)^\top \\
&= \bar{H}_{t+1}.
\end{aligned}$$

So long as we project $\bar{H}$ into the momentum buffers at every step, the two runs will orthogonalize the same matrix, and therefore the realized steps will agree.

B Alternative stable algorithm

See Alg. 3, a mathematically equivalent but more stable alternative to Alg. 2 from the main text. In particular, we perform two orthogonalization steps (Giraud et al., 2005) to help preserve orthogonality to machine precision, and include relative jitter to help stability for larger r. Another difference between Alg. 2 and 3 is that we use slightly different U_2, V_2. Alg. 3 uses $U_2 = \mathrm{msign}(\tilde{Y})$, where $Y = \tilde{Y} S_A^2$, and $V_2 = \mathrm{msign}(\tilde{Z})$, where $Z = \tilde{Z} S_B^2$. This is possible because the property we need is $\mathrm{col}(Y) = \mathrm{col}(\tilde{Y})$, and similar for Z/$\tilde{Z}$, which is true when S_B and S_A are invertible (we assume this to be the case).

For an exact projector P, $(I-P)^2 = I - P$, so repeating the projection leaves the exact result unchanged while reducing orthogonalization error in floating-point arithmetic. Constructing the bases before inverse-Gram scaling preserves their spans under the assumptions above. For either Gram matrix D, the jitter $\epsilon \max(\mathrm{tr}(D)/r, 1)$ scales with its mean eigenvalue, with a floor of ϵ that prevents the regularization from vanishing at zero initialization.

One can also consider additional interventions, such as replacing $\mathrm{msign}(X) = X(X^\top X)^{-1/2}$ for $X \in \mathbb{R}^{d\times r}$ with X `invroot`$(X^\top X)$. The latter is slightly faster for small r (since $X^\top X \in \mathbb{R}^{r\times r}$), but we found the benefits to be small in practice.

Algorithm 3: Stable sMuon

Input: Same as Alg. 1; jitter $\epsilon > 0$ (default to 10^{-4})

// Inverse roots

$D_B \leftarrow B^\top B$

$\epsilon_B \leftarrow \epsilon \max(\mathrm{tr}(D_B)/r, 1)$

$S_B \leftarrow \texttt{invroot}(D_B + \epsilon_B I)$

$D_A \leftarrow AA^\top$

$\epsilon_A \leftarrow \epsilon \max(\mathrm{tr}(D_A)/r, 1)$

$S_A \leftarrow \texttt{invroot}(D_A + \epsilon_A I)$

// Bases

$X \leftarrow S_B^2 M_A$

$V_1 \leftarrow \mathrm{msign}(A^\top)$

$\tilde{Y} \leftarrow M_B - B\big(S_B^2(B^\top M_B)\big)$

$\tilde{Y} \leftarrow \tilde{Y} - B\big(S_B^2(B^\top \tilde{Y})\big)$ // two orthog. steps

$U_2 \leftarrow \mathrm{msign}(\tilde{Y}), \quad Y \leftarrow \tilde{Y}\, S_A^2$

$\tilde{Z} \leftarrow M_A^\top - V_1(V_1^\top M_A^\top)$

$V_2 \leftarrow \mathrm{msign}(\tilde{Z} - V_1(V_1^\top \tilde{Z}))$ // two orthog. steps

$C \leftarrow \begin{bmatrix} S_B(M_A V_1) & S_B(M_A V_2) \\ (U_2^\top Y)(AV_1) & 0 \end{bmatrix}$

$\Omega \leftarrow \mathrm{msign}(C)$

// Step direction

$P \leftarrow \frac{1}{2}\big(AV_1 + (AV_1)^\top\big)$ // symmetric matrix

$T \leftarrow \frac{1}{2}\big(PS_A^2 + (PS_A^2)^\top\big)$ // symmetric matrix

$\delta A \leftarrow S_B(\Omega_{11} V_1^\top + \Omega_{12} V_2^\top)$

$\delta B \leftarrow U_2\, \Omega_{21}\, T$

// Update parameters

$s \leftarrow \sqrt{1 - \lambda\eta}$

$B' \leftarrow sB - \frac{\eta}{s}\delta B$

$A' \leftarrow sA - \frac{\eta}{s}\delta A$

// Project momentum

$M_A \leftarrow (B'^\top B)X + (B'^\top Y)A$

$M_B \leftarrow B(XA'^\top) + Y(AA'^\top)$

$B \leftarrow B'$

$A \leftarrow A'$

C Comparison of Muon-based LoRA optimizers

Every method that combines Muon with LoRA has the same obstruction: Muon requires a step along $\mathrm{msign}(G_{\Delta W})$, but a gradient step for a LoRA adapter is constrained inside the set $\mathcal{T} = \{Z : (I - P_B)Z(I - P_A) = 0\}$. The ideal step $\mathrm{msign}(G_{\Delta W})$ is not in general within this set, thus different methods make different approximations. We compare the different methods in Table 5.

Table 5: Comparison of the Muon-based LoRA optimizers, all written in the convention $\Delta W = BA$ of this paper. $\Pi_{\mathcal{T}}(\cdot)$ is the orthogonal projector onto the realizable step directions.

Method	$\delta\Delta W$	Momentum	Weight decay on ΔW	Expensive operations
Per-factor Muon (Qu et al., 2026; Kang et al., 2026)	$-\eta\big(\operatorname{msign}(M_B)A + B\ \operatorname{msign}(M_A)\big) + O(\eta^2)$	Standard per-factor EMAs, i.e. accumulate G_B in M_B ($M_B \leftarrow \beta M_B + (1-\beta)G_B$), likewise for M_A	Decoupled, per factor: $\Delta W \mapsto (1-\lambda\eta)^2\Delta W$	2 orthogonalizations
Spectron (Janson et al., 2026)	As above with η replaced by $\rho = \eta/(\|B\|_2+\|A\|_2+1)$; only $\|\delta\Delta W\|_2 \le \eta$ is controlled, no direction matched	Standard per-factor EMAs	Decoupled, per factor	2 orthogonalizations, 2 power iterations to estimate $\|B\|_2, \|A\|_2$
LoRA-Muon (Cesista et al., 2026)	$-\frac{\eta}{2}\big(\operatorname{msign}(P_B G_{\Delta W}) + \operatorname{msign}(G_{\Delta W} P_A)\big) + O(\eta^2)$	Standard per-factor EMAs	Split: let $s = \sqrt{1-\lambda\eta}$, and $A \leftarrow sA - \eta/s\,\delta A$ (same for B). Overall gives $-\lambda\eta\,\Delta W + O(\eta^2)$ (Sec. A.2)	The same 2 orthogonalizations, 2 $r \times r$ inverse square roots of $(B^\top B)^{-1/2}$ and $(AA^\top)^{-1/2}$
Riemannion (Bogachev et al., 2026)	$\operatorname{SVD}_r\big(\Delta W - \eta\,\Pi_{\mathcal{T}}(\operatorname{msign}(H))\big) - \Delta W$	$B^\top B = I$; let $Q_A = \operatorname{qr}(A^\top).Q$. Accumulate $(I - BB^\top)G_{\Delta W}Q_A$ in M_B, and $B^\top G_{\Delta W}$ in M_A. Let $H = BM_A + M_B Q_A^\top$. After retraction, using new B, Q_A, set $M_B = (I - BB^\top)HQ_A$, $M_A = B^\top H$	Prior to retraction, A is decayed via $-\eta\gamma\,A$; B is not decayed.	4 thin QRs, 2 SVDs ($2r \times 2r$)
Ours	$-\eta(I - P_B)\operatorname{msign}(H)P_A - \eta P_B \operatorname{msign}(H) + O(\eta^2)$	Standard per-factor EMAs, but let $H = (B^\dagger)^\top M_A + (I - P_B)M_B(A^\dagger)^\top$. After the gradient step, set $M_B = HA^\top$, $M_A = B^\top H$	Split	2 $r \times r$ inverse square roots, 4 orthogonalizations ($d \times r$, worst case)

D LEARNING RATE CURVES

We show learning rate curves for the commonsense runs in Figures 2 (overall commonsense accuracy) and 3 (val. loss). We swept different learning rates for different optimizers, since we found that the optimal learning rate varies between optimizers. The accuracy figures shown in the main text are chosen by taking the optimal learning rate according to loss on a held-out validation set.

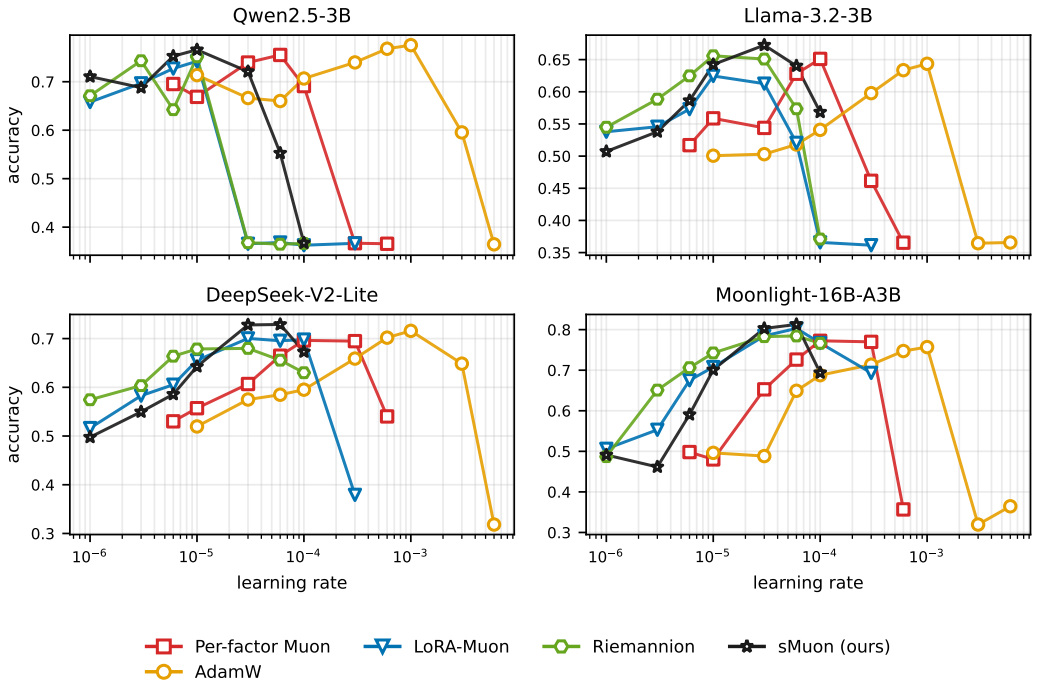

Figure 2: Common-sense accuracy against learning rate for each base model. Each curve is one optimizer; accuracy is the overall common-sense benchmark accuracy at the final training step.

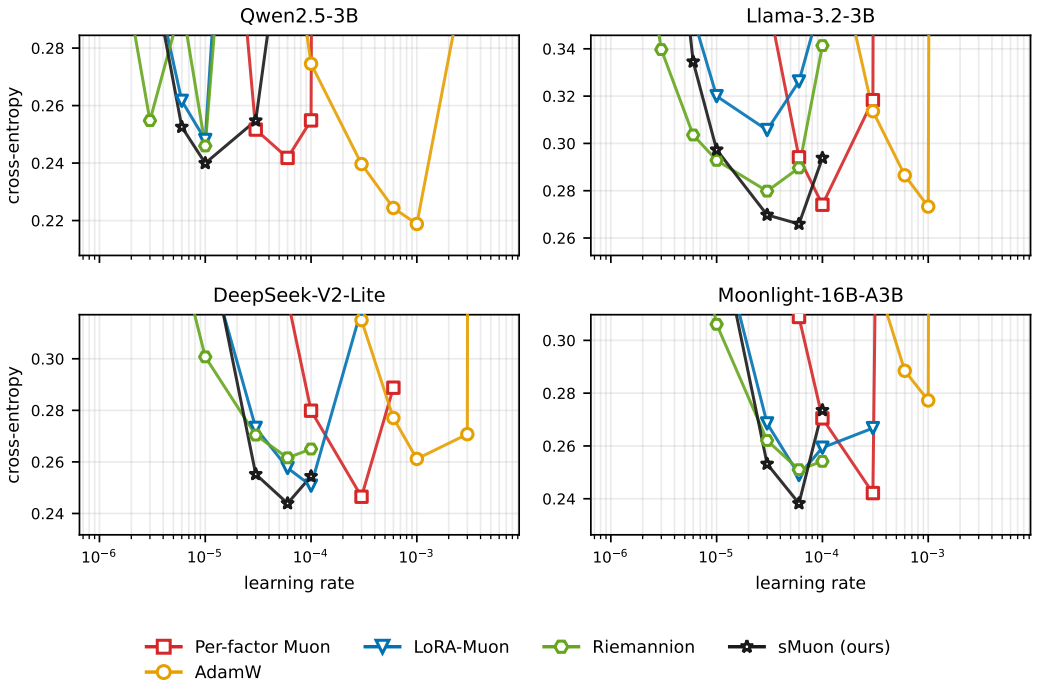

Figure 3: Validation loss against learning rate for each base model, over the same runs as Figure 2.

E SFT DATA AND EVALUATION DETAILS

Commonsense and MMLU training used 60 and 100 optimizer steps, respectively. All SFT runs used a cosine learning-rate schedule with linear warmup over the first 10% of scheduled steps. AdamW used $(\beta_1, \beta_2) = (0.9, 0.999)$. We used one seed per SFT sweep.

The commonsense training and validation-loss mixtures are given in Table 6. The loader counts examples after formatting and length filtering, taking fewer if a source is exhausted. FLAN examples are sampled from Open-Orca/FLAN, specifically the `t0_zsopt_data`, `t0_fsopt_data`, `t0_zsnoopt_data`, and `t0_fsnoopt_data` subsets (parts 0 and 1), shuffled together. We use debiased WinoGrande, ARC-Challenge rather than ARC-Easy, and the 'main' configuration of OpenBookQA. For MMLU SFT, we sample 30,000 examples each from FLAN and the auxiliary training split of cais/mmlu, with 500 validation examples from each. These validation-loss mixtures are distinct from the benchmark splits used to measure final accuracy. Commonsense and MMLU evaluations are 0-shot, selecting the answer label with the highest next-token logit. Commonsense accuracy is measured on the benchmarks' validation splits, and MMLU accuracy on its full test split.

Table 6: Requested example counts for commonsense SFT mixtures, before sequence packing.

Source	Training	Validation loss
FLAN	20,000	250
CommonsenseQA	5,000	125
BoolQ	2,000	50
PIQA	2,000	50
HellaSwag	2,000	50
ARC-Challenge	1,000	25
OpenBookQA	2,000	50
WinoGrande	2,000	50
Total	36,000	650

For coding, we train on Python functions from sahil2801/CodeAlpaca-20k, with 100 validation examples. All code evaluations use greedy completion. HumanEval+ and MBPP+ use the same generated solutions as HumanEval and MBPP, respectively, with expanded test suites.

F RELORA PRETRAINING CURVES

Figure 4 shows the validation-loss curves for the runs summarized in Table 3, using each optimizer's best learning rate according to final validation loss.

F.1 REDUCED-PRECISION IMPLEMENTATION

The main experiments used fp32 for the sMuon step calculation. We also implemented a bf16 version, retaining fp32 for the inverse-root routines while using bf16 for the other operations. Lower-precision inverse-root computation is possible (Zhang et al., 2026), but we leave this extension to future work. Figure 5 compares the fp32 and bf16 versions on the ReLoRA pretraining task. The curves remain close over the plotted interval and reach similar final losses despite multiple adapter resets, supporting the viability of bf16 in this experiment.

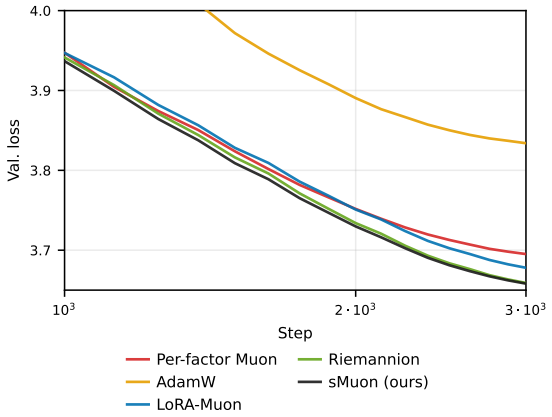

Figure 4: Validation loss during ReLoRA pretraining of a 162M-parameter GPT-2-style model, using the best run for each optimizer by final validation loss.

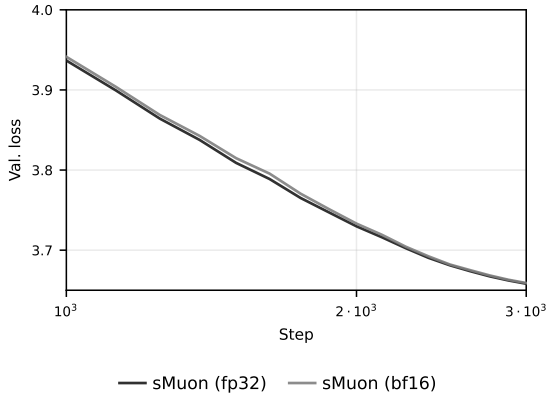

Figure 5: Validation loss for fp32 and bf16 sMuon during ReLoRA pretraining; the bf16 version retains fp32 inverse-root computation.