ROTATED MANIFOLD OPTIMIZATION FOR LOW-RANK ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a novel optimizer for low-rank adaptation (LoRA) that explicitly incorporates the gauge symmetry of low-rank factorization. Our optimizer extends recent matrix optimizers for full-parameter training to the manifold of fixed-rank matrices by interpreting them as normalization under a rotated basis. We show how rotation and normalization can be integrated with the fixed-rank manifold efficiently. Our optimizer converges faster to lower held-out loss and achieves better or comparable downstream performance on both supervised finetuning and reinforcement learning tasks.

1 INTRODUCTION

Large language models (LLMs) based on the transformer architecture (Vaswani et al., 2017) have achieved remarkable success in recent years (Brown et al., 2020; Touvron et al., 2023; Guo et al., 2025). These models typically adopt a pretraining-finetuning paradigm, where large-scale pretraining equips a model with general capabilities and subsequent finetuning adapts it to specific downstream applications. However, as LLMs grow to billions of parameters, full-parameter finetuning becomes expensive in both computation and memory. Such a challenge has motivated the development of parameter-efficient finetuning (PEFT) methods (Houlsby et al., 2019), which only update a small fraction of all parameters. Among these approaches, low-rank adaptation (LoRA) (Hu et al., 2022) has emerged as a particularly effective method and has been widely adopted. LoRA freezes the pretrained weights and parametrizes the weight updates as products of low-rank matrices, substantially reducing the number of trainable parameters and the memory footprint, while maintaining competitive performance.

Compared with full-parameter training, LoRA exhibits a unique geometric structure induced by its low-rank factorization. Specifically, the low-rank matrices have infinitely many reparameterizations that represent the same weight update, since the model's output depends on these low-rank factors only through their product. This invariance, commonly referred to as gauge symmetry, suggests that optimization should respect the underlying geometry of the weight space of the LoRA parametrization. Motivated by this observation, we revisit the optimization of LoRA and aim to explicitly incorporate its gauge symmetry into the design of the optimization algorithm.

We design our optimizer on the manifold of fixed-rank matrices, aiming to optimize the product of the low-rank factors directly rather than each factor separately. We extend the recent advances in matrix optimizers (Gong et al., 2026; Jordan et al., 2024; Vyas et al., 2025) to the fixed-rank manifold by interpreting their update rules as normalization under a rotated basis. Our optimizer applies row and column normalization in a conditioning frame defined by the left singular vectors of the tangent momentum, projects the resulting direction back onto the tangent space, and retracts to the fixed-rank manifold. By exploiting the low-rank structure of tangent vectors, all operations can be performed using thin factors without materializing full weight-sized matrices. We refer to our proposed optimizer as RoM (Rotated Manifold Optimization). We evaluate RoM on supervised finetuning and reinforcement learning tasks. RoM achieves substantially lower held-out loss and better or comparable downstream task performance compared with other LoRA optimizers, with only modest additional computational overhead.

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.

2 BACKGROUND

We briefly review the rotated optimization framework (Gong et al., 2026) and LoRA (Hu et al., 2022), and then introduce the basic concepts of Riemannian optimization. Notations defined in this section will be used throughout the paper. We discuss related work in Appendix B.

2.1 ROTATED OPTIMIZATION

Recent matrix optimizers (Jordan et al., 2024; Vyas et al., 2025) can be viewed through a rotated optimization framework (Gong et al., 2026). Given a matrix-valued momentum $\boldsymbol{M} \in \mathbb{R}^{m\times n}$, the framework first expresses $\boldsymbol{M}$ in a rotated coordinate system defined by a rotation matrix $\boldsymbol{R} \in \mathrm{SO}(m)$, applies a normalization function h in the rotated coordinates, and then rotates the result back to the original coordinates. The resulting update of the weight matrix $\boldsymbol{W}$ takes the form:

$$\Delta \boldsymbol{W} \propto -\boldsymbol{R} h\big(\boldsymbol{R}^\top \boldsymbol{M}\big)\,. \tag{1}$$

2.2 LOW-RANK ADAPTATION (LORA)

LoRA (Hu et al., 2022) freezes the pretrained weight matrix $\boldsymbol{W}_0 \in \mathbb{R}^{m\times n}$ and parametrizes the weight adaptation $\boldsymbol{W}_{\text{adp}}$ as the product of two low-rank factors:

$$\boldsymbol{W}_{\text{eff}} = \boldsymbol{W}_0 + \boldsymbol{W}_{\text{adp}} = \boldsymbol{W}_0 + \boldsymbol{A}\boldsymbol{B}^\top,\ \ \boldsymbol{A} \in \mathbb{R}^{m\times r},\ \ \boldsymbol{B} \in \mathbb{R}^{n\times r} \tag{2}$$

with $r \ll \min(m, n)$. $\boldsymbol{W}_0$ denotes the pretrained weight matrix and is kept frozen throughout LoRA finetuning. The loss function ℓ can thus be written as $\ell(\boldsymbol{A}, \boldsymbol{B})$.

Note that, given the pretrained weight $\boldsymbol{W}_0$, the output of the model depends only on the product of $\boldsymbol{A}$ and $\boldsymbol{B}^\top$. Therefore, there exist many reparametrizations of $\boldsymbol{A}$ and $\boldsymbol{B}$ that keep ℓ unchanged:

$$\ell(\boldsymbol{A}, \boldsymbol{B}) = \ell(\boldsymbol{A}\boldsymbol{Q}^{-1}, \boldsymbol{B}\boldsymbol{Q}^\top), \quad \forall\, \boldsymbol{Q} \in \mathrm{GL}(r) \tag{3}$$

where $\mathrm{GL}(r)$ denotes the set of all $r \times r$ invertible matrices. In this paper, we propose an optimizer that explicitly incorporates the gauge symmetry of the LoRA parametrization (Eq. 3) into its design.

2.3 RIEMANNIAN OPTIMIZATION

Let $\mathcal{M}$ be a smooth manifold (Boumal, 2023). We denote its tangent space at $\boldsymbol{w} \in \mathcal{M}$ by $\mathrm{T}_{\boldsymbol{w}}\mathcal{M}$. Intuitively, $\mathrm{T}_{\boldsymbol{w}}\mathcal{M}$ is a linear approximation of $\mathcal{M}$ locally at $\boldsymbol{w}$, and consists of all the velocities of smooth curves on $\mathcal{M}$ passing through $\boldsymbol{w}$. We use $\mathrm{T}\mathcal{M}$ to denote the tangent bundle, i.e., $\mathrm{T}\mathcal{M} = \{(\boldsymbol{w}, \boldsymbol{v}) \mid \boldsymbol{w} \in \mathcal{M}, \boldsymbol{v} \in \mathrm{T}_{\boldsymbol{w}}\mathcal{M}\}$. A retraction $\Phi : \mathrm{T}\mathcal{M} \to \mathcal{M},\ (\boldsymbol{w}, \boldsymbol{v}) \mapsto \Phi_{\boldsymbol{w}}(\boldsymbol{v})$ is a smooth map such that it defines a smooth curve $t \mapsto \Phi_{\boldsymbol{w}}(t\boldsymbol{v})$ on $\mathcal{M}$ that satisfies $\Phi_{\boldsymbol{w}}(\boldsymbol{0}) = \boldsymbol{w}$ and $\frac{d}{dt}\Phi_{\boldsymbol{w}}(t\boldsymbol{v})|_{t=0} = \boldsymbol{v}$.

We consider minimizing an objective $f(\boldsymbol{w})$ over $\mathcal{M}$. Since $\mathcal{M}$ is generally not a vector space, the notion of a gradient depends on how we measure the lengths and angles of tangent vectors. We therefore equip each tangent space $\mathrm{T}_{\boldsymbol{w}}\mathcal{M}$ with an inner product $\langle \boldsymbol{u}, \boldsymbol{v}\rangle_{\boldsymbol{w}}$. Intuitively, this inner product specifies the local geometry of $\mathcal{M}$ around $\boldsymbol{w}$ (i.e., how to compute local norms and angles). If the assignment $\boldsymbol{w} \mapsto \langle \cdot, \cdot\rangle_{\boldsymbol{w}}$ varies smoothly with $\boldsymbol{w}$, it is called a Riemannian metric, and $\mathcal{M}$ is referred to as a Riemannian manifold.

The Riemannian gradient of f at $\boldsymbol{w}$, denoted by $\mathrm{grad} f(\boldsymbol{w})$, is defined as:

$$\langle \mathrm{grad} f(\boldsymbol{w}), \boldsymbol{v}\rangle_{\boldsymbol{w}} = \frac{d}{dt} f(c(t))\Big|_{t=0}, \quad \forall\, \boldsymbol{v} \in \mathrm{T}_{\boldsymbol{w}}\mathcal{M} \tag{4}$$

where $c : \mathbb{R} \to \mathcal{M}$ is any smooth curve on $\mathcal{M}$ satisfying $c(0) = \boldsymbol{w}$ and $\frac{d}{dt}c(t)|_{t=0} = \boldsymbol{v}$. In practice, it is convenient to work with an extension $\bar{f}$ defined in an open neighborhood $\bar{\mathcal{M}}$ of $\mathcal{M}$ such that f is the restriction of $\bar{f}$ to $\mathcal{M}$, i.e., $f = \bar{f}|_{\mathcal{M}}$. Let $\nabla \bar{f}(\boldsymbol{w})$ denote the Euclidean gradient of $\bar{f}$ at $\boldsymbol{w}$, computed in the ambient space (e.g., via automatic differentiation in PyTorch). When $\mathcal{M}$ is an embedded submanifold of $\mathbb{R}^n$ and inherits the Euclidean inner product, i.e., $\langle \boldsymbol{u}, \boldsymbol{v}\rangle_{\boldsymbol{w}} = \boldsymbol{u}^\top \boldsymbol{v}$, we have the following relationship between $\mathrm{grad} f(\boldsymbol{w})$ and $\nabla \bar{f}(\boldsymbol{w})$:

$$\mathrm{grad} f(\boldsymbol{w}) = \mathrm{Proj}_{\boldsymbol{w}}\left(\nabla \bar{f}(\boldsymbol{w})\right) \tag{5}$$

where $\mathrm{Proj}_{\boldsymbol{w}} : \mathbb{R}^n \to \mathrm{T}_{\boldsymbol{w}}\mathcal{M}$ denotes the orthogonal projection onto $\mathrm{T}_{\boldsymbol{w}}\mathcal{M}$. Finally, given a step size $\eta > 0$, a basic Riemannian gradient descent update takes the form:

$$\boldsymbol{w}_{k+1} = \Phi_{\boldsymbol{w}_k}\big(-\eta\,\mathrm{grad} f(\boldsymbol{w}_k)\big)\,. \tag{6}$$

3 METHOD

We propose Rotated Manifold Optimization (RoM), which directly optimizes the LoRA weight adaptation $\boldsymbol{W}_{\text{adp}}$ on the manifold of fixed-rank matrices. For notational simplicity, throughout this section we write $\boldsymbol{W} := \boldsymbol{W}_{\text{adp}}$. We first introduce the geometry of the fixed-rank manifold in Section 3.1 and describe how to perform Riemannian optimization efficiently by exploiting the low-rank structure. Then in Section 3.2 we extend the rotated optimization framework (Eq. 1) to the manifold.

3.1 FIXED-RANK MANIFOLD

We consider the set of all $m \times n$ matrices with rank r:

$$\mathcal{M}_r = \{\boldsymbol{W} \in \mathbb{R}^{m\times n} \mid \mathrm{rank}(\boldsymbol{W}) = r\} \tag{7}$$

Each matrix $\boldsymbol{W} \in \mathcal{M}_r$ admits an SVD decomposition $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times r}$ have orthonormal columns, i.e., $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}_r$, and $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma})$ is a square diagonal matrix with positive diagonal entries $\boldsymbol{\sigma} \in \mathbb{R}^r_+$. We parametrize $\boldsymbol{W}$ using $\boldsymbol{U}$, $\boldsymbol{\sigma}$ and $\boldsymbol{V}$. Compared with the $\boldsymbol{A}$ and $\boldsymbol{B}$ factors in standard LoRA (Eq. 2), we only need negligible additional memory to store $\boldsymbol{\sigma}$.

Tangent space Given the factors $\boldsymbol{U}$ and $\boldsymbol{V}$, the tangent space at $\boldsymbol{W}$ takes the form:

$$\begin{aligned}\mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r = \Big\{\boldsymbol{Z} \in \mathbb{R}^{m\times n} \;\Big|\; & \boldsymbol{Z} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top + \boldsymbol{U}_p\boldsymbol{V}^\top + \boldsymbol{U}\boldsymbol{V}_p^\top,\\ & \boldsymbol{S} \in \mathbb{R}^{r\times r},\ \boldsymbol{U}_p \in \mathbb{R}^{m\times r},\ \boldsymbol{V}_p \in \mathbb{R}^{n\times r},\\ & \boldsymbol{U}^\top\boldsymbol{U}_p = \boldsymbol{0},\ \boldsymbol{V}^\top\boldsymbol{V}_p = \boldsymbol{0}\Big\}.\end{aligned} \tag{8}$$

Eq. 8 implies that, although a tangent vector $\boldsymbol{Z} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$ has shape $m \times n$ in the ambient space, it can actually be represented and stored using smaller components $\boldsymbol{S} \in \mathbb{R}^{r\times r}$, $\boldsymbol{U}_p \in \mathbb{R}^{m\times r}$ and $\boldsymbol{V}_p \in \mathbb{R}^{n\times r}$. A tangent vector $\boldsymbol{Z}$ defined as in Eq. 8 can be further written in the compact form:

$$\boldsymbol{Z} = \begin{bmatrix}\boldsymbol{U} & \boldsymbol{U}_p\end{bmatrix}\begin{bmatrix}\boldsymbol{S} & \boldsymbol{I}_r\\ \boldsymbol{I}_r & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{V} & \boldsymbol{V}_p\end{bmatrix}^\top \tag{9}$$

which indicates $\mathrm{rank}(\boldsymbol{Z}) \le 2r$ for any $\boldsymbol{Z} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$.

Orthogonal projection Let $\mathcal{L} : \mathcal{M}_r \to \mathbb{R}$ denote the objective to minimize and $\bar{\mathcal{L}}$ a smooth extension of $\mathcal{L}$ to an open neighborhood of $\mathcal{M}_r$ in the ambient space. Consistent with Section 2.3, we denote the Euclidean gradient and the Riemannian gradient at $\boldsymbol{W}$ by $\nabla\bar{\mathcal{L}}(\boldsymbol{W})$ and $\mathrm{grad}\,\mathcal{L}(\boldsymbol{W})$ respectively. We equip $\mathcal{M}_r$ with the Riemannian metric induced by the Euclidean inner product. Then we derive the orthogonal projection of $\nabla\bar{\mathcal{L}}(\boldsymbol{W}) \in \mathbb{R}^{m\times n}$ onto the tangent space $\mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$:

$$\mathrm{Proj}_{\boldsymbol{W}}\big(\nabla\bar{\mathcal{L}}(\boldsymbol{W})\big) = \nabla\bar{\mathcal{L}}(\boldsymbol{W}) - (\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^\top)\,\nabla\bar{\mathcal{L}}(\boldsymbol{W})\,(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^\top)\,, \tag{10}$$

and we have $\mathrm{grad}\,\mathcal{L}(\boldsymbol{W}) = \mathrm{Proj}_{\boldsymbol{W}}\big(\nabla\bar{\mathcal{L}}(\boldsymbol{W})\big)$. Because $\mathrm{Proj}_{\boldsymbol{W}}\big(\nabla\bar{\mathcal{L}}(\boldsymbol{W})\big) \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$, it admits the decomposition in Eq. 8. Its three components can be expressed in terms of $\nabla\bar{\mathcal{L}}(\boldsymbol{W}), \boldsymbol{U}, \boldsymbol{V}$ as:

$$\begin{aligned}\boldsymbol{S} &= \boldsymbol{U}^\top\,\nabla\bar{\mathcal{L}}(\boldsymbol{W})\,\boldsymbol{V}\,,\\ \boldsymbol{U}_p &= \nabla\bar{\mathcal{L}}(\boldsymbol{W})\,\boldsymbol{V} - \boldsymbol{U}\boldsymbol{S}\,,\\ \boldsymbol{V}_p &= \nabla\bar{\mathcal{L}}(\boldsymbol{W})^\top\,\boldsymbol{U} - \boldsymbol{V}\boldsymbol{S}^\top\,.\end{aligned} \tag{11}$$

Practical Riemannian gradient computation While the relationship between the Euclidean gradient and the Riemannian gradient at $\boldsymbol{W}$ is straightforward (Eq. 5), naively applying the projection Eq. 10 to the Euclidean gradient $\nabla\bar{\mathcal{L}}(\boldsymbol{W})$ would require materializing a full $m \times n$ matrix, which is memory intensive. We demonstrate how to compute the Riemannian gradient efficiently via a lightweight modification to the auto-differentiation, without instantiating $\nabla\bar{\mathcal{L}}(\boldsymbol{W})$ explicitly.

Consider $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ as the weight adaptation of a linear layer that takes $\boldsymbol{X} \in \mathbb{R}^{N\times m}$ as input and outputs $\boldsymbol{Y} \in \mathbb{R}^{N\times n}$, where N represents the effective batch size (i.e., the total number of tokens in the mini-batch). In the standard auto-differentiation framework, $\boldsymbol{X}$ is saved during the forward pass for gradient computation, and when the gradient with respect to $\boldsymbol{Y}$, denoted by $\boldsymbol{G}_Y = \partial\bar{\mathcal{L}}/\partial\boldsymbol{Y} \in \mathbb{R}^{N\times n}$ is back-propagated to this layer, $\nabla\bar{\mathcal{L}}(\boldsymbol{W})$ is computed as $\nabla\bar{\mathcal{L}}(\boldsymbol{W}) = \boldsymbol{X}^\top\boldsymbol{G}_Y$. Note that to compute $\operatorname{grad}\mathcal{L}(\boldsymbol{W})$, we only need $\boldsymbol{S}$, $\boldsymbol{U}_p$ and $\boldsymbol{V}_p$ in Eq. 11, which depend on $\nabla\bar{\mathcal{L}}(\boldsymbol{W})\boldsymbol{V}$ and $\nabla\bar{\mathcal{L}}(\boldsymbol{W})^\top\boldsymbol{U}$ rather than any standalone $\nabla\bar{\mathcal{L}}(\boldsymbol{W})$. We can compute these two quantities directly without the need to materialize the $m \times n$ matrix $\nabla\bar{\mathcal{L}}(\boldsymbol{W})$:

$$\nabla\bar{\mathcal{L}}(\boldsymbol{W})\boldsymbol{V} = \boldsymbol{X}^\top\Big(\boldsymbol{G}_Y\boldsymbol{V}\Big) \in \mathbb{R}^{m\times r} \qquad \nabla\bar{\mathcal{L}}(\boldsymbol{W})^\top\boldsymbol{U} = \boldsymbol{G}_Y^\top\Big(\boldsymbol{X}\boldsymbol{U}\Big) \in \mathbb{R}^{n\times r} \tag{12}$$

We compute the bracketed products first, so that every matrix multiplication involves an operand with one dimension equal to r, keeping all intermediate matrices low-rank and memory-efficient. We customize the backward propagation (e.g., via PyTorch's backward hooks) to compute $\nabla\bar{\mathcal{L}}(\boldsymbol{W})\boldsymbol{V}$ and $\nabla\bar{\mathcal{L}}(\boldsymbol{W})^\top\boldsymbol{U}$ on the fly as in Eq. 12, and store them for retraction. The memory usage is of the same order as that for computing and storing the gradients of $\boldsymbol{A}$ and $\boldsymbol{B}$ in standard LoRA (Eq. 2).

Retraction Given a tangent update $\boldsymbol{Z} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$, we use the rank-$r$ truncated SVD as the retraction, i.e., $\Phi_{\boldsymbol{W}}(\boldsymbol{Z}) = \mathrm{SVD}_r(\boldsymbol{W} + \boldsymbol{Z})$, which retains the top r singular values of $\boldsymbol{W} + \boldsymbol{Z}$ and their corresponding singular vectors. With the factorization of $\boldsymbol{Z}$ in Eq. 9, $\boldsymbol{W} + \boldsymbol{Z}$ can be written in a compact form that enables efficient computation of the truncated SVD:

$$\boldsymbol{W} + \boldsymbol{Z} = [\boldsymbol{U} \quad \boldsymbol{U}_p]\begin{bmatrix}\boldsymbol{\Sigma} + \boldsymbol{S} & \boldsymbol{I}_r \\ \boldsymbol{I}_r & \boldsymbol{0}\end{bmatrix}[\boldsymbol{V} \quad \boldsymbol{V}_p]^\top . \tag{13}$$

We apply thin QR factorizations $\boldsymbol{U}_p = \boldsymbol{Q}_U\boldsymbol{R}_U$ and $\boldsymbol{V}_p = \boldsymbol{Q}_V\boldsymbol{R}_V$. Then the truncated SVD is only applied to the $2r \times 2r$ matrix $\begin{bmatrix}\boldsymbol{\Sigma} + \boldsymbol{S} & \boldsymbol{R}_V^\top \\ \boldsymbol{R}_U & \boldsymbol{0}\end{bmatrix}$. Let $\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}^\top$ denote the rank-r truncated SVD of this small matrix, then the retracted point can be written as:

$$\Phi_{\boldsymbol{W}}(\boldsymbol{Z}) = ([\boldsymbol{U} \;\; \boldsymbol{Q}_U]\widetilde{\boldsymbol{U}})\widetilde{\boldsymbol{\Sigma}}([\boldsymbol{V} \;\; \boldsymbol{Q}_V]\widetilde{\boldsymbol{V}})^\top \tag{14}$$

which gives a best rank-r approximation of $\boldsymbol{W} + \boldsymbol{Z}$ in Frobenius norm.

Momentum transport A natural extension of the standard (Euclidean) momentum to the manifold suggests taking the moving average of the Riemannian gradients. However, Riemannian gradients of an optimization trajectory live in different tangent spaces and cannot be combined directly. Let $\boldsymbol{M}_k \in \mathrm{T}_{\boldsymbol{W}_k}\mathcal{M}_r$ be the momentum at the k-th step of an optimization trajectory, and let $\operatorname{grad}\mathcal{L}(\boldsymbol{W}_{k+1})$ be the fresh Riemannian gradient at $\boldsymbol{W}_{k+1}$. To get the updated momentum $\boldsymbol{M}_{k+1}$, we first project $\boldsymbol{M}_k$ onto the new tangent space $\mathrm{T}_{\boldsymbol{W}_{k+1}}\mathcal{M}_r$ and then combine the projection of $\boldsymbol{M}_k$ with $\operatorname{grad}\mathcal{L}(\boldsymbol{W}_{k+1})$:

$$\boldsymbol{M}_{k\to k+1} = \mathrm{Proj}_{\boldsymbol{W}_{k+1}}(\boldsymbol{M}_k) \qquad \boldsymbol{M}_{k+1} = \beta\boldsymbol{M}_{k\to k+1} + (1-\beta)\operatorname{grad}\mathcal{L}(\boldsymbol{W}_{k+1}) \tag{15}$$

Similar to the Riemannian gradient, the momentum is also stored using its factorization (Eq. 8), and the transport operation (Eq. 15) remains factorized. A detailed algorithm on Eq. 15 is provided in Algorithm 2 in the appendix.

3.2 Rotated optimization on manifold

Starting from the tangent momentum $\boldsymbol{M} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$ (Eq. 15), we now extend the rotated optimization framework (Eq. 1) to the manifold setting. This introduces two challenges. First, a naive rotation in the ambient space would require an $m \times m$ rotation matrix ($O(m^2)$ memory), destroying the memory efficiency of LoRA. Second, after rotation and normalization, the resulting vector may not remain in the tangent space. We show that both issues can be addressed efficiently by using the rank-$2r$ structure of tangent vectors.

Algorithm 1: Rotated optimization on the fixed-rank manifold

Input: Weight adaptation $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$; momentum $\boldsymbol{M} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$ represented by components $(\boldsymbol{S}, \boldsymbol{U}_p, \boldsymbol{V}_p)$; conditioning frame $\boldsymbol{F} \in \mathbb{R}^{m\times 2r}$ with $\boldsymbol{F}^\top\boldsymbol{F} = \boldsymbol{I}_{2r}$; normalization function h.

Output: Components of the final tangent update: $(\widetilde{\boldsymbol{S}}, \widetilde{\boldsymbol{U}}_p, \widetilde{\boldsymbol{V}}_p)$

// Form thin factors $\boldsymbol{L}, \boldsymbol{R}$ of $\boldsymbol{M}$ from its components ($\boldsymbol{M} = \boldsymbol{L}\boldsymbol{R}^\top$)

$\boldsymbol{L} \leftarrow [\boldsymbol{U}\boldsymbol{S} + \boldsymbol{U}_p \quad \boldsymbol{U}] \in \mathbb{R}^{m\times 2r}$

$\boldsymbol{R} \leftarrow [\boldsymbol{V} \quad \boldsymbol{V}_p] \in \mathbb{R}^{n\times 2r}$

// Express $\boldsymbol{M}$ in the rotated coordinates: $\boldsymbol{C} = \boldsymbol{F}^\top\boldsymbol{M}$

// Note the order of products that avoids $m \times n$ matrix

$\boldsymbol{C} \leftarrow \left(\boldsymbol{F}^\top\boldsymbol{L}\right)\boldsymbol{R}^\top \in \mathbb{R}^{2r\times n}$

// Apply the normalization function h

$\widetilde{\boldsymbol{C}} \leftarrow h(\boldsymbol{C})$

// Map back to the original coordinates $\boldsymbol{F}\widetilde{\boldsymbol{C}}$ and project

// Matrix multiplications are reordered to avoid $m \times n$ matrix

$\boldsymbol{P} \leftarrow \boldsymbol{F}^\top\boldsymbol{U} \in \mathbb{R}^{2r\times r}$

$\boldsymbol{Q} \leftarrow \widetilde{\boldsymbol{C}}\boldsymbol{V} \in \mathbb{R}^{2r\times r}$

$\widetilde{\boldsymbol{S}} \leftarrow \boldsymbol{P}^\top\boldsymbol{Q}$

$\widetilde{\boldsymbol{U}}_p \leftarrow \boldsymbol{F}\boldsymbol{Q} - \boldsymbol{U}\widetilde{\boldsymbol{S}}$

$\widetilde{\boldsymbol{V}}_p \leftarrow \widetilde{\boldsymbol{C}}^\top\boldsymbol{P} - \boldsymbol{V}\widetilde{\boldsymbol{S}}^\top$

Conditioning frame Since the momentum $\boldsymbol{M} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$ has rank at most $2r$, its column space is low-dimensional. Therefore, it suffices to perform the rotation within a $2r$-dimensional subspace containing the column space of $\boldsymbol{M}$, rather than in the full ambient space[1]. Concretely, we construct a conditioning frame $\boldsymbol{F} \in \mathbb{R}^{m\times 2r}$ with orthonormal columns (i.e., $\boldsymbol{F}^\top\boldsymbol{F} = \boldsymbol{I}_{2r}$) for this subspace. The frame $\boldsymbol{F}$ defines the rotated coordinate system, in which $\boldsymbol{M}$ is represented by $\boldsymbol{F}^\top\boldsymbol{M}$. In full-parameter training, the left singular vectors of $\boldsymbol{M}$ have been chosen by several previous works (Jordan et al., 2024; Vyas et al., 2025) to define the rotated coordinate system, but computing them can be expensive. When $\boldsymbol{M} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$, its low-rank structure (Eq. 9) allows us to compute these singular vectors efficiently using two thin QR factorizations and an SVD of a $2r \times 2r$ matrix, analogous to how we compute the retraction in Section 3.1. We therefore use the resulting left singular vectors as the conditioning frame $\boldsymbol{F}$.

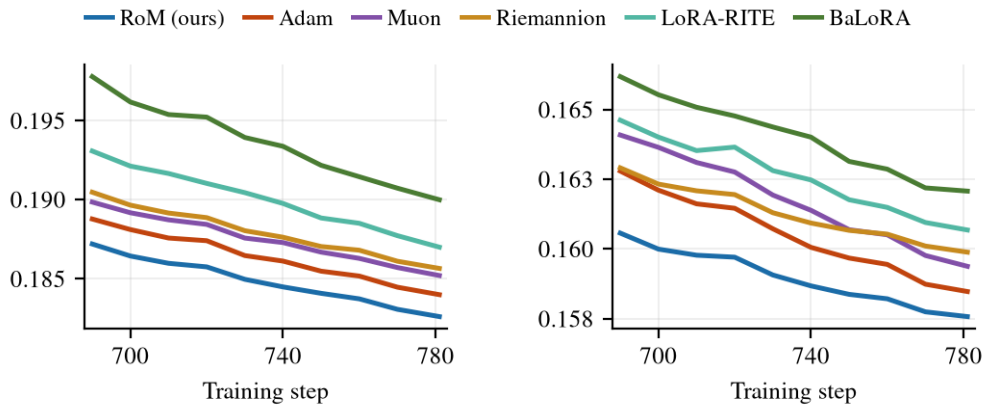
Normalization function We next specify the normalization function h applied to the momentum in the rotated coordinates. Note that the normalization is applied to the reduced representation $\boldsymbol{F}^\top\boldsymbol{M} \in \mathbb{R}^{2r\times n}$, rather than to a full $m \times n$ matrix. Following Scetbon et al. (2025), we adopt a multi-norm normalization that has shown effectiveness in LLM pre-training (Gong et al., 2026). For any matrix $\boldsymbol{G} \in \mathbb{R}^{m\times n}$, h is defined as:

$$h(\boldsymbol{G}) = \underset{\boldsymbol{H}\in\mathbb{R}^{m\times n}}{\arg\max} \langle \boldsymbol{G}, \boldsymbol{H}\rangle \quad \text{s.t.} \quad \max_i \|\boldsymbol{H}_{i,:}\|_2 = 1 \quad \text{and} \quad \max_j \|\boldsymbol{H}_{:,j}\|_2 = 1 \tag{16}$$

Here, $\boldsymbol{H}_{i,:}$ and $\boldsymbol{H}_{:,j}$ denote the i-th row ($1 \le i \le m$) and j-th column ($1 \le j \le n$) of $\boldsymbol{H}$ respectively; $\langle \boldsymbol{G}, \boldsymbol{H}\rangle = \mathrm{tr}(\boldsymbol{G}\boldsymbol{H}^\top)$ denotes the Frobenius inner product; and $\|\cdot\|_2$ denotes the ℓ_2 norm. Eq. 16 generalizes the single-norm steepest-descent formulation of Bernstein & Newhouse (2024) to simultaneous row- and column-norm constraints. In practice, we approximate Eq. 16 by alternating row and column normalization for a small fixed number of iterations K, which we set to 5 throughout all experiments. The full procedure is given in Algorithm 3 in the appendix.

Final update rule Given the conditioning frame $\boldsymbol{F}$ and tangent momentum $\boldsymbol{M} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$, we first represent $\boldsymbol{M}$ in the rotated coordinates as $\boldsymbol{F}^\top\boldsymbol{M}$, apply the normalization function h, and then map the resulting direction back to the original coordinates as $\boldsymbol{F}h(\boldsymbol{F}^\top\boldsymbol{M})$. Although $\boldsymbol{M}$ lies in the tangent space $\mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$, the normalization function h does not, in general, preserve this tangent

[1] Any rotation acting only within the orthogonal complement of $\boldsymbol{M}$'s column space leaves $\boldsymbol{M}$ unchanged.



(a) Llama-3.2-3B (b) Llama-3.1-8B

Figure 1: Validation loss (mean cross-entropy across response tokens) over the last 100 training steps on the math SFT task at rank $r = 16$ (see Appendix C for the $r = 32$ case and additional results across different ranks).

space constraint. We therefore project the resulting direction back onto $\mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$. With a step size $\eta > 0$, the final tangent update $\boldsymbol{\xi} \in \mathrm{T}_{\boldsymbol{W}}\mathcal{M}_r$ is:

$$\boldsymbol{\xi} = -\eta \operatorname{Proj}_{\boldsymbol{W}}\Big(\boldsymbol{F} h\Big(\boldsymbol{F}^\top \boldsymbol{M}\Big)\Big) \tag{17}$$

Importantly, the orthogonal projection in Eq. 17 can be combined with the preceding operations so that $\boldsymbol{F} h(\boldsymbol{F}^\top \boldsymbol{M})$ is never materialized as a full $m \times n$ matrix. We detail the efficient implementation of one step of RoM in Algorithm 1.

4 Experiments

We evaluate RoM on supervised finetuning (SFT) and reinforcement learning (RL) tasks, comparing its optimization and downstream performance against existing LoRA optimizers. We describe the experimental setup in Section 4.1, present the SFT and RL results in Section 4.2 and 4.3, respectively, and further study the contribution of its components and its sensitivity to learning-rate selection in Section 4.4.

4.1 Experimental setup

We apply LoRA adapters $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ to the weight matrices of all linear layers in *both* the attention and MLP modules, while keeping one-dimensional parameters, input embeddings and the output head frozen. We initialize $\boldsymbol{U}$ and $\boldsymbol{V}$ randomly with orthonormal columns, and set $\boldsymbol{\Sigma} = 10^{-3}\boldsymbol{I}_r$. This initialization ensures that each adapter is a valid point on the fixed-rank manifold, while keeping the effective weights close enough to their pretrained values.

We compare RoM with Adam (Kingma & Ba, 2014), Muon (Jordan et al., 2024), LoRA-RITE (Yen et al., 2025), BaLoRA (Castin et al., 2026), and Riemannion (Bogachev et al., 2026). Adam and Muon adopt the conventional two-factor LoRA (Eq. 2) and optimize each factor separately. LoRA-RITE (Yen et al., 2025) uses matrix preconditioners to achieve transformation invariance. BaLoRA (Castin et al., 2026) addresses the parametrization redundancy of LoRA (Eq. 3) through a balanced projection step after each Adam update to enforce $\boldsymbol{A}^\top\boldsymbol{A} = \boldsymbol{B}^\top\boldsymbol{B}$. Riemannion (Bogachev et al., 2026) projects each Muon update onto the tangent space of the fixed-rank manifold.

For SFT, we use the same learning rate schedule for all optimizers: linear warm-up over the first 5% of training steps, followed by linear decay to 10% of the peak learning rate. We find this warm-up schedule particularly beneficial for Adam, improving its performance compared with previously reported results. For RL finetuning, we follow the setup of Liu et al. (2025) and use a constant learning rate. For each optimizer, we tune its learning rate over the same logarithmically spaced grid and expand the grid when necessary to ensure that the best-performing value is bracketed on

Table 1: Math SFT results for the Llama models at rank $r = 16$: final MetaMathQA validation loss ($\times 10^{-3}$) and test accuracies (%). Results are mean $\pm$ standard deviation over three seeds; the pretrained models are evaluated once. Bold marks the best entry and entries whose intervals overlap it; no entry is bold if all intervals overlap.

	Llama-3.2-3B			Llama-3.1-8B		
	MetaMathQA val ↓	GSM8K ↑	MATH ↑	MetaMathQA val ↓	GSM8K ↑	MATH ↑
Pretrained	728.23	18.1	7.3	652.22	41.9	15.5
RoM (ours)	**182.69**±0.21	**65.4**±0.1	**18.2**±0.3	**157.60**±0.06	77.7±1.0	**29.2**±0.3
Adam	184.11±0.14	**64.9**±0.9	**18.0**±0.1	158.70±0.26	77.2±0.6	28.4±0.4
Muon	185.42±0.19	64.3±0.9	**17.8**±0.4	159.45±0.13	77.3±0.9	28.0±0.5
Riemannion	185.79±0.22	64.0±0.7	**17.7**±0.4	159.91±0.13	76.9±0.7	**29.0**±0.6
LoRA-RITE	187.16±0.16	64.5±0.4	17.2±0.5	160.54±0.16	76.6±0.8	28.0±0.7
BaLoRA	190.31±0.28	62.8±1.1	17.3±0.2	161.93±0.12	76.1±0.8	28.0±0.2

both sides. We use a momentum coefficient of 0.9 for all optimizers. For Adam, we set the second-moment coefficient to $\beta_2 = 0.95$. We use no weight decay.

4.2 Supervised finetuning

Mathematical reasoning We finetune Llama-3.2-3B, Llama-3.1-8B (Grattafiori et al., 2024), and Gemma-2-27B (Gemma Team, 2024) at rank $r = 16$ on 100K MetaMathQA examples (Yu et al., 2024) for one epoch, using an effective batch size of 128. We construct a held-out 10K-example validation set and measure validation loss as the mean cross-entropy over response tokens. For each optimizer, we independently select the learning rate that minimizes validation loss and repeat the selected configuration over three seeds. Training for additional epochs did not further reduce the final validation loss. We evaluate downstream performance using zero-shot accuracy on GSM8K (1319 examples) and MATH (5000 examples) with greedy decoding and bfloat16 precision. Figure 1 shows the validation-loss trajectories for the two Llama models. Tables 1 and 2 report the mean and standard deviation of the final validation loss and downstream accuracies for the Llama models and Gemma-2-27B, respectively.

At all three model sizes, RoM achieves the lowest final validation loss consistently, followed by Adam among the baselines[2]. RoM obtains the highest mean accuracy on both downstream benchmarks for the Llama models, although the differences are small relative to variation across seeds. On Gemma-2-27B, RoM yields the highest mean accuracy on MATH, while the standard-deviation intervals on GSM8K overlap across all optimizers. In contrast to these small downstream differences, the validation-loss gaps are much larger than the variation across seeds, showing a clear optimization advantage for RoM .

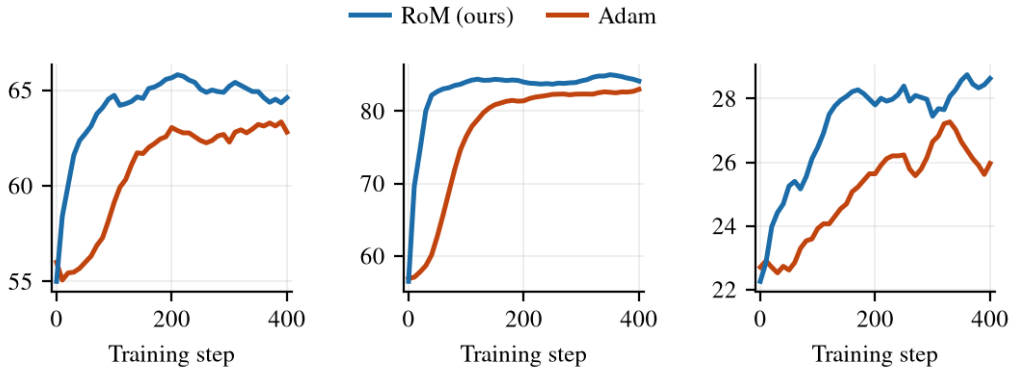
Table 2: Math SFT results for Gemma-2-27B at rank 16. Bold follows the convention in Table 1.

	Math SFT val ↓	GSM8K ↑	MATH ↑
RoM (ours)	**126.19**±0.13	84.2±0.8	**43.4**±0.3
Adam	126.55±0.21	84.1±0.6	42.2±0.5
Muon	127.02±0.29	83.8±0.7	42.6±0.4
Riemannion	127.85±0.20	83.8±0.9	42.2±0.6
LoRA-RITE	129.71±0.13	83.7±0.5	42.7±0.2
BaLoRA	129.05±0.27	84.6±0.9	**43.3**±0.1

Code generation We finetune the Llama-3.1-8B base model on 100K examples of the CodeFeedback dataset (Zheng et al., 2024) using the Alpaca prompt templates (Taori et al., 2023). We compute validation loss as the mean cross-entropy over response tokens on a held-out validation set of 10K examples. Since CodeFeedback sequences are typically longer than those in MetaMathQA, we train for one epoch with a smaller effective batch size of 64. Figure 3 shows the validation loss over the final 100 training steps. RoM achieves the lowest validation loss.

We evaluate the finetuned model on HumanEval (Chen et al., 2021) (164 test examples) and MBPP (Austin et al., 2021) (378 test examples curated by EvalPlus (Liu et al., 2023)) using greedy

[2]Our BaLoRA results use the same training protocol as the other optimizers in our comparison, including the learning-rate schedule and independent learning-rate tuning, and are therefore not directly comparable to those reported by Castin et al. (2026).



(a) MATH-500 (b) GSM8K (c) OlympiadBench

Figure 2: Evaluation accuracy (%) during GRPO finetuning of Qwen2.5-3B with rank $r = 16$, on each of the three held-out benchmarks. The curves are smoothed using a 5-point centered moving average for visualization; higher accuracy is better.

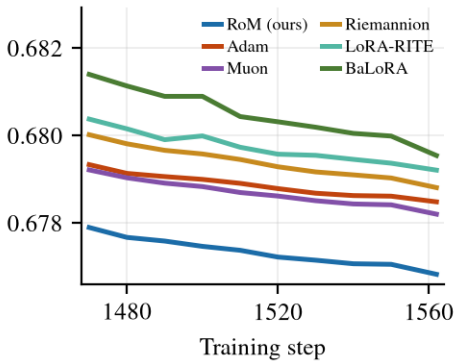


Figure 3: Validation loss over the final 100 training steps on the code SFT task, for Llama-3.1-8B at rank $r = 32$.

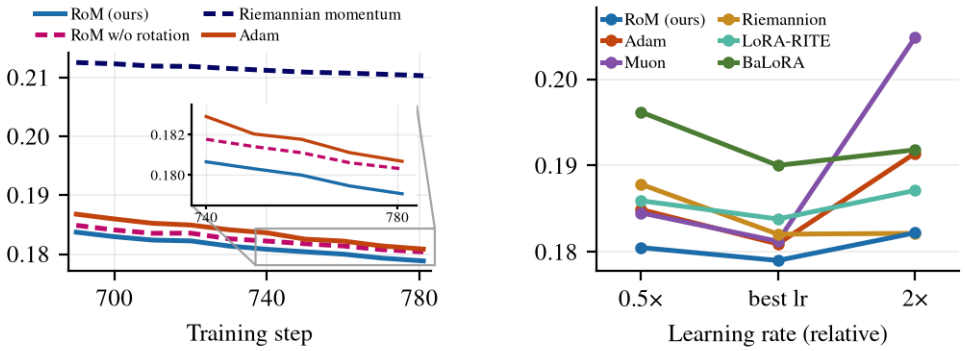
Table 3: Code SFT results for Llama-3.1-8B at rank 32: final validation loss ($\times 10^{-3}$) on CodeFeedback and Pass@1 (%) on HumanEval and MBPP. Results are mean $\pm$ standard deviation over 3 seeds except for the pretrained model. Bold follows the convention in Table 1.

	Code SFT val ↓	HumanEval ↑	MBPP ↑
Pretrained	811.99	42.7	60.6
RoM (ours)	**676.81**±0.04	55.7±2.6	68.5±0.5
Adam	678.50±0.03	55.3±2.9	68.3±0.3
Muon	678.19±0.05	55.3±0.9	68.7±1.0
Riemannion	678.87±0.06	55.7±1.4	68.2±0.4
LoRA-RITE	679.25±0.08	55.5±0.6	68.1±0.6
BaLoRA	679.63±0.15	56.7±1.0	68.6±1.0

decoding with bfloat16 precision and no in-context examples, and report Pass@1. For each optimizer, we train three runs with different random seeds. Table 3 reports the mean and standard deviation of the final validation loss and downstream performance. All optimizers clearly improve Pass@1 over the pretrained model. However, the differences in Pass@1 among optimizers are minor relative to the variation across random seeds, and we therefore do not observe a clear downstream advantage for any optimizer. In contrast, the validation loss gaps substantially exceed the seed variation, with RoM achieving the lowest value. This suggests that RoM's optimization advantage extends to code SFT, although it does not translate into a clear Pass@1 improvement in this setting.

4.3 Reinforcement Learning

We further evaluate RoM on LoRA-based RL finetuning using the VeRL framework (Sheng et al., 2025). Since the other LoRA baselines do not specify an RL training recipe, we focus on comparing RoM against Adam. We train the Qwen2.5-3B base model (Yang et al., 2024) using GRPO (Shao et al., 2024) on the training split of MATH (7.5K problems) with rank $r = 16$. At each rollout step, we sample 8 responses for each of 128 prompts, yielding 1024 sequences per optimization step. We use no KL penalty and train for 400 steps. Following the setup of Liu et al. (2025), we use the bare questions as prompts with no chat template. We evaluate the finetuned model using greedy decoding on three held-out benchmarks: MATH-500 (500 examples), GSM8K test (1319 examples), and OlympiadBench (He et al., 2024) (674 examples). We tune Adam's learning rate on a logarithmic grid and select 3×10^{-6}. We use the same learning rate for RoM without further tuning.



(a) Ablation study

(b) Learning rate sensitivity

Figure 4: Llama-3.2-3B at rank $r = 32$ on the math SFT task. (a) Validation loss over the last 100 training steps, removing parts of RoM. (b) Validation loss at different ratios of best learning rate.

Figure 2 shows evaluation accuracy on the three held-out benchmarks throughout RL finetuning. RoM improves accuracy faster than Adam and reaches higher final accuracy on all three benchmarks. Table 4 reports the final evaluation results over three independent runs. Unlike in SFT, where downstream differences among optimizers are relatively small, RoM shows a clearer advantage over Adam in RL finetuning, while requiring only $1.08\times$ the per-step training time on 4 NVIDIA RTX PRO 6000 GPUs.

Table 4: Final downstream accuracy (%) after GRPO finetuning over three independent runs. Results are mean $\pm$ standard deviation. "Time" denotes the median end-to-end wall clock time per optimizer step, normalized to Adam.

	MATH-500	GSM8K	Olympiad	Time
Pretrained	55.5	56.8	22.0	–
Adam	62.9±0.3	82.7±0.4	25.9±0.8	1.00×
RoM (ours)	**64.9**±0.6	**84.2**±0.4	**29.0**±0.8	1.08×

4.4 Additional analysis

Ablation study We ablate the rotation and normalization components of RoM on the math SFT task using Llama-3.2-3B at rank $r = 32$. We consider RoM without rotation, which applies the normalization function (Eq. 16) directly to the momentum (Eq. 15), and RoM without both rotation and normalization, which we refer to as *Riemannian momentum*. We tune the learning rate for each variant following the same procedure as in Section 4.2. Figure 4a shows the validation loss over the final 100 training steps. Riemannian momentum performs substantially worse than RoM, while adding normalization without rotation reduces the gap and brings its performance close to Adam. Full RoM achieves the lowest validation loss, demonstrating the additional benefit of rotation.

Learning rate sensitivity We evaluate each optimizer at $0.5\times$, $1\times$, and $2\times$ its best learning rate under the same math SFT setting. Figure 4b shows the final validation loss at each learning-rate ratio. The optimizers that account for the symmetry of LoRA's parametrization tend to exhibit less degradation away from their best learning rates than Adam and Muon. RoM maintains the lowest validation loss across all three learning-rate ratios.

5 Conclusion

We presented RoM, a manifold-based optimizer for LoRA that directly optimizes the fixed-rank weight adaptation while avoiding the redundancy of factorized parameterizations. By exploiting the low-rank structure of tangent vectors, RoM enables efficient Riemannian optimization with memory requirement comparable to conventional LoRA optimization. Across supervised finetuning and reinforcement learning experiments, RoM consistently achieves lower validation loss than existing LoRA optimizers, while also improving downstream performance in the RL setting. Ablation experiments further show that both normalization and rotation contribute to its optimization performance.

AI USE STATEMENT

During the preparation of this work, we used LLMs to assist with coding and debugging, to help document and organize experimental progress, and to polish the text and figures of the manuscript. The authors have reviewed and verified all LLM-assisted outputs and take full responsibility for the final content presented in this work.

REFERENCES

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

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 *ICLR*, 2026.

Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023. doi: 10.1017/9781009166164.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Valérie Castin, Kimia Nadjahi, Pierre Ablin, and Gabriel Peyré. Balanced lora: Removing parameter invariance to accelerate convergence. In *ICML*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

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

Wenbo Gong, Javier Zazo, Qijun Luo, Puqian Wang, James Hensman, and Chao Ma. Aro: A new lens on matrix optimization for large models. *arXiv preprint arXiv:2602.09006*, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 2025.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *ICML*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *ACL*, 2024.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

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. In *ICLR*, 2022.

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

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

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *NeurIPS*, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *COLM*, 2025.

Meyer Scetbon, Chao Ma, Wenbo Gong, and Edward Meeds. Gradient multi-normalization for efficient LLM training. In *NeurIPS*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, 2025.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. SOAP: Improving and stabilizing shampoo using adam for language modeling. In *ICLR*, 2025.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

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 *ICLR*, 2025.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *ICLR*, 2024.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned loRA for fine-tuning foundation models. In *ICML*, 2024.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. In *Findings of ACL*, 2024.

A ALGORITHMS

Algorithm 2: Factorized momentum transport on the fixed-rank manifold

Input: Old and new orthonormal frames $\boldsymbol{U}_k, \boldsymbol{V}_k, \boldsymbol{U}_{k+1}, \boldsymbol{V}_{k+1}$; momentum coefficient $\beta \in [0, 1)$; old momentum components $(\boldsymbol{S}_k, \boldsymbol{U}_{p,k}, \boldsymbol{V}_{p,k})$ such that $\boldsymbol{M}_k = \boldsymbol{U}_k \boldsymbol{S}_k \boldsymbol{V}_k^\top + \boldsymbol{U}_{p,k} \boldsymbol{V}_k^\top + \boldsymbol{U}_k \boldsymbol{V}_{p,k}^\top$, with $\boldsymbol{U}_k^\top \boldsymbol{U}_{p,k} = \boldsymbol{0}$ and $\boldsymbol{V}_k^\top \boldsymbol{V}_{p,k} = \boldsymbol{0}$; fresh Riemannian gradient components $(\boldsymbol{S}_{k+1}^{(g)}, \boldsymbol{U}_{p,k+1}^{(g)}, \boldsymbol{V}_{p,k+1}^{(g)})$ at $\boldsymbol{W}_{k+1}$.

Output: Updated momentum components $(\boldsymbol{S}_{k+1}, \boldsymbol{U}_{p,k+1}, \boldsymbol{V}_{p,k+1})$ representing $\boldsymbol{M}_{k+1} = \beta \operatorname{Proj}_{\boldsymbol{W}_{k+1}}(\boldsymbol{M}_k) + (1-\beta)\operatorname{grad} \mathcal{L}(\boldsymbol{W}_{k+1})$.

`// Compute frame overlaps`

$\boldsymbol{C}_U \leftarrow \boldsymbol{U}_k^\top \boldsymbol{U}_{k+1}, \boldsymbol{C}_V \leftarrow \boldsymbol{V}_k^\top \boldsymbol{V}_{k+1}$

`// Compute` $\boldsymbol{\Psi}_V = \boldsymbol{M}_k \boldsymbol{V}_{k+1}$ `(no` $m \times n$ `matrix)`

$\boldsymbol{\Psi}_V \leftarrow \boldsymbol{U}_k\big(\boldsymbol{S}_k \boldsymbol{C}_V + \boldsymbol{V}_{p,k}^\top \boldsymbol{V}_{k+1}\big) + \boldsymbol{U}_{p,k} \boldsymbol{C}_V$

`// Compute` $\boldsymbol{\Psi}_U = \boldsymbol{M}_k^\top \boldsymbol{U}_{k+1}$ `(no` $m \times n$ `matrix)`

$\boldsymbol{\Psi}_U \leftarrow \boldsymbol{V}_k\big(\boldsymbol{S}_k^\top \boldsymbol{C}_U + \boldsymbol{U}_{p,k}^\top \boldsymbol{U}_{k+1}\big) + \boldsymbol{V}_{p,k} \boldsymbol{C}_U$

`// Project onto new tangent space`

$\boldsymbol{S}_{k\to k+1} \leftarrow \boldsymbol{U}_{k+1}^\top \boldsymbol{\Psi}_V$

$\boldsymbol{U}_{p,k\to k+1} \leftarrow \boldsymbol{\Psi}_V - \boldsymbol{U}_{k+1} \boldsymbol{S}_{k\to k+1}$

$\boldsymbol{V}_{p,k\to k+1} \leftarrow \boldsymbol{\Psi}_U - \boldsymbol{V}_{k+1} \boldsymbol{S}_{k\to k+1}^\top$

`// Accumulate with fresh Riemannian gradient`

$\boldsymbol{S}_{k+1} \leftarrow \beta\, \boldsymbol{S}_{k\to k+1} + (1-\beta)\, \boldsymbol{S}_{k+1}^{(g)}$

$\boldsymbol{U}_{p,k+1} \leftarrow \beta\, \boldsymbol{U}_{p,k\to k+1} + (1-\beta)\, \boldsymbol{U}_{p,k+1}^{(g)}$

$\boldsymbol{V}_{p,k+1} \leftarrow \beta\, \boldsymbol{V}_{p,k\to k+1} + (1-\beta)\, \boldsymbol{V}_{p,k+1}^{(g)}$

Algorithm 3: Row and column normalization function h

Input: Reduced representation $\boldsymbol{C} \in \mathbb{R}^{2r\times n}$; number of iterations K.

Output: Normalized representation $\widetilde{\boldsymbol{C}} \in \mathbb{R}^{2r\times n}$.

for $k = 1, \ldots, K$ **do**

  $\boldsymbol{C}_{i,:} \leftarrow \boldsymbol{C}_{i,:} / \|\boldsymbol{C}_{i,:}\|_2$ for $i = 1, \ldots, 2r$

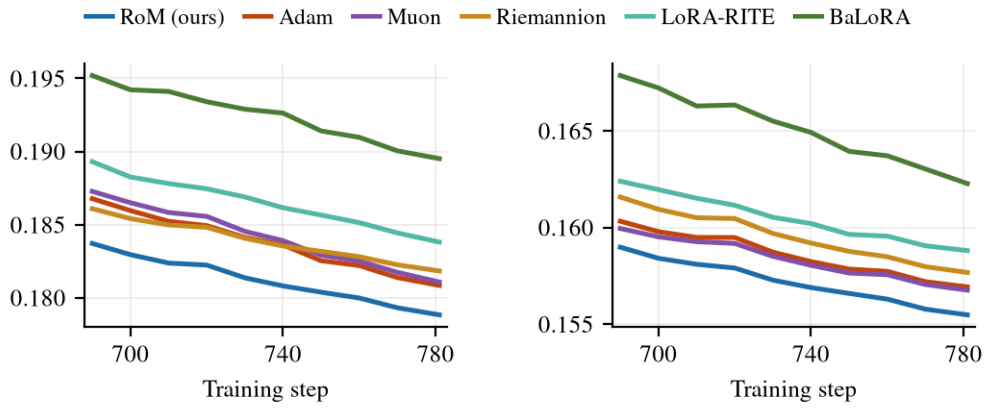
  $\boldsymbol{C}_{:,j} \leftarrow \boldsymbol{C}_{:,j} / \|\boldsymbol{C}_{:,j}\|_2$ for $j = 1, \ldots, n$

return $\widetilde{\boldsymbol{C}} \leftarrow \boldsymbol{C}$

B RELATED WORK

We review the relevant literature on optimization methods for LoRA. Hayou et al. (2024) advocate setting different learning rates for the two low-rank factors to achieve efficient feature learning. Zhang & Pilanci (2024) extend this approach by introducing matrix preconditioners to stabilize feature learning and draw a connection to a Riemannian metric on a quotient manifold. Yen et al. (2025) design preconditioners that make the update to the weight adapter (i.e., $\boldsymbol{W}_{\text{adp}}$) invariant to reparametrizations of low-rank factors. Castin et al. (2026) enforce the condition $\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{B}^\top \boldsymbol{B}$ after each Adam update step via a balanced projection operation and theoretically demonstrate that balanced low-rank factors yield fast convergence. However, their analysis does not take into account the effect of momentum. A concurrent work (Ghosh et al., 2026) extends prior preconditioning approaches by bounding the per-sample loss change. These optimizers compute individual updates for each low-rank factor. The most relevant work to ours is Riemannion (Bogachev et al., 2026), which projects the Muon update onto the tangent space of the fixed-rank manifold. While our approach also adopts the fixed-rank manifold perspective, we develop our optimizer through the rotated optimization framework, with rotation and normalization as its core components, distinguishing it from Riemannion (Bogachev et al., 2026).

C EFFECT OF LORA RANK



(a) Llama-3.2-3B (b) Llama-3.1-8B

Figure 5: Validation loss (mean cross-entropy across response tokens) over the last 100 training steps on the math SFT task at rank $r = 32$, the counterpart of Figure 1.

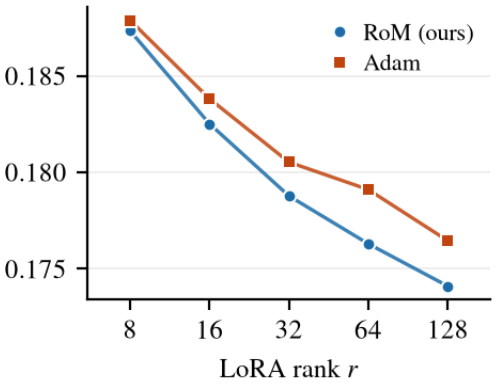


Figure 6: Final validation loss for RoM and Adam across LoRA ranks on math SFT, using Llama-3.2-3B. Learning rate is independently tuned for each optimizer and rank.